



# Universidad de Huelva

## Semandal: semantizando Andalucía

Memoria presentada por  
Ángel Cantó Vicente  
Daniel Albendín Moya

Director: Gonzalo A. Aranda Corral

Huelva, 16 de Abril de 2015.



---

---

## Agradecimientos

---

A nuestras familias por su apoyo y comprensión durante estos años de estudio.

A nuestros profesores por su esfuerzo y dedicación, en especial a Gonzalo A. Aranda y Miguel Ángel Román por su ayuda y atención durante la realización de este proyecto.



---

---

# Índice general

---

<b>Introducción</b>	<b>5</b>
<b>I Extracción de información</b>	<b>9</b>
<b>1. Obtención de municipios y sus datos</b>	<b>11</b>
1.1. Obtención de pueblos y datos a partir de xls . . . . .	12
1.2. Extracción de url de los ayuntamientos . . . . .	12
1.3. Obtención de coordenadas . . . . .	14
1.4. Búsqueda de municipios cercanos . . . . .	15
1.5. Páginas con OpenCMS . . . . .	15
1.6. Página de wikipedia . . . . .	16
1.7. Extracción de datos del Instituto de estadística y cartografía de Andalucía . . . . .	17
1.8. Visualizar la estructura de la web de un municipio . . . . .	18
1.9. Extracción de noticias desde la web de un municipio . . . . .	25
<b>2. Búsqueda de categorías</b>	<b>33</b>

<b>II De información a conocimiento</b>	<b>37</b>
<b>    3. Métodos de clasificación</b>	<b>39</b>
3.1. Selección de palabras clave de una categoría . . . . .	39
3.2. Análisis formal de conceptos . . . . .	44
3.3. Árboles de decisión . . . . .	50
3.4. Sistemas de inducción de reglas . . . . .	58
<b>    4. Motor de clasificación</b>	<b>61</b>
4.1. Sistema experto . . . . .	61
4.2. Entradas del sistema experto . . . . .	62
4.3. Método propuesto . . . . .	62
<b>III Publicación</b>	<b>67</b>
<b>    5. Django</b>	<b>69</b>
5.1. Instalación . . . . .	70
5.2. Creación del proyecto . . . . .	71
5.3. Creación de la base de datos - /polls/models.py . . . . .	72
5.4. Ejecución de código - /polls/views.py . . . . .	76
5.5. Definición de parámetros en el proyecto - /mysite/settings.py . . . . .	77
5.6. Creación de comandos personalizados . . . . .	78
5.7. Uso de Django-extensions . . . . .	79
5.8. Anotaciones . . . . .	79
<b>    6. API</b>	<b>81</b>
6.1. JSON . . . . .	82
6.2. Llamadas . . . . .	83

Índice general	3
<hr/>	
<b>7. Android</b>	<b>85</b>
7.1. Funcionalidades de la aplicación . . . . .	87
7.2. Funcionalidades auxiliares . . . . .	89
7.3. Elementos de navegación . . . . .	91
7.4. Casos de uso . . . . .	93
 <hr/>	
<b>IV Arquitectura de la solución</b>	<b>101</b>
 <hr/>	
<b>8. Servidor</b>	<b>103</b>
8.1. Tareas programas en el servidor . . . . .	103
8.2. Base de datos . . . . .	105
8.3. Clasificación y categorías . . . . .	105
8.4. Usuarios . . . . .	106
 <hr/>	
<b>V Conclusiones</b>	<b>111</b>
 <hr/>	
<b>9. Conclusiones</b>	<b>113</b>
9.1. Obtención de datos . . . . .	113
9.2. Clasificación . . . . .	113
9.3. Android . . . . .	114
9.4. API . . . . .	114
9.5. Django . . . . .	114
 <hr/>	
<b>VI Trabajos futuros</b>	<b>117</b>
9.6. Trabajos futuros . . . . .	119

<b>VII Apéndices</b>	<b>123</b>
<b>A. Base de datos</b>	<b>125</b>
A.1. Noticias y pueblos . . . . .	125
A.2. Clasificación y categorías . . . . .	126
A.3. Usuarios . . . . .	127
A.4. Control . . . . .	128
<b>B. Librerías y software utilizado</b>	<b>129</b>
B.1. Librerías . . . . .	129
B.2. Software . . . . .	131
<b>VIII Bibliografía</b>	<b>133</b>
<b>Bibliografía</b>	<b>135</b>

---

---

# Introducción

---

Actualmente en la web podemos encontrar publicada bastante información sobre municipios españoles y sus ayuntamientos, a veces el acceso a esta información es complicado debido a la diversidad en las estructuras de las páginas web de los distintos municipios ya que hay una gran variedad a la hora de realizar éstas. Existe un amplio número de frameworks que aportan una plantilla y aún así dentro de esa plantilla predefinida por el framework se puede estructurar el contenido de una forma distinta dependiendo del desarrollador. También existen páginas webs que están encargadas a una empresa y esta usa su propia plantilla para mostrar el contenido.

No existe una fuente unificada de información. Si un usuario quisiera acceder a la información de distintos municipios debería consultar varias fuentes siendo él quién realice la búsqueda de los datos.

Debemos dotar a la información de un significado, de una semántica, de una estructura común y adaptar la información extraída de los ayuntamientos a dicha estructura.

La variedad de contenido en una página web al ser tan grande debemos ir paso a paso con lo que iremos analizando la información en distintas etapas.

A parte del análisis y obtención de información, también queremos darle al usuario una forma de acceder a dicha información de una forma simplificada y estructurada, de tal forma que éste pueda consultar algo de la misma forma en cualquier ayuntamiento.

## Proyecto SEMANDAL

*Semandal* nace debido a varias necesidades como por ejemplo.

1. Unificación de datos en una misma fuente. La información de un municipio se puede encontrar en varias fuentes y el acceso a ésta depende del lugar donde esté ubicada. Si tuviésemos una fuente unificada de datos, sería mucho más sencillo el acceso a la misma.
2. Necesidad de jerarquización y clasificación de datos. Los datos podemos encontrarlos, como hemos dicho antes, en distintas fuentes y con una estructura diferente en cada una de ellas. Sería bastante bueno para el posterior tratamiento de la información y el acceso, crear una ontología con los datos obtenidos.
3. Acceso sencillo a dicho contenido. Una vez que tenemos todos estos datos, ofrecemos un sencillo acceso a dicho contenido tanto a usuarios humanos como máquinas.

Para la resolución de estos problemas necesitamos tres partes diferenciadas en el proyecto:

### Extracción de información

Existen bastantes datos de un ayuntamiento en su propia página web, pero también en documentos oficiales de los ministerios del estado o internet. *Semandal* almacenará varios datos de los municipios como su página web, localización, deuda, población, etc ... Debido a la diversidad de los datos que queremos recopilar, usaremos varias fuentes de información. Tenemos que obtener datos de fuentes muy diversas, como Wikipedia, Google o el INE, salvo esta última, todas tienen una forma de extracción de datos propia (Ver figura 1).

### Tratamiento de la información

Aquí es donde realmente *Semandal* pone su nombre en práctica y se realiza la semantización y jerarquización de datos usando técnicas de *aprendizaje automático* pertenecientes al campo de la *inteligencia artificial*. Le damos a la información un significado y la almacenamos de tal forma que esta pueda ser fácilmente recuperada de forma automática por un software, que es lo que usaremos para la publicación de la información.

### Publicación de la información

El último pero no menos importante apartado de nuestro proyecto ya que éste

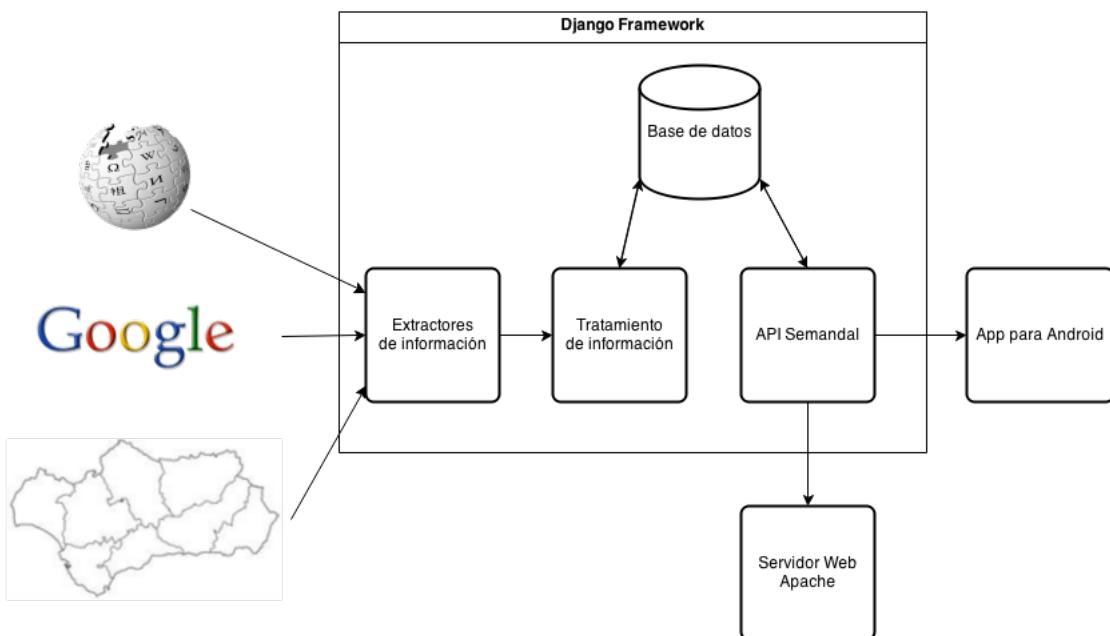


Figura 1: Infraestructura Semandal

será el que verá el usuario final. En este apartado es donde recopilaremos la información que hemos tratado y la mostraremos al usuario de una forma sencilla y concisa de tal forma que sea de fácil acceso tanto para usuarios humanos como para máquinas.

Para dar soporte a las tres partes de nuestro proyecto nace una más. La infraestructura.

**Infraestructura** Las tres partes en las que hemos dividido el proyecto no servirían si no se pudieran sustentar en algo. Debemos crear una infraestructura que sea capaz de soportar las distintas tecnologías que vamos a utilizar en nuestro proyecto.

Además, a medida que se ha ido desarrollando el proyecto, se le han ido añadiendo funcionalidades de tal forma que le ha salido una parte que vimos necesaria.

**Social** Los usuarios podrán interaccionar entre sí además de con los distintos municipios y su contenido.

## Objetivos

Para cada parte del proyecto, salvo la de infraestructura ya que su único objetivo es garantizar el correcto funcionamiento de los distintos componentes del proyecto, existe un objetivo.

**Extracción de la información** Programar un scrapper, programa que extraiga un contenido específico, para cada fuente de información que tenemos.

**Tratamiento de la información** Crear un clasificador para transformar a la información extraída en conocimiento y la creación de una ontología.

**Publicación de la información**

1. Crear una API, conjunto de subrutinas, métodos, y procedimientos que ofrecen un servicio, para que
2. Crear una Aplicación android (APP) que use dicha API.

**Social** Se implementará un sistema de registro de usuarios e inicio de sesión. Así mismo, nuestra aplicación para Android tendrá un sistema donde cada usuario podrá valorar de forma positiva o negativa un comentario.

# **Parte I**

## **Extracción de información**



# Obtención de municipios y sus datos

---

Nuestro proyecto gira en torno a los municipios y queríamos integrar información de estos desde distintas fuentes. Para ello lo que necesitábamos era una lista oficial de municipios y encontramos un *CSV* con todos los municipios españoles a día **01/01/2014** con algunos datos. Con este fichero empezamos a importar los datos a nuestra base de datos para tener la tabla con todos los pueblos para poder realizar búsqueda de otros datos que explicaremos posteriormente. El fichero en cuestión era el **CODMUN14.xls** [www.ine.es/daco/daco42/codmun/codmun14/14codmun.xls](http://www.ine.es/daco/daco42/codmun/codmun14/14codmun.xls) , pero como podemos comprobar los datos de este pueblo son bastante escasos, así que tuvimos que buscar más datos de todos los pueblos españoles. Consultamos la página del INE (Instituto Nacional de Estadística) y encontramos un fichero con la deuda de todos los municipios españoles a fecha **31/12/2013** e incorporamos los datos. También encontramos un fichero con los habitantes, y la superficie, etc ...

En una primera aproximación, decidimos que íbamos a centrarnos en realizar una extracción de datos desde las páginas web de los ayuntamientos, por tanto faltaban datos necesarios para nosotros que no estaban en los xls citados anteriormente cómo la url de la página web de cada pueblo, el más importante ya que sin este no podríamos obtener la información accesible únicamente desde éstas, las coordenadas y la página de wikipedia.

Como posible implementación extra, decidimos usar un listado de provincias españolas, para poder tener a qué provincia pertenece un municipio. Para la APP esta información no sirve de mucho, (únicamente para decir en qué provincia está un pueblo) pero desde la API se puede llegar a consultar los pueblos de una provincia, y sus

correspondientes datos.

Para la obtención de estos datos, usaremos varias APIs de Google y realizaremos varios scripts que funcionaran como wrappers. A continuación explicaremos es funcionamiento del estos scripts.

## 1.1. Obtención de pueblos y datos a partir de xls

Como explicamos en la introducción, el listado de pueblos y algunos datos lo obtenemos de tres xls. Por tanto debemos realizar parseadores de xls para obtener los datos y llevarlos a la base de datos por lo que debemos tener creadas las tablas en ésta y a partir de ahí podemos empezar a crear registros o actualizarlos.

La extracción de datos no se hizo de forma incremental empezando por los xls y posteriormente los datos extraidos con scripts de internet, si no que creamos la tabla y fueron surgiendo propuestas de atributos para los municipios almacenados, por tanto tuvimos que cambiar en varias ocasiones la estructura de la base de datos.

Cuando creamos la base de datos en primer lugar, se guardaron los nombres de los pueblos y se asoció una correspondiente ID a cada uno. Ese ID no venía en los distintos documentos así que tuvimos que hacer una búsqueda por el nombre del pueblo, en nuestra tabla de datos y actualizar los valores necesarios. Para esto un municipio debería llamarse igual en los tres archivos xls que teníamos y al ser documentos oficiales este hecho se constata y hemos unificado los datos de los distintos xml en nuestra base de datos.

## 1.2. Extracción de url de los ayuntamientos

La extracción de las direcciones web de los ayuntamientos ha sido posible gracias a la API de búsqueda de Google, *Custom Search*, que nos permite realizar 100 búsquedas con el motor de búsqueda de Google pudiendo modificar éste con ciertos parámetros que la *Custom Search* nos permite cambiar como por ejemplo el idioma, la región o el número de respuestas que queríamos en una llamada (limitada a 10).

Creamos el buscador, y hacemos referencia al éste, con la id que nos da Google cuando lo creamos, en la url de la llamada a la API *Custom Search*. Hemos usado como palabras claves el nombre del pueblo ayuntamiento y la comunidad autónoma, para evitar confusiones de dos pueblos con el mismo nombre.

El inconveniente de este método es el límite diario de búsquedas que te tiene. La respuesta de esta API es en formato JSON. En los resultados de esta API, podemos obtener datos no válidos para nuestra experimentación. En nuestro caso nos hemos encontrado con resultados en las respuestas JSON de páginas que no eran las del ayuntamiento (En ocasiones normales porque éste no tenía) como por ejemplo *ayuntamiento.es* <http://www.ayuntamiento.es/>, *páginas amarillas* [www.paginasamarillas.es/](http://www.paginasamarillas.es/), y otras páginas webs la cual nos daban información sobre los pueblos en lugar de la página del ayuntamiento como la página del ayuntamiento de badajoz. Y en ocasiones, si la llamada a la API nos daba 10 resultados, un 80 % de los enlaces devueltos por la búsqueda con nuestro motor, eran de este tipo.

Para la diferenciación de los pueblos que no tienen página web del ayuntamiento con los que no han sido explorados todavía, ponemos el campo url de los pueblos en la tabla de datos a null por defecto, y los pueblos que exploramos y no tienen página web del ayuntamiento lo marcamos como analizado.

Es importante mencionar que aunque este proyecto se centra en los pueblos andaluces, Y en concreto con OpenCMS, se han configurado varios buscadores, para abarcar también municipios que tienen otra lengua además del castellano ya que en dichos municipios la búsqueda tiene un mayor porcentaje de éxito si tienen la palabra **Ayuntamiento** en dicha lengua.

Tenemos más de 8000 municipios españoles, y tenemos las url de los ayuntamientos de un 50 % aproximadamente. La ejecución de este script debe ser un proceso supervisado y cuando decidimos acortar el número de pueblos con los que íbamos a trabajar, decidimos que únicamente íbamos a trabajar con municipios andaluces, optamos por dejar de actualizar las urls de los municipios y dejar el resto para trabajos futuros.

Hay páginas webs de algunos ayuntamientos los cuales o han cambiado de dirección, o la página principal se encuentra en una url del mismo dominio pero no en el index, y éstas páginas con la etiqueta *http-equiv=x-refresh* redireccionan a la auténtica página principal del ayuntamiento.

La base teórica de este script se centra en la necesidad de estar en la página principal para navegar a través de ésta para realizar la recopilación de datos automáticamente. Por eso aunque al usuario le damos la url tal y como la recuperamos del script sin redireccionar, ya que el usuario accederá a través de un navegador, para nuestro uso a la hora de realizar el scrap a los datos de los ayuntamientos, este script es necesario.

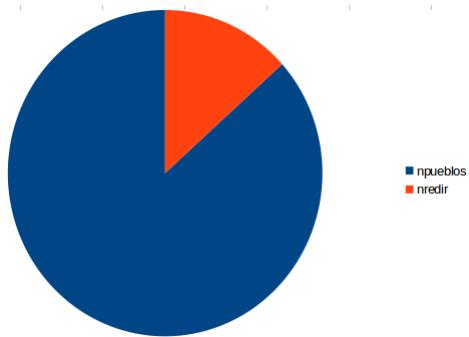


Figura 1.1: Porcentaje de pueblos con redirección



Figura 1.2: Pueblos con noticias en Semandal

### 1.3. Obtención de coordenadas

Para la obtención de coordenadas hemos usado la API de Geolocalización de Google (*Geocode*). Para la construcción de la búsqueda en la API, usamos la estructura de **pueblo + provincia + España** (Para evitar confusión).

La respuesta de la API es en JSON, así que accediendo a las posiciones adecuadas de los arrays devueltos, podemos obtener la latitud y la longitud de nuestros municipios.

Hemos construido un fichero **plantilla\_vacia.html** con el contenido necesario para mostrar un mapa de Google maps en blanco, sin ningún tipo de marcador. Este fichero necesita una API key, única para cada usuario, que nos permite visualizar el mapa. Una vez creado este fichero, obtenemos la latitud y longitud de todos nuestros municipios en un script que se encargará de, a nuestro mapa en blanco, agregar los datos adecuados para representar los municipios que elegimos, como por ejemplo, mostrar los municipios de los que tenemos noticias almacenadas en la base de datos.



Figura 1.3: Pueblos de España



Figura 1.4: Prueba del correcto posicionamiento de los pueblos

A continuación mostramos todos los pueblos que tenemos (Ver figura 1.3) y una prueba de que el pueblo que nos muestra está correctamente posicionado (Ver figura 1.4)

## 1.4. Búsqueda de municipios cercanos

Una vez obtenidas todas las coordenadas de los distintos municipios decidimos determinar para cada municipio los cinco municipios más cercanos. Para hacer el cálculo se determina la distancia de cada municipio con el resto, luego se ordena la lista generada de menor a mayor para tomar los cinco primeros, que son los cinco municipios más cercanos.

## 1.5. Páginas con OpenCMS

Al acotar el radio de actuación del proyecto a los pueblos que tenían OpenCMS o no, debíamos saber qué pueblos implementaban esta opción. La dificultad que tiene este tipo de pueblos, es que algunos sí tienen en su redirección el `/opencms/opencms`

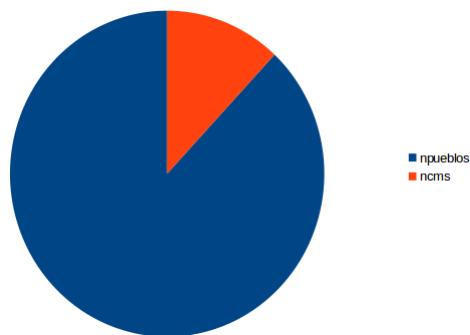


Figura 1.5: Porcentaje de pueblos con OpenCMS

pero otros aunque la página index.html sea la misma que se encuentra en /opencms/opencms/index.html no lo redireccionan, por tanto hemos hecho un script que nos dice qué municipios tienen su página web funcionando bajo la plantilla opencms. Así a la hora de recopilar datos, con una consulta, nos traemos los pueblos que tienen el campo opencms marcado.

## 1.6. Página de wikipedia

A la hora de obtener la página de wikipedia, sólo queremos la página que tenga una bandera o escudo, para mostrarla en la aplicación.

El problema a la hora de buscar pueblos es que en ocasiones nos podemos encontrar con una página de desambiguación, por este motivo debemos implementar dos scripts, uno que cuando no se encuentre la palabra desambiguación actualice la url de wikipedia del pueblo, y cuando se encuentre una página de desambiguación, use el mismo método que hemos usado para buscar las url de los pueblos, es decir la API *Custom Search* de Google, buscando esta vez la palabra wikipedia más el nombre del municipio más la provincia.

Existe un caso específico en el que este algoritmo falla y es cuando la búsqueda no nos da error o desambiguación, y la página que encontramos no es la del pueblo. De momento los casos que hemos encontrado son: LOJA, QUESADA, GALERA.

## 1.7. Extracción de datos del Instituto de estadística y cartografía de Andalucía

Otra de las fuentes utilizadas para la extracción de información es el Instituto de estadística y cartografía de Andalucía<sup>1</sup>. En este portal, dentro de la parte de estadística, se encuentran unas fichas municipales llamadas Andalucía pueblo a pueblo<sup>2</sup>. En estas fichas se encuentran publicados una serie de datos relativos a todos los municipios de Andalucía, estos datos incluyen información sobre el propio municipio, la población o a la economía de la zona. Aun siendo una información que necesite ser actualizada en algunos casos, puede ser de interés para el ciudadano.

El acceso a los datos se realiza con la librería `urllib2`, la URL tiene una parte fija y una variable. La parte fija es `...estadisticaycartografia/sima/htm/sm` y la parte variable es un código de provincia que ya extrajimos previamente desde el Instituto Nacional de Estadística (INE).

La estructura interna de la página es bastante sencilla y se estructura en tablas, esta estructura facilita el proceso de extracción de datos. Este proceso está basado, como en casos anteriores, en la librería BeautifulSoup de Python, con ella extraemos las celdas cuyos datos son de nuestro interés y a continuación se almacenan en la base de datos asignándolos al municipio correspondiente.

---

<sup>1</sup><http://www.juntadeandalucia.es/institutodeestadisticaycartografia>

<sup>2</sup><http://www.juntadeandalucia.es/institutodeestadisticaycartografia/sima/index.htm>

## 1.8. Visualizar la estructura de la web de un municipio

Cuando se crea un extractor de datos para la web se hace a medida de la web que se va a explorar, como queremos crear un sistema lo más genérico posible debemos buscar aquellas que tengan una estructuración tanto en código como en contenido similares. Para facilitar esta tarea las páginas que vamos a procesar son aquellas que hacen uso del gestor de contenidos OpenCMS, este gestor de contenidos es la base de un grupo de más de 100 municipios de Andalucía y además su uso está bastante extendido dentro de la Junta de Andalucía.

Como las empresas que crean las webs de estos municipios utilizan el mismo software como base para sus portales y los contenidos que se ofrecen en ellos serán similares, podemos suponer que la estructura interna de las páginas serán, de la misma manera, similares. Para ver esta estructura interna hemos creado un pequeño scrapper que busca y sigue los enlaces internos de la página para generar un grafo que representa la estructura conceptual de la web.

De aplicar este procedimiento a un pequeño conjunto de páginas, centrándonos en la búsqueda de noticias que es la información que deseamos almacenar, obtenemos los siguientes grupos:

### Grupo 1

Pueblos cuyas noticias se encuentran en las rutas `../actualidad/noticias/index.jsp` o `../actualidad/`, ordenadas por categorías. Su estructura conceptual se puede ver en la figura 1.6.

### Grupo 2

Misma ruta que el Grupo 1, `../actualidad/noticias/index.jsp`, pero las noticias no tienen asignada una categoría. Su estructura conceptual se puede ver en la figura 1.7.

### Grupo 3

Páginas estructuradas en XML, la ruta hacia las noticias es `../content/noticias_list_xmlpage.html?target=X`, donde X es una etiqueta. Las noticias se encuentran ordenadas según esas etiquetas. Su estructura conceptual se puede ver en la

figura 1.8.

## Grupo 4

Mismas rutas que en Grupo 1 pero con un código HTML distinto. Su estructura conceptual se puede ver en la figura 1.6.

## Grupo 5

Pueblos cuyas noticias se encuentran bajo las rutas `.../actualidad/noticias/` o `.../noticias/` sin ningún tipo de categorización. Su estructura conceptual se puede ver en la figura 1.9.

## Grupo 6

Pueblos cuyas noticias se encuentran bajo la ruta `.../portal/noticias.html?general=true` sin que las noticias tengan asignada una categoría.

## Grupo 7

Las noticias se muestran en la página principal sin tener asignada una categoría.

### 1.8.1. Resultados

Una vez podemos determinar a que grupo pertenece cada municipio, la figura 1.10 muestra la distribución de los datos:

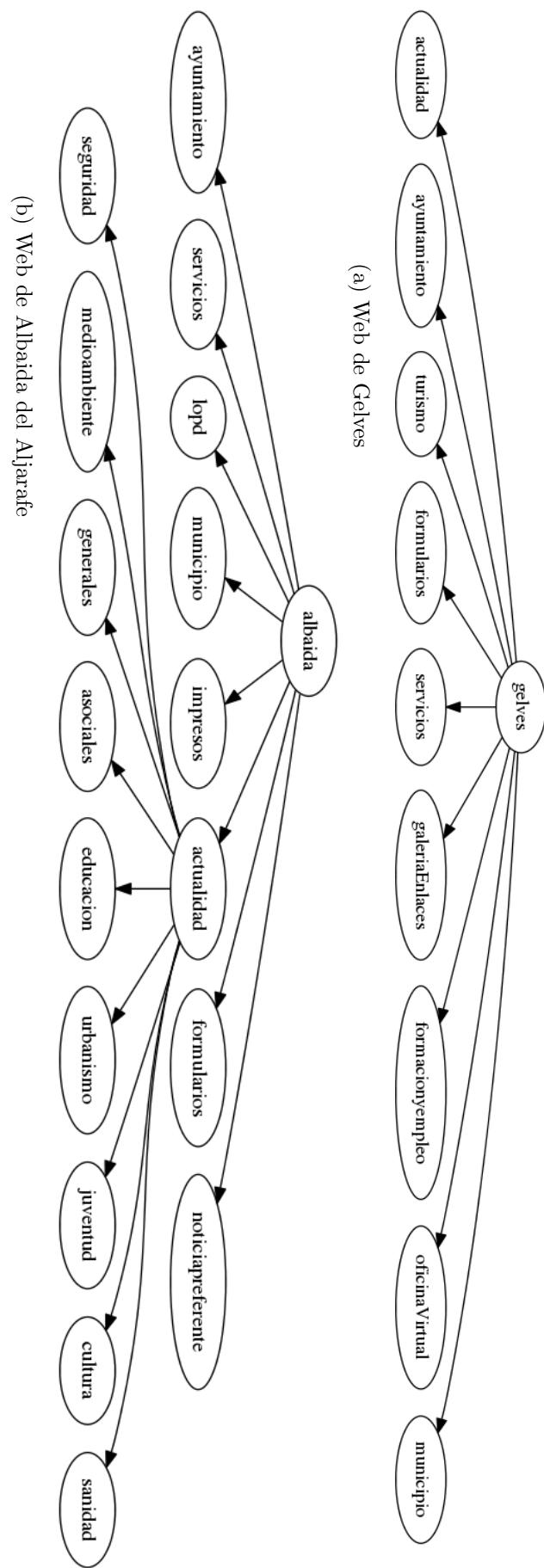


Figura 1.6: Grupos 1 y 4

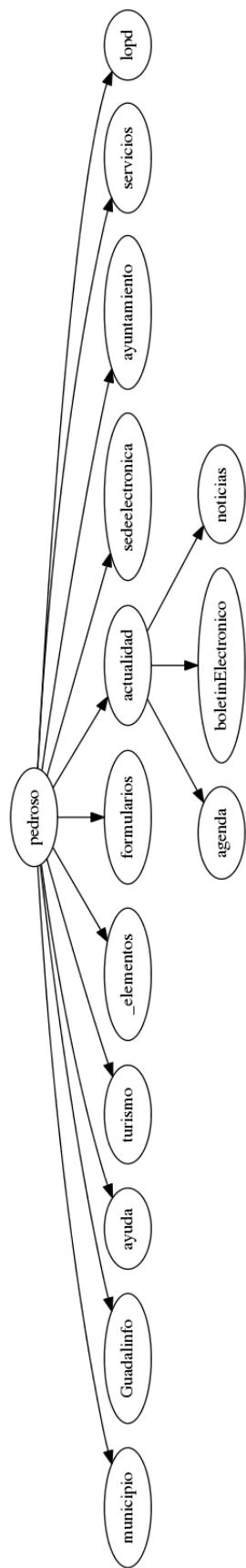


Figura 1.7: Web de El Pedroso, Grupo 2

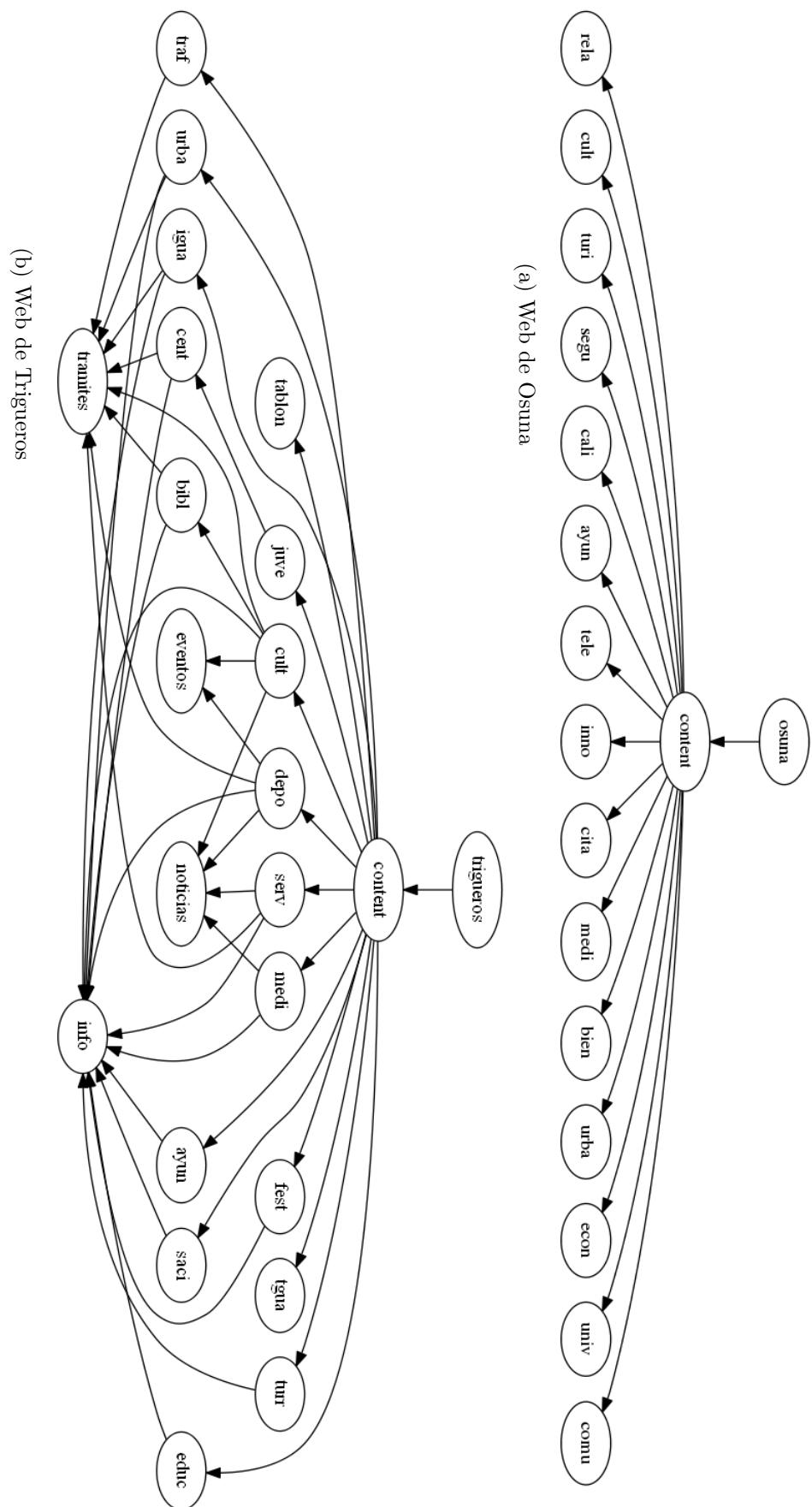


Figura 1.8: Grupo 3

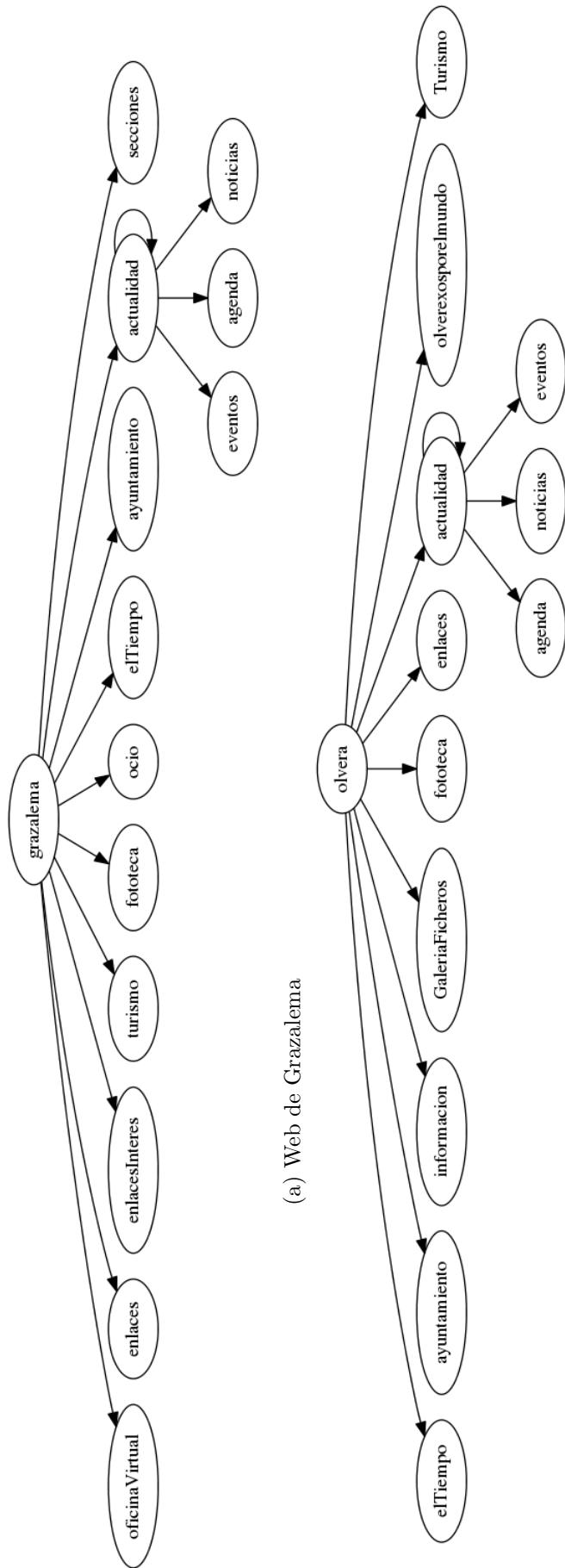


Figura 1.9: Grupo 5

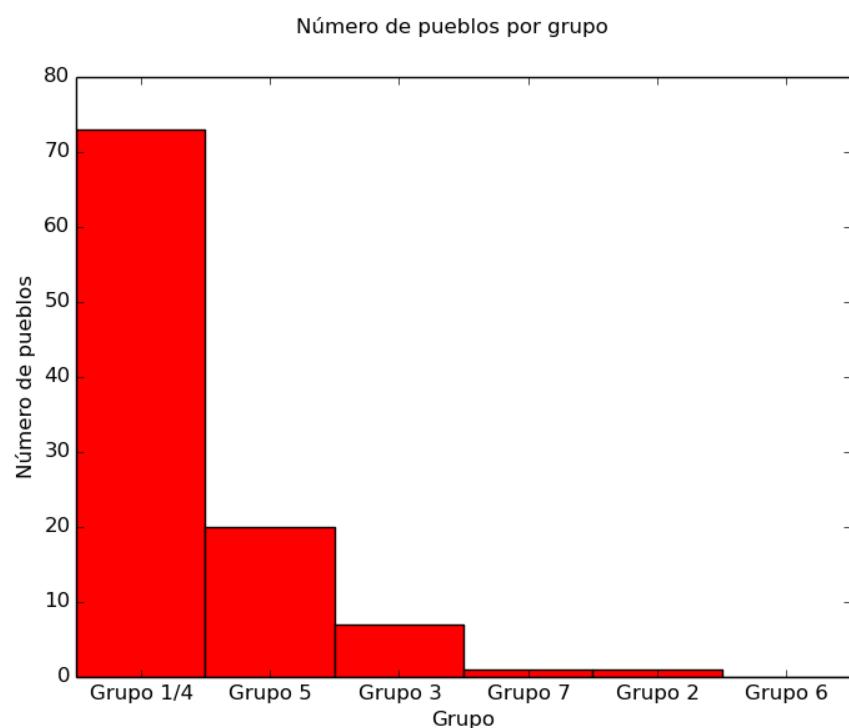
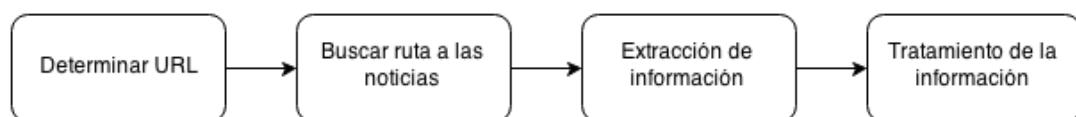


Figura 1.10: Municipios por grupos

## 1.9. Extracción de noticias desde la web de un municipio

La extracción de información es uno de los puntos clave del proyecto. Extraemos información de varias fuentes, siendo la más importante de ellas la web oficial del ayuntamiento de cada municipio. El caso que vamos a estudiar es el de aquellos ayuntamientos que hacen uso del gestor de contenidos OpenCMS, ya que se puede suponer que la estructura de las páginas serán similares al estar creadas con la misma herramienta.

Tras un estudio inicial para determinar la estructura interna de estas páginas hemos determinado siete formas distintas de estructurar los contenidos de la web. Como tenemos varios grupos, el primer paso es determinar a partir de la dirección web la ruta hacia las noticias, una vez queda determinada la ruta se procede a la extracción de las noticias. Para cada grupo se ha creado un script en Python para extraer las noticias según su estructura. Por último se trata la información obtenida para adecuarla nuestra base de datos.



### 1.9.1. Obtención de ruta hacia la información

Las páginas que hacen uso del gestor de contenido OpenCMS no almacenan la página principal en el directorio raíz del servidor, en la mayoría de casos esta se encuentra bajo la ruta `.../opencms/opencms/X` donde `X` es normalmente el nombre del municipio, por ejemplo, para Andújar la página del municipio es `http://andujar.es/` y la ruta que genera OpenCMS es `http://andujar.es/opencms/opencms/andujar/`. El determinar esta parte de la ruta a veces es un proceso simple pues al hacer una petición HTTP al servidor la URL de respuesta ya es la ruta completa por lo que hay una redirección automática y solo hay que seguirla.

Cuando no se da la redirección automática el servidor devuelve un pequeño código HTML en el que utiliza el atributo `HTTP-EQUIV` para forzar al navegador a hacer un refresco de la página con la dirección completa, en este caso hacemos, de nuevo, uso de expresiones regulares para obtener la parte extra que añade OpenCMS y construir la dirección completa que nos lleva a la página principal del ayuntamiento que estamos estudiando.

También puede ocurrir que no se dé la redirección automática y que no se devuelva el código anterior sino que directamente se muestra la página principal sin que se vea ningún cambio en la URL, en este caso lo que se hace es buscar todos los enlaces internos de la página principal haciendo uso de expresiones regulares, ya que incluyen la parte de OpenCMS que buscamos, y partimos los enlaces en tantas partes como caracteres "/" tenga el enlace y nos quedamos con las tres primeras partes para reconstruir parcialmente el enlace, estos enlaces parciales se guardan en una lista para que una vez tenemos todos enlaces poder buscar el elemento más común de la lista, que será la parte de la dirección que nos falta.

Una vez que tenemos la ruta completa, hacemos un repaso por los enlaces internos para determinar la estructura de la página y se compara con los siete grupos que obtuvimos en el análisis inicial intentando acceder a distintas rutas conocidas probando cuál de ellas es correcta y no devuelve ningún código de error al hacer la petición HTTP para obtener la ruta hasta la información que deseemos buscar, en este caso las noticias. Cuando se tiene la ruta a la información que se desea extraer se lanza el script correspondiente para comenzar a extraer información.

### 1.9.2. Extracción de noticias

Una vez tenemos la ruta hacia las noticias hay dos casos bien diferenciados, el primero es que las noticias se encuentren ordenadas por categorías, o por el contrario las noticias se encuentran todas en el mismo lugar sin ningún tipo de organización.

Si ya se encuentran ordenadas por categorías las almacenamos en nuestra base de datos respetando la categoría que se le da en la web.

Si en la web no se le asigna una categoría, las guardamos bajo una categoría especial para posteriormente, durante la fase de tratamiento de la información, asignarles una categoría de un conjunto creado por nosotros mismos.

Los atributos que almacenamos de cada noticia, además de una categoría, son: Fecha, Municipio, Titular, Resumen, Cuerpo y la URL a la noticia.

En ambos casos el proceso que se sigue para extraer las noticias es común, lo primero es acceder al código HTML donde se encuentra la información, para ello hacemos uso de la librería urllib2 para hacer la petición HTTP y guardar la respuesta, que si no se ha habido ningún problema será el código HTML. Una vez tenemos el código completo hacemos uso de la librería BeautifulSoup, uno de los parseadores de HTML para Python más utilizados actualmente.

```

1 cat = Categoria.objects.filter(etiqueta__exact = "sin_categoria")
2 try:
3     respuesta = urllib2.urlopen(url)
4 except:
5     return []
6 soup = BeautifulSoup(respuesta)
7 noticias = soup.body.findAll('div', attrs={'class' : 'result_group'})
8 for element in noticias:
9     fecha = None
10    titular = None
11    enlace = None
12    cuerpo = None
13    fecha = element.text
14    fecha = re.findall("\[(.*?)\]", fecha)
15    if len(fecha) != 0:
16        fecha = fecha[0]
17        fecha = fecha.split("/")
18        dia = date(day = int(fecha[0]), month = int(fecha[1]), year = int(fecha[2]))
19    else:
20        dia = None
21    titu = element.find('a')
22    if titu is not None:
23        titular = titu.text
24        titular = modifica_string.elimina_blanco(titular)
25        titular = modifica_string.elimina_char_especial(titular)
26        titular = titular.replace("\n", "-")
27        partes = url.split('/')
28        U = partes[0] + '/' + partes[2]
29        enlace = U + titu.get('href')
30        cuerpo = element.find('span')
31    .
32    .
33    .

```

Con esta librería seleccionamos y extraemos el contenido de las etiquetas HTML que contienen la información que buscamos, a veces sacamos dicha información haciendo uso de expresiones regulares y otras simplemente guardamos el texto que queda dentro de las etiquetas. Como obtenemos el enlace a la noticia completa también accedemos a esta dirección para obtener el texto completo de la noticia y en algunos casos la fecha si no ha sido posible obtenerla en el paso anterior. Una vez se ha extraído toda la información posible se construye el enlace a la siguiente página de noticias y seguir extrayendo información de manera recursiva. Como condición de parada de la recursividad tenemos la ocurrencia de algún error en la petición HTML, como un error 404, o que en el código se indique que ya no es existen más páginas para seguir trabajando.

Una vez se han extraído todas las noticias de un municipio, o de una categoría dentro de un municipio se almacenan en la base de datos ordenadas por orden de publicación, siendo la última la más recientemente publicada.

### 1.9.3. Almacenamiento de la información

A la hora de almacenar la información obtenida en concreto los resúmenes y los textos completos de las noticias nos hemos encontrado con determinados caracteres que no son admitidos por el gestor de base de datos. El origen de estos caracteres especiales está en que posiblemente OpenCMS ofrezca formularios para permitir al usuario publicar información y en ese formulario se copie el texto de otros programas que si admiten estos caracteres especiales como editores de texto, por lo que el primer paso es eliminar dichos caracteres, para ello repasamos todo el texto eliminando caracteres que queden fuera de la codificación UTF-8.

Otra situación que nos obliga a hacer tratamiento del texto obtenido son los fallos que contiene el código de las propias páginas web, ya que en muchos casos las etiquetas no están correctamente cerradas lo que hace que parte del código y comentarios se interpreten como texto por la librería BeautifulSoup. Estos fallos en ocasiones provocan una visualización incorrecta de la web.

En este apartado la tarea principal es realizar la clasificación automática de aquellas noticias que se encuentren sin clasificar. Para ello buscamos aquellas palabras que tienen relevancia en cada categoría para poder comparar el texto de cada noticia con estos conjuntos de palabras y determinar con la mayor certeza posible a qué categoría pertenece.

### 1.9.4. Sistema tolerante a fallos

Como dependemos de Internet para realizar esta tarea, es posible que se produzcan fallos en la conexión. Para poder recuperar la tarea de extracción sin tener de iterar de nuevo por todos los pueblos, se ha creado un sistema tolerante a fallos. Cada vez que se termina de buscar nuevas noticias en un municipio se almacena un estado que indica que el proceso ha terminado correctamente. En el caso de error, en la base de datos queda un estado que indica que el proceso no se ha completado.

Para recuperarse en caso de error se vuelve a lanzar el proceso de búsqueda de noticias que buscará posibles errores en la base de datos para realizar la búsqueda solo con aquellos pueblos que sean necesarios.

### 1.9.5. Resultados de la extracción

Tras realizar la fase de extracción de noticias y buscar noticias nuevas de forma posterior, contamos con 59177 noticias de 95 pueblos distintos, la distribución de los datos es la siguiente:

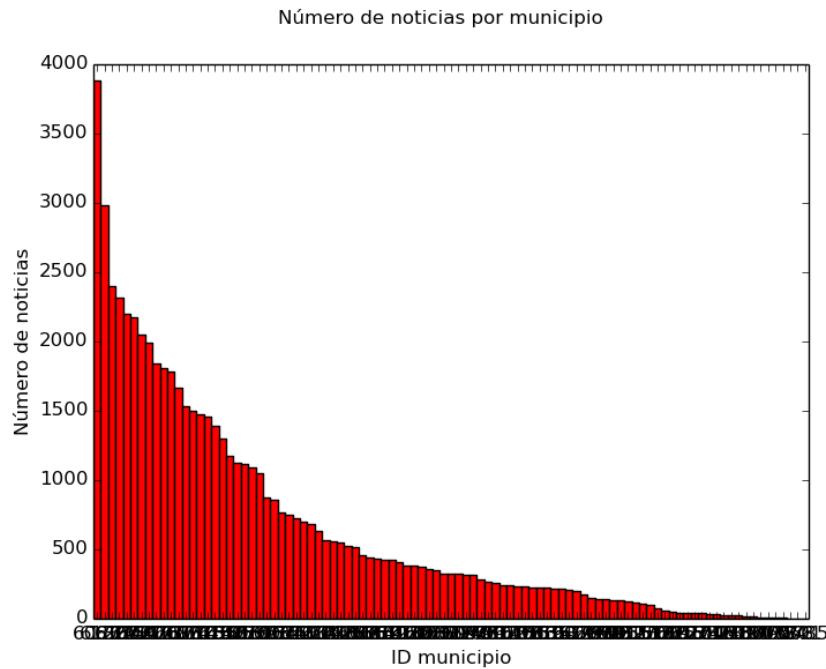


Figura 1.11: Número de noticias por municipio

Los pueblos con más noticias son:

- |                                |                 |
|--------------------------------|-----------------|
| 1. Castilblanco de los Arroyos | 4. Pedrera      |
| 2. Mairena del Alcor           | 5. Punta Umbría |
| 3. Benalup-Casas Viejas        | 6. Tocina       |

A continuación se muestra la figura 1.12 donde se muestra en número de noticias ordenado por provincias y la figura 1.13 muestra el número de noticias por año de publicación.

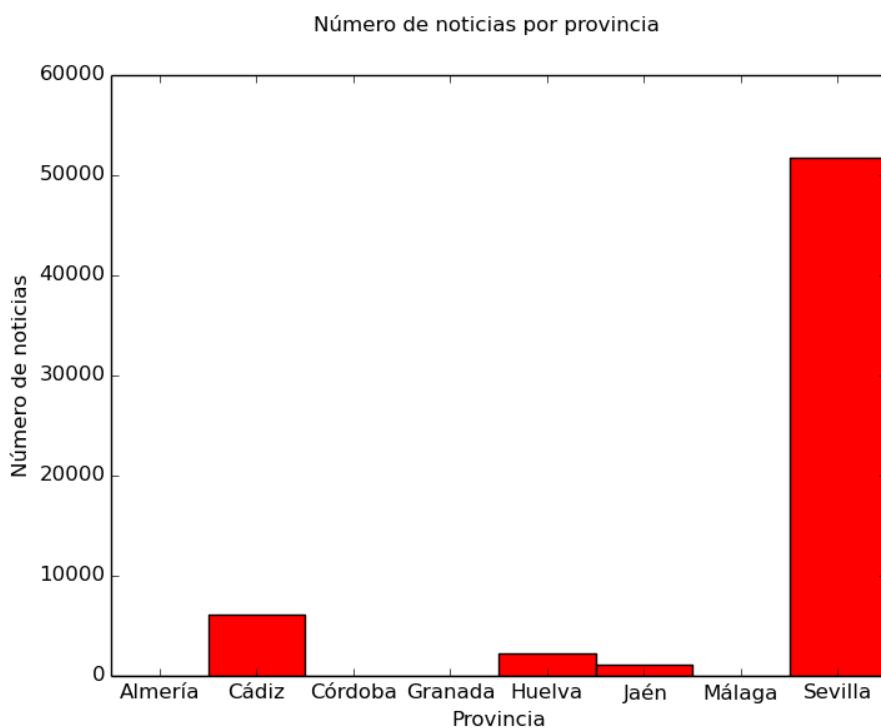


Figura 1.12: Número de noticias por provincia

Tras la fase de extracción, disponemos de la siguiente información:

<b>Nombre:</b>	Punta Umbría
<b>Provincia:</b>	Huelva
<b>Código:</b>	21060
<b>Latitud:</b>	37.1852585
<b>Longitud:</b>	-6.9704976
<b>URL:</b>	<a href="http://www/ayto-puntaumbria.es/">http://www/ayto-puntaumbria.es/</a>
<b>Deuda:</b>	1140 Euros
<b>Habitantes:</b>	14976 habitantes
<b>Superficie:</b>	38.77 Km <sup>2</sup>
<b>Densidad población:</b>	386.28 habitantes/Km <sup>2</sup>
<b>Página Wikipedia:</b>	<a href="http://es.wikipedia.org/wiki/Punta_Umbr%C3%ADa">http://es.wikipedia.org/wiki/Punta_Umbr%C3%ADa</a>
<b>Contratos indefinidos 2013:</b>	300
<b>Contratos temporales 2013:</b>	9489
<b>Noticia:</b>	Punta Umbría comienza mañana martes sus actos...
<b>Noticia:</b>	Punta Umbría rotula hoy y mañana vías con el nombre...

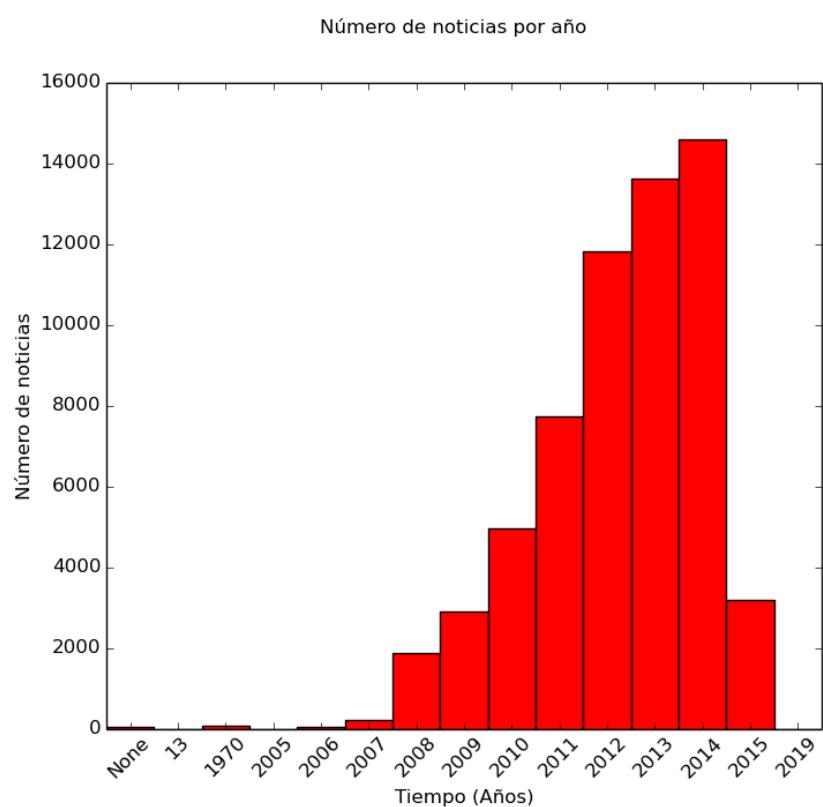


Figura 1.13: Número de noticias por año



# Búsqueda de categorías

---

Cuando estudiamos la arquitectura de las distintas páginas, nos fijamos en que muchas de ellas estructuran las noticias según una serie de categorías. Estas categorías son un buen punto de partida para comenzar a etiquetar la información que estamos extrayendo y no cuenta con alguna etiqueta, pero como las páginas están estructuradas de forma distinta nos encontramos que dada una categoría (Ej: *Deportes*), es referenciada con distintas etiquetas (Ej: *deportes*, *Deporte*, *depo*, *etc*) según el municipio que se visite, necesitamos un trabajo preliminar con el que determinar que conjunto de etiquetas hacen referencia a una misma categoría.

## 2.0.6. Búsqueda de sinónimos

Como hemos visto anteriormente, en la mayoría de casos un conjunto de palabras muy similares entre sí hacen referencia a una misma categoría, o nos encontramos con que se juntan etiquetas que hacen referencia a distintas categorías pero que en una o varias páginas se entiende como una única categoría (Ej: *saludyconsumo*). Las diferencias normalmente están en el uso de mayúsculas/minúsculas o singular/plural.

El primer paso del proceso es obtener una lista con todas las etiquetas únicas, para ello buscamos en la estructura de todas las páginas las etiquetas que usa cada una y aplicamos la distancia de Levenshtein.

### Distancia de Levenshtein

La distancia de Levenshtein es una forma de medir la distancia entre palabras, esta distancia está definida como el número de pasos que hay que dar para llegar de

una palabra A a una palabra B. Los posibles pasos son sustituir, insertar o eliminar una letra. Si aplicamos el algoritmo con las palabras *general* y *generales* obtenemos una distancia de dos, ya que tenemos que insertar dos letras para llegar de una palabra a otra.

Con este algoritmo recorremos la lista de categorías tomando una palabra y la comparamos con las siguientes, y si la distancia que indica el algoritmo es menor o igual a 3, consideramos que son sinónimos y que hacen referencia a la misma categoría. Las relaciones que vamos estableciendo las almacenamos en una base de datos.

Con el proceso anterior podemos hacer grupos con la mayoría de las etiquetas pero aún quedan palabras que no podemos asociar a ningún grupo ya que no cumplen el umbral de la distancia de Levenshtein establecido.

Una opción es el caso de encontrarnos con una categoría que en realidad podemos separar en dos etiquetas, para clasificar esta etiqueta buscamos otras cadenas que estén contenidas en la etiqueta compuesta. En el caso de *saludyconsumo* tenemos que salud y consumo forman parte de la etiqueta por lo que podemos decir que *saludyconsumo* responde a estas dos categorías, salud y consumo.

Otro caso son el de aquellas etiquetas que no se parecen, según el criterio establecido, a ninguna otra palabra de la lista, bien sea por que la etiqueta es un acrónimo, una abreviatura o una palabra totalmente distinta pero que representa el mismo concepto de una determinada categoría (Ej. La etiqueta *terceraedad* ha añadido a la categoría *Mayores*). Estos casos se han añadido a alguna categoría de manera manual.

### 2.0.7. Listado de categorías

Tras terminar los procesos anteriores la lista obtenida es la siguiente:

- |                       |                         |                           |
|-----------------------|-------------------------|---------------------------|
| 1. Turismo            | 8. Consumo              | 14. Municipal             |
| 2. Servicios sociales | 9. Grupos municipales   | 15. Deportes              |
| 3. Recursos Humanos   | 10. Seguridad Ciudadana | 16. Salud                 |
| 4. Obras              |                         | 17. Generales             |
| 5. Igualdad           | 11. Comercio            | 18. Atención al ciudadano |
| 6. Mujer              | 12. Juventud            |                           |
| 7. PIM                | 13. Medio Ambiente      | 19. Formación             |

---

20. Noticias anuales	32. Festejos	42. Social
21. Régimen Interior	33. Ayuntamiento	43. Tecnología
22. Desarrollo	34. Relaciones institucionales	44. Educación
23. Economía	35. Vivienda	45. Ciudadanía
24. Empleo	36. Urbanismo	46. Parques y Jardines
25. Guadalinfo	37. Cursos	47. Tráfico
26. Policía	38. Asuntos Sociales	48. Delegación
27. Bienestar social	39. Participación Ciudadana	49. Patrimonio
28. Agricultura	40. Infraestructuras	50. Mayores
29. Cultura	41. RSS	51. Radio y TV
30. Servicios		52. Contratante
31. Comunicación		

### 2.0.8. Refinado de las categorías seleccionadas

La lista de categorías se ha obtenido de forma automática en base a las etiquetas encontradas en la web, ahora podemos estudiar si existe algún tipo de jerarquía entre estas etiquetas y determinar si alguna categoría puede considerarse como subcategoría de otras ya que representan conceptos muy similares (Ej: *Policía y Seguridad Ciudadana*). Además podemos eliminar categorías que hemos encontrado pero que no representan ningún concepto como la etiqueta RSS o Noticias Anuales.

A continuación creamos una jerarquía con las categorías que tenemos y establecemos relaciones entre algunas de ellas ya que representan conceptos similares. La figura 2.1 muestra las categorías que utilizamos para crear nuestra jerarquía.

Una vez tengamos un sistema de extracción de noticias tendremos en cuenta esta jerarquía para etiquetar el contenido extraído.

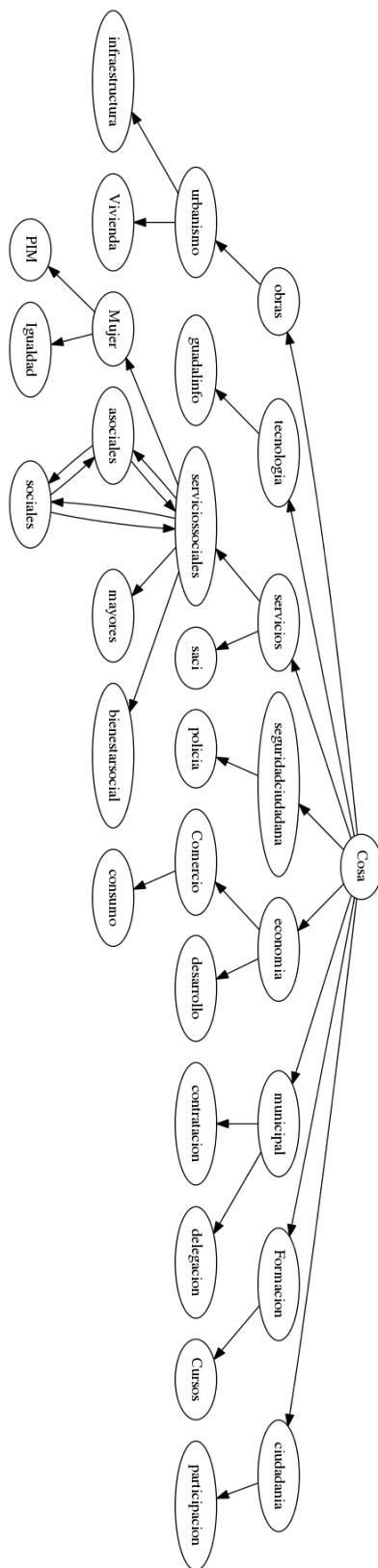


Figura 2.1: Jerarquía

## **Parte II**

### **De información a conocimiento**



# Métodos de clasificación

---

## 3.1. Selección de palabras clave de una categoría

Para determinar que palabras son las más representativas dentro de una categoría, buscamos aquellas palabras que se usan con más frecuencia dentro de cada categoría, una vez las tengamos las utilizaremos para clasificar aquellas noticias que no se encuentran recogidas bajo ninguna categoría. Para crear estos grupos de palabras utilizaremos las noticias que previamente hemos almacenado y conocemos su categoría, aproximadamente unos 50000 ejemplos de texto.

### 3.1.1. Método propuesto

Lo primero es crear una lista con todas las palabras que aparecen en las noticias de una determinada categoría eliminando determinadas palabras que no son importantes para la categoría a pesar de poder ser muy común encontrarla en los textos.

El criterio para eliminar estas palabras es el siguiente, tenemos una lista con una serie de palabras como pronombres, preposiciones, otras palabras de uso común, nombres propios y nombres de municipios. La razón de eliminar nombres propios y nombres de municipios es que encontramos referencias al pueblo, a la provincia o a personas que ostentan distintos cargos públicos en la mayoría de los textos por lo que acaban siendo palabras muy comunes que no aportan significado a la categoría donde se encuentran.

Tras este primer filtrado tenemos un segundo proceso en el que eliminamos caracteres numéricos, algunos caracteres de control y aquellas palabras que no superen los tres caracteres, la razón para esto es que las palabras con esa longitud que aca-

ban como palabras comunes y por ello representativas de la categoría son en realidad números romanos, que hacen referencia a algún tipo de evento, o abreviaturas como tlf, ext, o m.

Una vez tenemos la lista con las palabras candidatas generamos un diccionario a partir de dicha lista y así tener una estructura de datos donde la clave es la propia palabra y el valor es el número de apariciones de la palabra.

Una vez generados todos los diccionarios, uno por categoría, vamos iterando la lista de diccionarios. Para cada diccionario sacamos una lista con todas las claves y estudiamos como es de representativa cada palabra en todas las categorías. Para ello generaremos un vector en el que almacenamos como de común es la palabra que estamos estudiando en cada categoría. Pero como no tenemos un número de noticias igual en cada categoría necesitamos una medida relativa para poder comparar, por lo que dividimos el número de apariciones entre el número de noticias de esa categoría y obtenemos el número medio de apariciones de una palabra por noticia.

A continuación dividimos entre el máximo valor de la lista y multiplicamos por mil para normalizarla y poder tener una medida que como de común es la palabra dentro de la categoría con valores de cero a mil. Valores cercanos a cero indican palabras poco representativas y valores cercanos a mil indican palabras muy comunes. En este punto calculamos el valor medio del vector ya que un valor medio alto nos indica que la palabra es muy común en la mayoría de las categorías por lo que no aporta nada a ninguna categoría en particular, en este caso la palabra se elimina de todas las categorías donde aparezca. El siguiente paso es ver como es de común la palabra en el diccionario que se está estudiando comparando con el valor mayor valor, es decir la categoría donde la palabra es más común. Para ello utilizamos la diferencia relativa entre dos números con la siguiente formula.

$$\text{diferencia}(X, Y) = \frac{X - Y}{\text{MAX}(X, Y)} \times 1000 \quad (3.1)$$

Con esta fórmula obtenemos valores entre menos mil y mil, valores cercanos a menos mil indican que Y es muy distinto y mayor que X, valores cercanos a cero indican números muy parecidos y valores cercanos a mil indican que X es muy distinto y mayor que Y. Por lo que si comparamos siendo X el mayor valor del vector y Y el valor del vector que estamos estudiando, si obtenemos un valor cercano a mil estamos ante el caso de que la palabra en la categoría actual es muy poco representativa en comparación con el máximo por lo que eliminamos la palabra de la categoría actual.

### Experimentación y resultados

Si tomamos los datos anteriores y creamos un grafo conectando palabras y categorías, asignando un peso a cada arista igual al valor de frecuencia calculado anteriormente obtenemos el siguiente ejemplo:

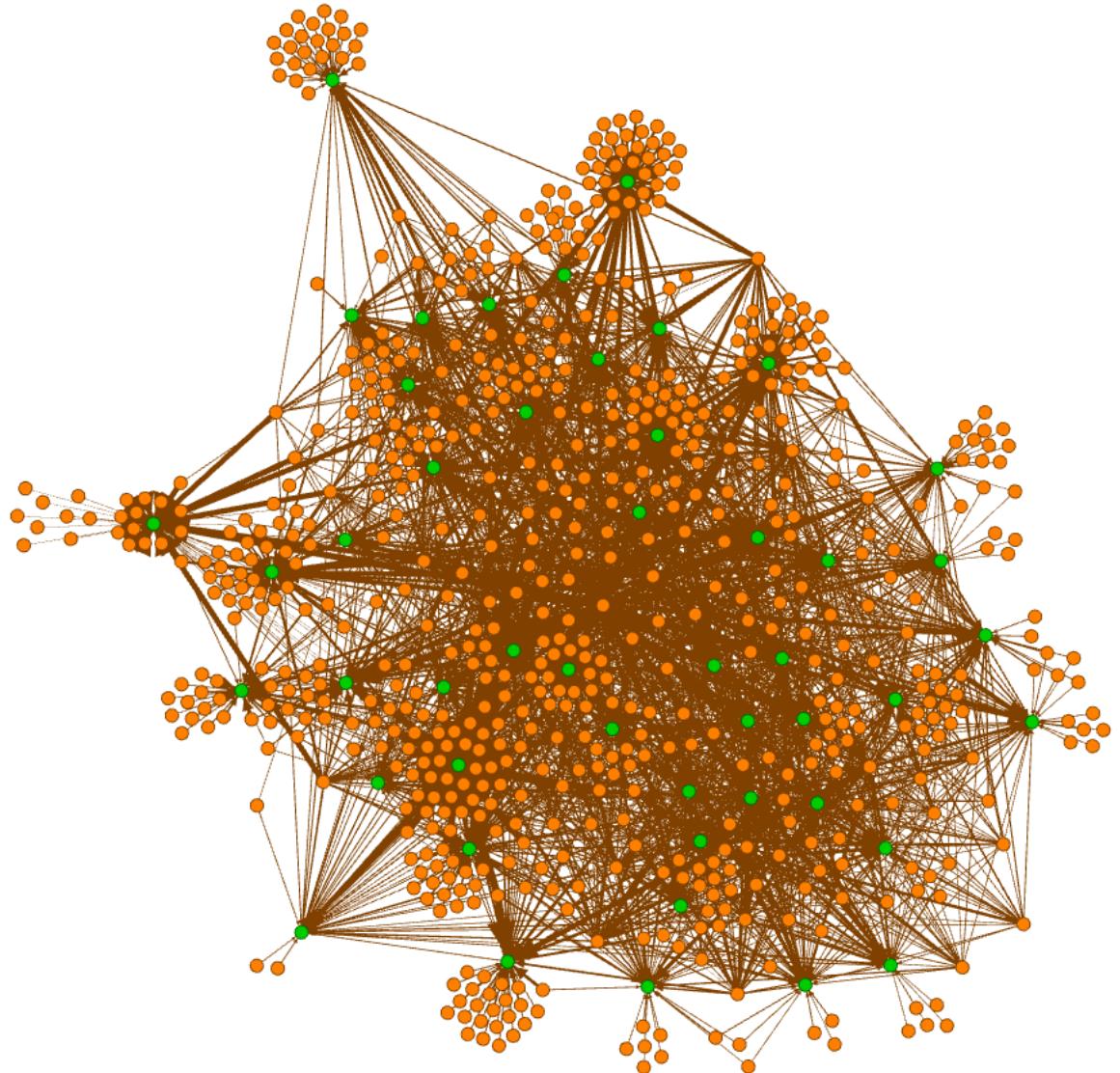
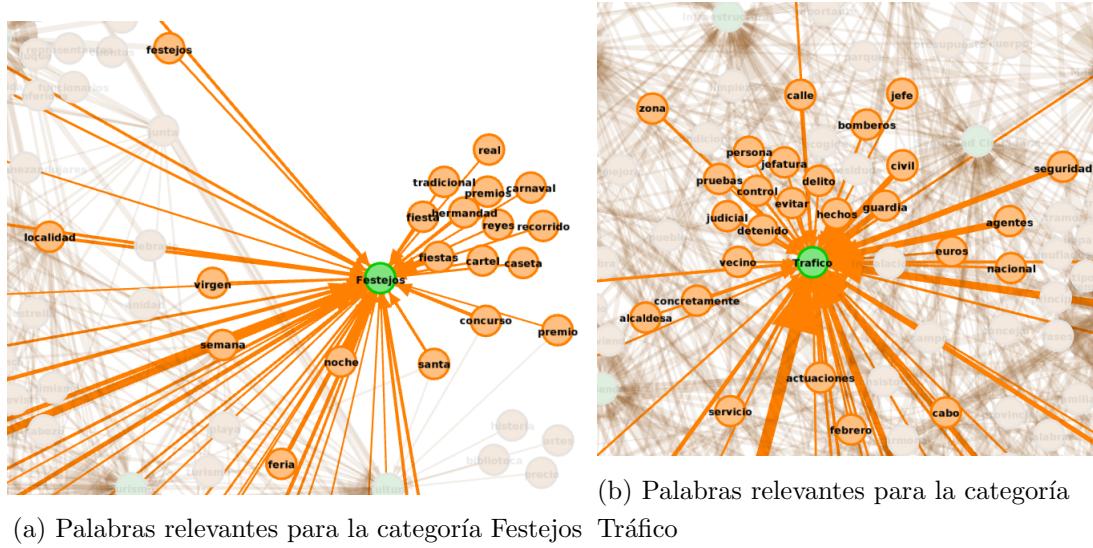


Figura 3.1: Grafo palabras y categorías

En este grafo vemos en color verde aquellos nodos que representan una categoría y en naranja las palabras. Cada categoría se rodea de aquellas palabras que son relevantes para ella.

A continuación se muestra el ejemplo de la categorías Festejos y Tráfico para mostrar como las palabras se organizan al rededor de sus categorías.



(a) Palabras relevantes para la categoría Festejos (b) Palabras relevantes para la categoría Tráfico

Categorías que representen conceptos similares y que por ello también comparten muchas palabras también quedan cerca. Como es el caso de las categorías Igualdad, PIM (Punto de información de la mujer) y Mujer.

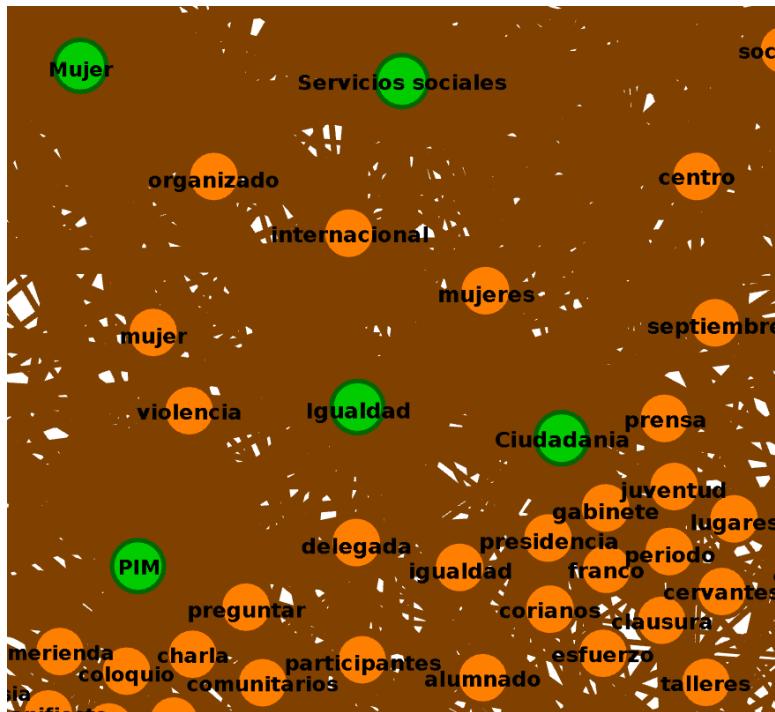


Figura 3.3: Grafo palabras y categorías

### 3.1.2. Aproximación TF-IDF

Otra forma de determinar que palabras son relevantes para una categoría es la aplicación del algoritmo TF-IDF. Este algoritmo se divide en dos partes, la primera es la frecuencia del término, esto es simplemente el número de veces que una palabra aparece en el documento. La segunda parte es la frecuencia inversa en el documento, con esto se intenta crear un factor de corrección para aquellas palabras que aparecen con mucha frecuencia en muchos documentos.

Desde un punto de vista matemático podemos definirlo como:

$$TF - IDF(w, d) = TF \times IDF(w, D) \quad (3.2)$$

La frecuencia de términos es el número de veces que el término  $w$  aparece en el documento  $d$ . La frecuencia inversa de documentos determina como de común es el término  $w$  dentro del conjunto de documentos  $D$ .

$$IDF(w, D) = \log\left(\frac{N}{\|\{d \in D : w \in d\}\|}\right) \quad (3.3)$$

## 3.2. Análisis formal de conceptos

### 3.2.1. Introducción

El análisis formal de conceptos, FCA por sus siglas en inglés, es una teoría matemática basada en conceptos y jerarquías de conceptos que tiene como objetivo el descubrir estructuras conceptuales dentro de un conjunto de datos. Esta teoría queda definida por Rudolf Willie en 1982 y actualmente su uso está bastante extendido especialmente en problemas lingüísticos, como el procesamiento del lenguaje natural y la creación de bases de datos lingüísticas.

### 3.2.2. Conceptos y contextos formales

Como se ha dicho anteriormente, el FCA se basa en conceptos y jerarquías, desde un punto de vista lingüístico, un concepto se compone de dos partes, una de ellas es conjunto de objetos que pertenecen al concepto (Extensión) y la otra es una serie de atributos o propiedades que tienen todos los objetos anteriores (Intensión). En FCA se crea un modelo matemático del concepto que permite hablar de objetos, atributos y las relaciones entre ellos que nos permiten expresar que un objeto tiene un atributo. Este modelo es el contexto formal. [10]

El contexto formal es la forma utilizada en FCA para la representación de los datos y se define como un conjunto de la forma  $K := (G, M, I)$  donde G, del alemán Gegenstande, es el conjunto de objetos. M, del alemán Merkmale, son los atributos e I son las relaciones binarias entre G y M (g tiene el atributo m). La representación más sencilla para un contexto formal es una matriz donde las filas se corresponden con el conjunto de objetos y las columnas se corresponden con el conjunto de atributos.

Un concepto formal del contexto formal  $K := (G, M, I)$  queda definido como una tupla  $(A, B)$ , donde A es un conjunto de objetos (Extensión) tal que  $A \subseteq G$  y B es un conjunto de atributos (Intensión) tal que  $B \subseteq M$ , además se debe cumplir que  $A = B'$  y  $B = A'$  donde  $A'$  y  $B'$  son operadores de derivación tal que:

$$A' := \{m \in M \mid gIm \text{ para todo } g \in A\} \quad (3.4)$$

$$B' := \{g \in M \mid gIm \text{ para todo } m \in B\} \quad (3.5)$$

Como estos operadores forman una conexión de Galois, se cumplen las siguientes propiedades tanto para el conjunto A como para el conjunto B:

$$Z_1 \subseteq Z_2 \implies Z'_1 \supseteq Z'_2 \quad (3.6)$$

$$Z \subseteq Z'' \quad (3.7)$$

$$Z''' = Z' \quad (3.8)$$

Además los conceptos se pueden ordenar, se considera que un objeto es mayor que otro cuando:

$$(A_1, B_1) \leq (A_2, B_2) :\Leftrightarrow A_1 \subseteq A_2 \quad (3.9)$$

De esta forma decimos que  $(A_2, B_2)$  es un superconcepto de  $(A_1, B_1)$  y que  $(A_1, B_1)$  es un subconcepto de  $(A_2, B_2)$ .

### 3.2.3. Retícula

La retícula es una forma de representar conocimiento en FCA de forma ordenada. Esta representación se realiza con un diagrama etiquetado, o diagrama de Hasse, un ejemplo de como es una retícula se puede ver en la figura 3.4. Esta forma de representación no reduce los datos, por lo que tiene la ventaja de que mantiene todo el conocimiento descrito en el contexto formal y además permite visualizar la estructura conceptual que este conocimiento tiene [9].

En el diagrama, los nodos representan conceptos y las aristas representan relaciones entre conceptos, estas relaciones son del tipo A es subconcepto de B o A es superconcepto de B.

Si en un nodo aparece la etiqueta de un objeto  $g$ , ese nodo representa el concepto más pequeño que contiene a  $g$  en su extensión. Si aparece la etiqueta de un atributo  $m$ , el nodo representa el concepto más grande que tiene a  $m$  en su intensión. De esta forma, observando la retícula, podemos leer para un concepto  $(A, B)$  su extensión A y su intensión B.

### 3.2.4. Reglas de asociación

Las reglas de asociación son una proposición probabilística que se utilizan para describir hechos que se dan dentro de un conjunto de datos, en este caso dentro del contexto formal. Este tipo de reglas puede verse de la forma SI  $X$  ENTONCES  $Y$ , donde  $X$  es el antecedente o parte izquierda de la regla e  $Y$  es el consecuente o parte derecha de la regla[4]. Una regla puede tener uno o varios antecedentes o

consecuentes, para unir estos elementos se hace uso de los operadores de conjunción (AND) y disyunción (OR). Los parámetros que definen una regla son los siguientes:

Soporte, esta medida refleja cuantos objetos apoyan el antecedente o el consecuente de la regla. Confianza, la confianza refleja en tanto por ciento cuantos objetos que apoyan en antecedente cumplen el consecuente.

### 3.2.5. Concept Explorer

Concept Explorer es la herramienta que hemos utilizado cargar el contexto que vamos a generar y obtener las reglas de asociación que posteriormente utilizaremos en la fase de clasificación. Además podremos visualizar la jerarquía de nuestro contexto.

#### Creación del contexto formal

Para la creación de un contexto  $K := (G, M, I)$  necesitamos definir un conjunto de atributos ( $M$ ) y un conjunto de objetos ( $G$ ). Como atributos hemos tomado un conjunto de palabras extraídas de las noticias formado por las palabras más comunes de cada categoría de noticias. Como conjunto de objetos tenemos el texto de un número determinado de noticias de cada categoría.

El proceso que hemos seguido para especificar las relaciones entre objetos y atributos ( $I$ ) es el siguiente, lo primero es generar el conjunto de atributos como está descrito en la sección Selección de palabras clave de una categoría. Para obtener el conjunto de objetos hacemos consultas a la base de datos para obtener, de cada categoría, un subconjunto de noticias seleccionadas de manera aleatoria. Una vez hemos determinado ambos conjuntos, tomamos el texto y la lista de atributos y recorremos la lista comprobando si el elemento está en el texto si está lo señalamos en el fichero con una X y sino con un punto.

El contexto creado se guarda con un fichero de extensión ".ext" generado de manera automática. La estructura de un fichero de contexto es la siguiente:

```

1 B
2
3 46
4 90
5
6 64110
7 112609
8 68691
9 .
10 .
11 .
12 .

```

El contexto queda representado en Concept Explorer de la siguiente manera:

Figura 3.4: Representación del contexto obtenido

## Creación del retículo

Otra de las posibilidades que ofrece FCA es la representación del contexto y de las relaciones entre objetos y atributos, estas relaciones son conocidas como relaciones de Galois. Como durante la realización del proyecto hemos generado contexto con un número alto de atributos y objetos la representación del retículo no se puede visualizar

de forma clara. Aún así es posible obtener una representación simplificada reduciendo el número de objetos, el retículo obtenido se muestra en la figura 3.5.

### 3.2.6. Experimentación

Para determinar como afecta el tamaño de los conjuntos de objetos y atributos a la precisión del clasificador hemos realizado pruebas con diferentes tamaños de conjuntos con los valores indicados en la siguiente tabla:

Experimento	Número de objetos	Número de atributos
Experimento 1	133	145
Experimento 2	258	208
Experimento 3	372	257

Con la herramienta Concept Explorer generamos reglas de asociación en base Stem. Para generar el fichero .clp que necesitamos para la fase de clasificación utilizamos un código Java escrito por Gonzalo Aranda para leer las reglas obtenidas en Concept Explorer y generar el fichero de salida. Este código ha sido modificado para tener en cuenta una serie de restricciones del problema. La primera es eliminar aquellas reglas con un soporte igual a cero, ya que no tienen ningún objeto que las cumpla. También eliminamos aquellas reglas que en su parte izquierda no tengan una etiqueta de alguna categoría ya que no nos aportan información útil en nuestro estudio.

Durante la experimentación se obtuvieron los siguientes números de reglas:

Experimentos	Número de reglas obtenidas	Número de reglas válidas
Experimento 1	7412	2650
Experimento 2	26344	9813
Experimento 3	66910	53898

La regla <3>talleres desarrollo proyecto =[100 %]=><3>actividades; no es una regla válida por no tener una etiqueta en el consecuente y la regla <2>delegado sanidad =[100 %]=><2>salud objetivo ETIQUETA\_consumo ETIQUETA\_salud se considera correcta.

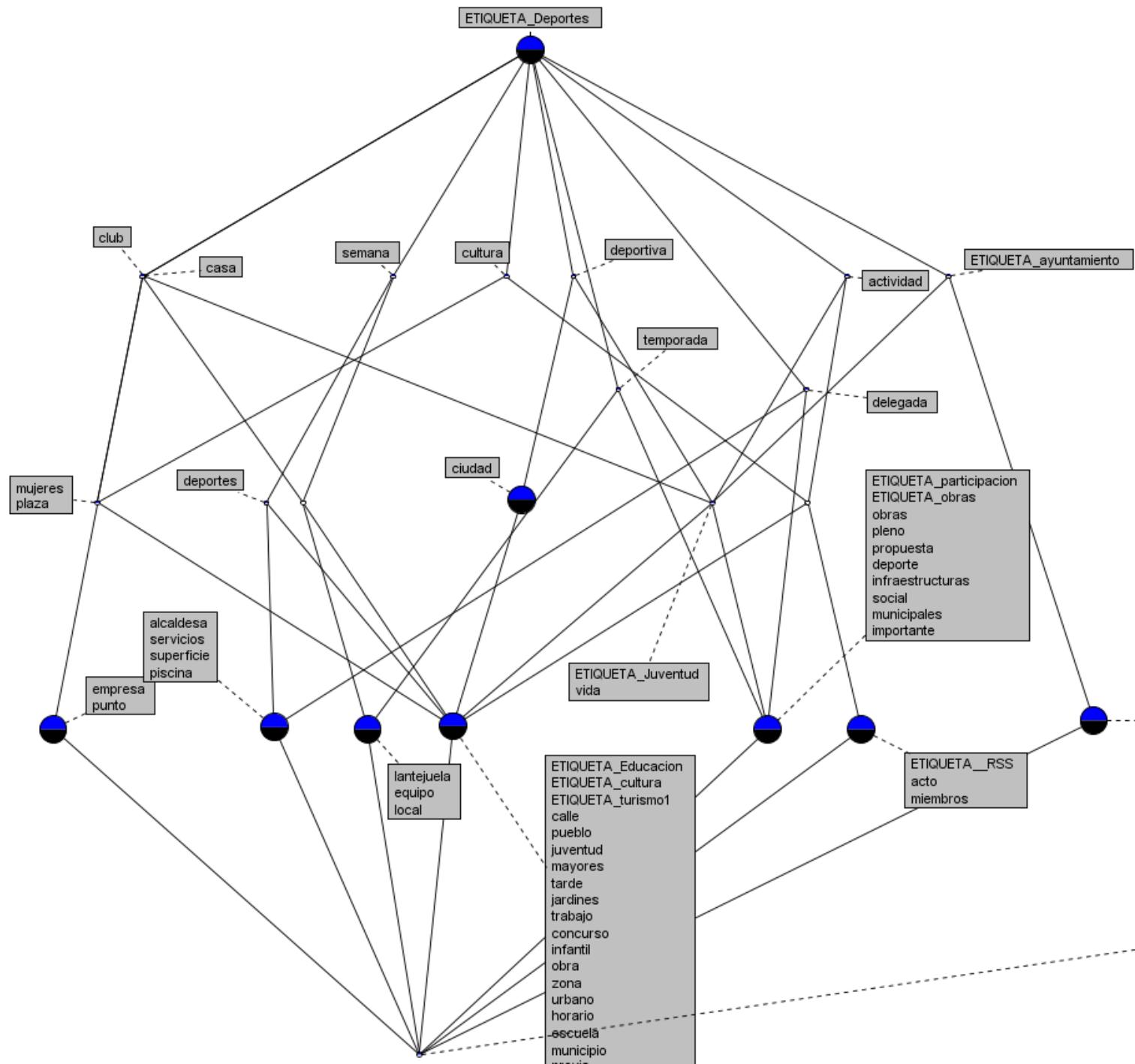


Figura 3.5: Representación del contexto obtenido

### 3.3. Árboles de decisión

Los árboles de decisión son modelos de predicción utilizados en inteligencia artificial que toma como entrada un conjunto de atributos y devuelve un valor que representa la decisión tomada por el árbol, este valor puede ser discreto o continuo. Los nodos internos del árbol contienen un test sobre una o varios atributos de entrada, según el resultado del test así será el camino que se sigue hasta llegar a una hoja, en las hojas se encuentran las posibles decisiones que el árbol puede tomar[8].

Como los árboles de decisión son muy aplicados en problemas de clasificación, vamos a utilizar esta técnica para la clasificación de aquellas noticias de las que desconocemos su categoría. Para la experimentación hemos utilizado la herramienta WEKA y la implementación del algoritmo C4.5 que incluye.

#### 3.3.1. Algoritmo C4.5

El algoritmo C4.5 es un algoritmo propuesto por Quinlan basado en un algoritmo anterior llamado ID3. El C4.5 genera árboles de decisión que tiene como objetivo determinar la clase ( $C_1, C_2, \dots, C_x$ ) a la que pertenece un objeto a partir de la evaluación de un conjunto de atributos, su uso está bastante extendido en las áreas de aprendizaje automático y minería de datos[7].

El árbol de decisión se genera a partir de un set de entrenamiento, este set contiene un conjunto de ejemplos ya clasificados  $S = (s_1, s_2, \dots, s_n)$  y cada ejemplo  $s_n$ , está formado por un conjunto de atributos  $(d_1, d_2, \dots, d_n)$  y una clase  $C_x$ . El árbol se compone de nodos de decisión y hojas, en los nodos de decisión se toma un atributo y se realiza un test sobre él, para el caso de atributos continuos este test determina un valor de umbral  $t$  para tomar una decisión en función de dos posibilidades, que son  $d_n \leq t$  o  $d_n > t$ . Las hojas del árbol se corresponden con alguna clase.

El árbol generado se aplica en problemas de clasificación, basándonos en los valores de los atributos vamos construyendo un camino desde la raíz del árbol hasta alguna hoja, la clase que contiene la hoja a la que se llega será la clase que sugiera el algoritmo como clase de un ejemplo sin clasificar.

### Determinar la calidad del árbol

Para determinar la bondad del clasificador obtenido se mide la cantidad de ejemplos que se han clasificado de forma correcta. Esta valoración se suele dar en tanto por ciento y se calcula a partir de dos parámetros que son el número de ejemplos correctamente clasificados (True Positive Rate) y los clasificados incorrectamente (False Positive Rate).

Con los datos anteriores también se puede crear un matriz de confusión, esta matriz cuadrada de tamaño  $n$ , donde  $n$  es el número de clases y contiene el número de elementos clasificados de forma correcta e incorrecta. Las columnas contienen las clases que predice el algoritmo y las filas las propias clases de forma que la diagonal de la matriz contiene los ejemplos correctamente clasificados y el resto de celdas contiene errores. La ventaja de usar esta matriz es que permite visualizar de forma rápida si una o varias clases se están clasificando de forma errónea. [6]

#### 3.3.2. Creación del conjunto de datos

Un conjunto de datos de weka tiene la estructura que se puede ver en el siguiente bloque de texto, los elementos principales son los atributos (`@attribute`) y los datos (`@data`).

Para la realización de experimentos de clasificación con Weka necesitamos definir lo que son atributos y datos, para la selección de atributos tenemos un conjunto de valores numéricos, cada valor representa una palabra tomada del conjunto de palabras seleccionado con el procedimiento descrito en la sección Selección de palabras clave de una categoría y un atributo extra que representa las distintas clases que tenemos, estas clases es nuestro conjunto de categorías.

Para generar la parte de datos se han tomado dos aproximaciones. La primera es crear un conjunto de datos donde el valor de cada atributo es la frecuencia media con la que aparece la palabra dentro de la categoría a la que pertenece la noticia, la segunda es una aproximación booleana donde solo se tiene en cuenta si la palabra aparece o no en la noticia.

A continuación se muestra un fragmento de uno de los conjuntos de datos que hemos creado:

```
1 @relation Semandal
2
3 @attribute @@class@@ { Agricultura , Comercio , cultura , Cursos , Deportes , economia , Educacion ... }
4 @attribute sector numeric
5 @attribute cobre numeric
```

```

6| @attribute cultura numeric
7| @attribute sociales numeric
8| @attribute agricultura numeric
9| @attribute empleo numeric
10| .
11| .
12| .
13| .
14| @attribute seguridad numeric
15| @attribute actividades numeric
16|
17| @data
18| turismol , 0, 0, 0, 0, 0, 0, ... 0, 0
19| ssociales , 0, 0, 0, 0.545905707196, 0, ... 0, 0
20| recursosHumanos , 0, 0, 0, 0, 0, 0, ... 0, 0
21| obras , 0, 0, 0, 0, 0, 0, ... 0, 0, 0
22| igualdad , 0, 0, 0, 0, 0, 0, ... 0, 0
23| Mujer , 0, 0, 0, 0, 0, 0, ... 0, 0
24| PIM , 0, 0, 0, 0, 0, 0, ... 0, 0
25| consumo , 0, 0, 0, 0, 0, 0, ... 0, 0

```

### Conjunto de datos con frecuencia de apariciones

Para crear la sección de atributos, primero hemos calculado el conjunto de palabras característico de cada categoría obteniendo una lista de palabras junto con su relevancia dentro de la categoría, siguiendo el procedimiento descrito en Selección de palabras clave de una categoría. A continuación se han concatenado las listas de todas las categorías eliminando repetidos, esta nueva lista de atributos incluye palabras como *agricultura*, *cobre*, *cultura*, *empleo*, etc, son las palabras que se pueden ver en el conjunto de datos anterior en la parte de *@attribute*.

Luego hemos tomado un número de noticias por categoría, cada texto se corresponderá con un registro del conjunto de datos (*@data*), a continuación se muestra un posible ejemplo:

„El Ayuntamiento de Punta Umbría, con la colaboración de los centros de Primaria de la localidad, ha organizado un certamen de dibujo para escolares bajo el título ‘El agua es vida’. El concejal delegado de Medio Ambiente del Consistorio costero, Andrés Franco Ramos, repartirá mañana miércoles, 25 de marzo, los premios a los mejores trabajos de cada colegio puntaumbriense. Así, a las 13.00 horas está previsto que acuda al CEIP Virgen del Carmen.

Con este concurso el Consistorio costero conmemoró el Día Mundial del Agua, que se celebró el pasado 22 de marzo “

Para crear una línea del conjunto de datos en la sección *@data*, lo primero buscar la categoría de la noticia, esta categoría se corresponde con uno de los posibles valores del atributo *@@class@@*. A continuación debemos apuntar *n* elementos, uno por cada

palabra de la lista de atributos. Para determinar el valor que hay que anotar debemos ver si cada palabra de la lista de atributos aparece en la noticia. Si no se encuentra la palabra en la noticia, en el conjunto de datos escribimos el valor cero. Si la palabra aparece en la noticia, buscamos el valor de frecuencia de apariciones de la palabra dentro de la categoría de la noticia y anotamos el valor correspondiente. Si la palabra no es relevante dentro de la categoría de la noticia, aunque se encuentre en la lista de atributos y en la noticia, escribimos un cero. El resultado de este proceso es cada una de las líneas que se puede ver en el conjunto de datos anterior en la sección `@data`.

### Conjunto de datos booleano

Una segunda aproximación es crear un conjunto de datos donde solo habrá dos posibles valores para los atributos, 1 si la palabra aparece en la noticia o 0 si la palabra no está en la noticia. El método de construcción de este conjunto de datos es el mismo que hemos utilizado en el caso anterior, pero anotando uno de los dos valores anteriores en vez de la frecuencia con la que aparece la palabra.

#### 3.3.3. Experimentación

##### Pruebas realizadas

Para las pruebas hemos realizado un conjunto de datos para la herramienta WEKA, para ver la relación entre el tamaño del conjunto de datos y la precisión del clasificador resultante hemos probado distintos tamaños de conjunto de datos que se detallan a continuación:

Experimentos	Número de palabras	Número de noticias
Experimento 1	1064	2842
Experimento 2	1649	4478
Experimento 3	2124	5253

Como algoritmo hemos utilizado PART en WEKA con un cross-validation de 10 folds para el experimento uno y 15 folds para los experimentos dos y tres.

### 3.3.4. Resultados obtenidos

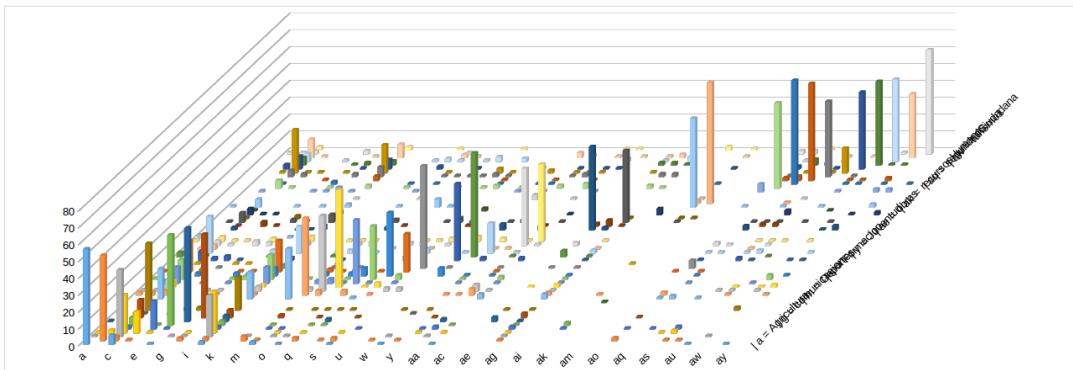
#### Frecuencia de términos

Los resultados obtenidos muestran una mejora de la precisión del clasificador obtenido conforme se aumenta el tamaño del conjunto de datos, aunque son sucesivos aumentos del conjunto de datos la mejora es cada vez menor. Las tasas de aciertos obtenidas son las siguientes:

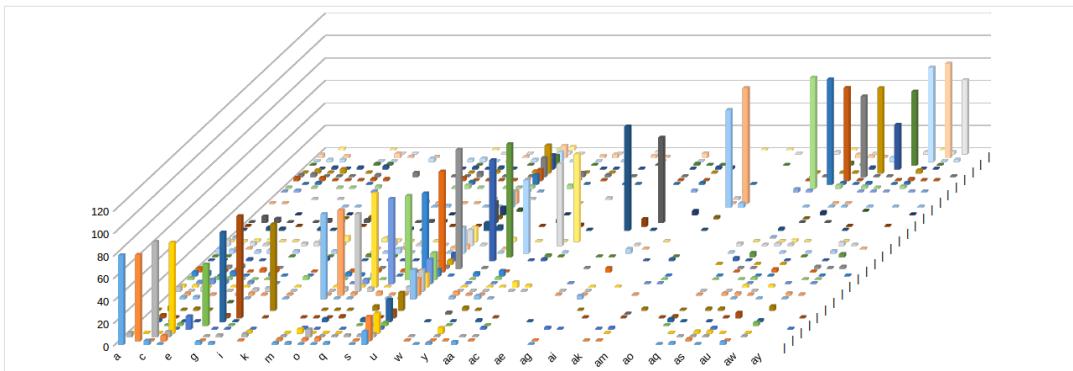
-	Experimento 1	Experimento 2	Experimento 3
Tasa de acierto	56.4391 %	64.9397 %	64.6868 %

Sería posible seguir aumentando el tamaño del conjunto de datos pero cada vez la mejora entre experimentos es menor. Para dar una representación visual de la calidad del clasificador utilizamos matrices de confusión. Las matrices obtenidas son las siguientes:

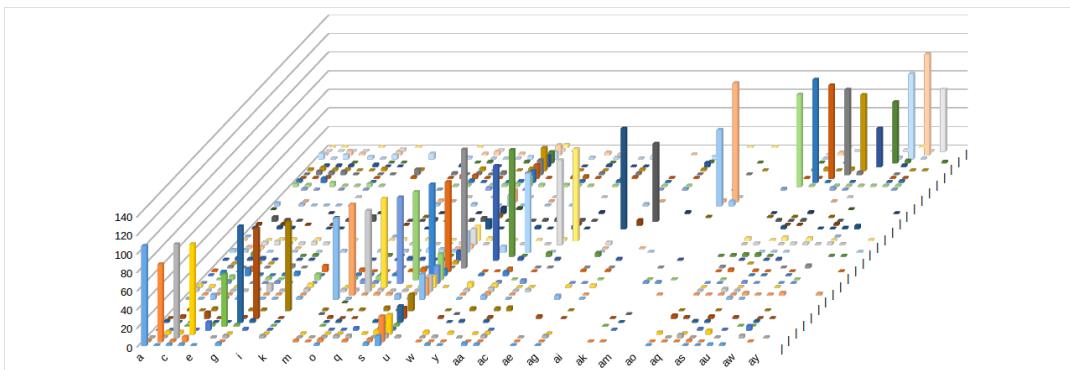
Experimento 1



Experimento 2



Experimento 3



Como se puede observar en el primer experimento el clasificador es capaz de etiquetar correctamente la mayoría de los casos. Solo muestra clasificaciones erróneas con las noticias de las categorías *Ayuntamiento* y *Cultura*, posiblemente por compartir un gran número de palabras clave con otras categorías. En los siguientes experimentos, al ampliar el conjunto de palabras que determinan cada categoría y el número de ejemplos que se dan al algoritmo, se mejora la clasificación de estas categorías, pero empeora la clasificación de la categoría *Generales*. Esta categoría es una fuente de ruido pues muchos municipios la utilizan sobre noticias que podrían considerarse de otras categorías y hace que la lista de palabras que definen la categoría *Generales* tome palabras de otras categorías que realmente no aportan información dentro de la categoría *Generales*.

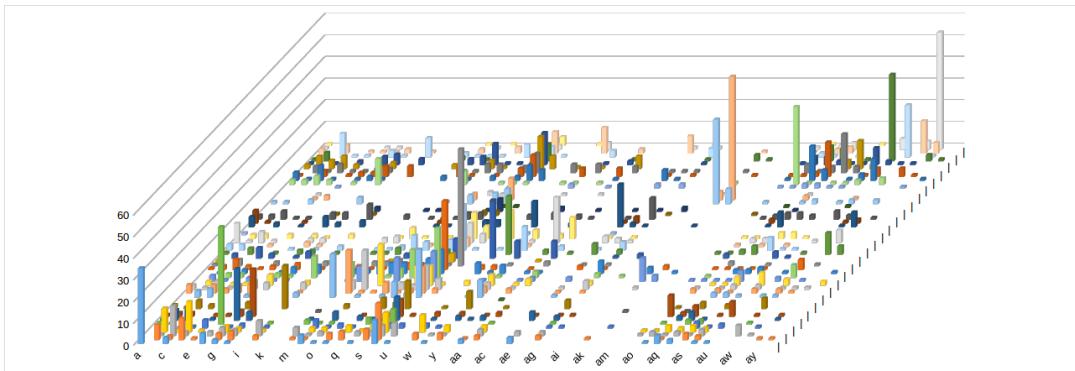
### Conjunto de datos Booleanos

Como en el caso anterior, la precisión aumenta si aumentamos el tamaño del conjunto de datos, pero como se muestra a continuación los resultados son bastante peores que los obtenidos utilizando frecuencia de términos.

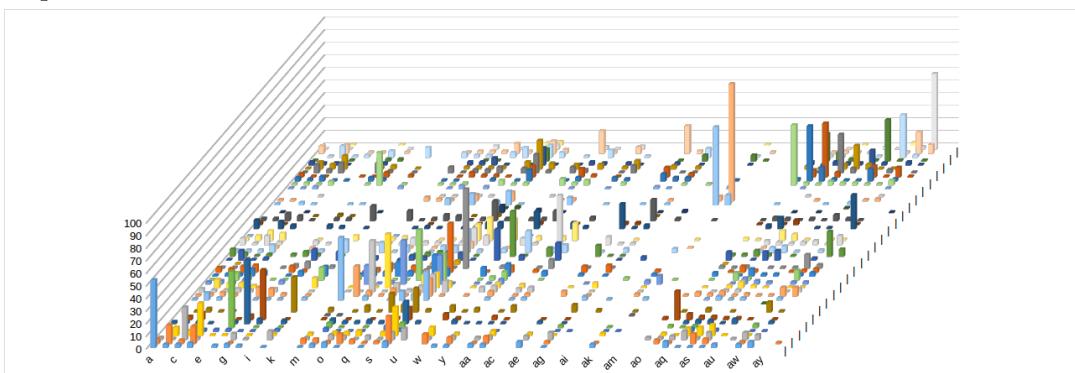
-	Experimento 1	Experimento 2	Experimento 3
Tasa de acierto	28.0185 %	29.1871 %	29.209 %

Las matrices de confusión obtenidas son las siguientes:

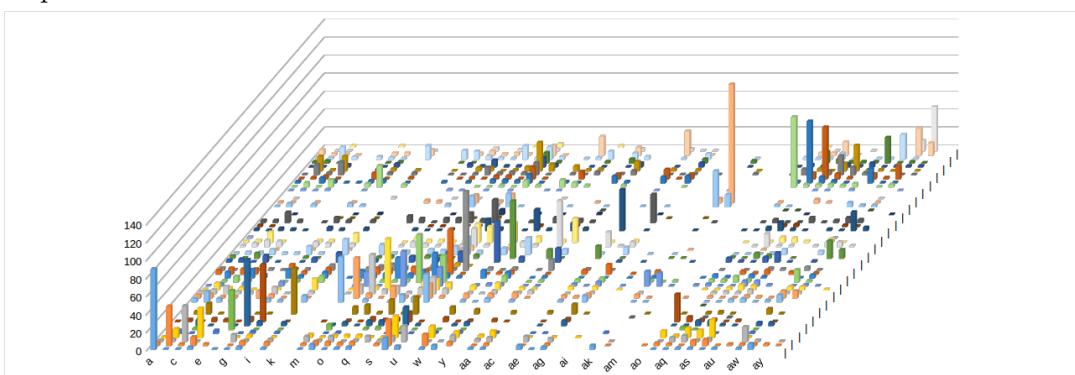
Experimento 1



Experimento 2



Experimento 3



A la vista de las matrices de confusión obtenidas podemos ver como esta aproximación obtiene unos resultados peores. La razón para esta diferencia en la precisión es que no se tiene en cuenta que el número de textos en cada categoría no es igual y que una palabra, aunque se utilice en varias categorías, no tiene la misma relevancia en todas ellas.

### 3.3.5. Extracción de reglas

Para extraer las reglas hemos utilizado la opción PART de WEKA que ejecuta el algoritmo C4.5 y se utiliza la estrategia divide y vencerás y técnicas de poda para crear árboles parciales de los cuales extraer reglas hasta cubrir todas las hojas.[3][5]

Una vez obtenidas las reglas en Weka, hemos tomado un código Java escrito por Gonzalo Aranda para cargar las reglas en memoria, modificando el código para leer las reglas en el formato que da Weka, y generar un fichero con las mismas reglas en un formato compatible con CLIPS. Este nuevo fichero se utilizará durante la fase de clasificación.

## 3.4. Sistemas de inducción de reglas

### 3.4.1. Algoritmo RIPPER

El algoritmo RIPPER (Repeated Incremental Pruning to Produce Error Reduction) es una versión mejorada del algoritmo IREP que busca minimizar el error, es decir se intenta reducir el número de ejemplos clasificados incorrectamente [1].

Está diseñado para generar reglas a partir de un conjunto de datos de forma incremental de la siguiente forma, para casos como el estudiado, con múltiples clases  $\{C_1, C_2, C_3 \dots C_k\}$ , lo primero es ordenar las clases de menor a mayor relevancia para intentar extraer reglas que separen la clase con menor relevancia ( $C_1$ ) del resto haciendo una llamada al algoritmo IREP siendo  $C_1$  la clase positiva y el resto la clase negativa [5].

Una vez se obtienen las reglas, todos las instancias que quedan cubiertas por las reglas obtenidas se eliminan del conjunto de datos y se repite el proceso siendo ahora la clase menos relevante  $C_2$ . Cuando solo queda la clase más relevante se termina el proceso y se determina que esta clase es la clase a asignar por defecto.

### 3.4.2. Creación del conjunto de datos

Para poder comparar los resultados obtenidos con los resultados de los árboles de decisión hemos utilizado los mismos conjunto de datos en ambos casos.

### 3.4.3. Experimentación

#### Pruebas realizadas

En esta caso se han utilizado los mismos conjunto de datos y los mismos parámetros de la herramienta Weka que se han utilizado con árboles de decisión con el objetivo de poder comprobar las diferencias en la precisión de ambos algoritmos. Dado que se comprobó anteriormente que la eficiencia de un conjunto de datos booleano era muy inferior a utilizar frecuencia de términos solo se ha utilizado esta última aproximación con este algoritmo.

Los conjuntos de datos empleados son los siguientes:

Experimentos	Número de palabras	Número de noticias
Experimento 1	1064	2842
Experimento 2	1649	4478
Experimento 3	2124	5253

Como algoritmo hemos utilizado JRIP en WEKA con un cross-validation de 10 folds para el experimento uno y 15 folds para los experimentos dos y tres.

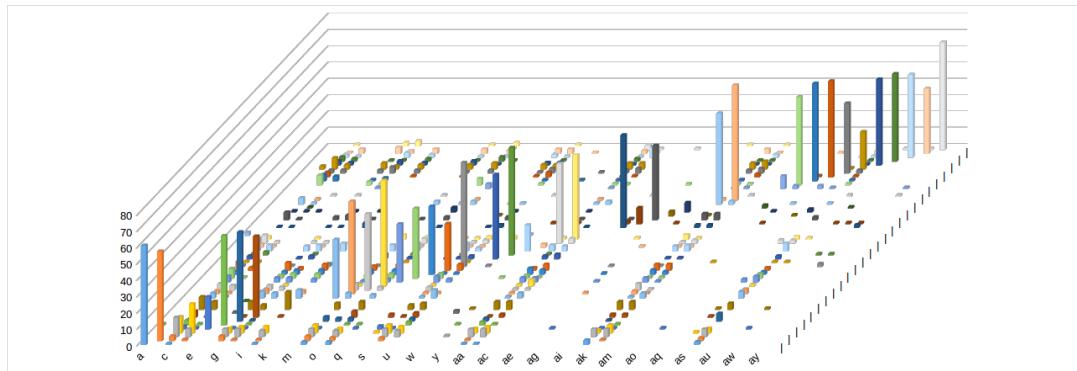
### Resultados obtenidos

Los resultados obtenidos muestran una mejora de la precisión del clasificador obtenido conforme se aumenta el tamaño del conjunto de datos, aunque son sucesivos aumentos del conjunto de datos la mejora es cada vez menor. Las tasas de aciertos obtenidas son las siguientes:

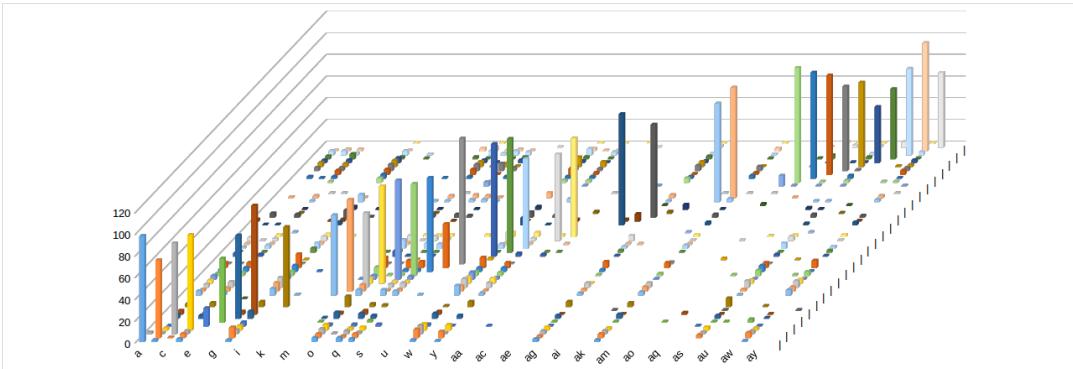
-	Experimento 1	Experimento 2	Experimento 3
Tasa de acierto	61.7171 %	69.1603 %	70.2456 %

Las matrices de confusión obtenidas son las siguientes:

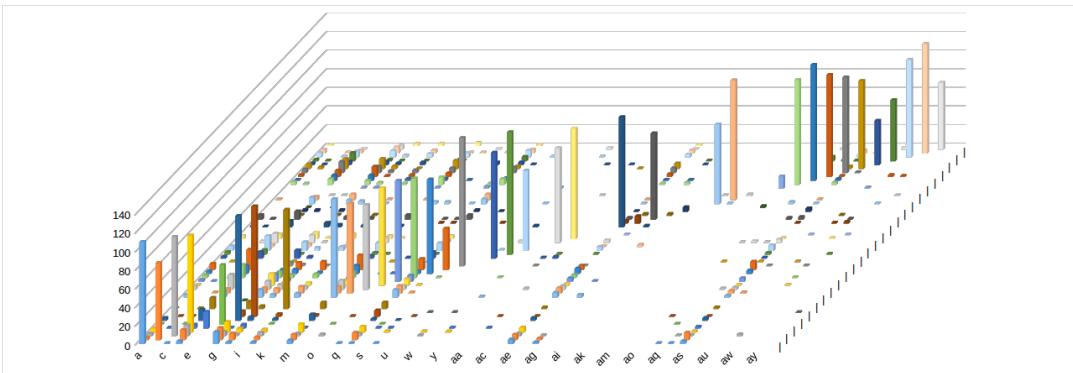
Experimento 1



Experimento 2



## Experimento 3



Comparando este algoritmo con los resultados obtenidos en 3.3.4, podemos observar que este algoritmo no presenta problemas con categorías concretas, no presenta problemas con las categorías *Ayuntamiento*, *Cultura* o *Generales*, pero en este caso el error en cada categoría es mayor, de media, el ratio de falsos positivos con RIPPER es mayor que el encontrado con árboles de decisión.

### 3.4.4. Extracción de reglas

Como en casos anteriores usamos un programa en Java para obtener las reglas en un fichero .clp adaptándolo para poder procesar como entrada las reglas en el formato que tiene Weka.

# Motor de clasificación

---

Para la clasificación de noticias hemos propuesto un sistema experto flexible que permita clasificar utilizando distintas metodologías, para ello como motor de nuestro clasificador la librería JESS y como reglas tenemos varios conjuntos de reglas en formato CLIPS que generamos a partir de árboles de decisión, análisis formal de conceptos y sistemas de producción de reglas.

## 4.1. Sistema experto

Como motor de clasificación hemos utilizado la herramienta JESS, que integra CLIPS con Java. Su funcionamiento se basa en el algoritmo RETE, este algoritmo se utiliza para ver las coincidencias entre un conjunto de patrones y un conjunto de objetos[2].

### 4.1.1. Algoritmo RETE

Los algoritmos de búsqueda de patrones se tratan de buscar coincidencias entre un conjunto de reglas y un conjunto de hechos en un proceso iterativo para buscar que reglas quedan satisfechas por el conjunto de hechos. Este proceso puede llegar a ser muy costoso desde el punto de vista computacional cuando el conjunto de reglas es grande.

El algoritmo RETE se diferencia de otros algoritmos de este tipo en que guarda en memoria la información obtenida en las iteraciones realizadas por lo que en siguientes iteraciones se ahorra el tener que repetir iteraciones ya hechas. De esta manera el

algoritmo es más rápido que algoritmos del mismo tipo utilizados con anterioridad.

El funcionamiento más detallado es el siguiente, la base es la creación de una red de nodos. Esta red almacena la información de la búsqueda de patrones entre iteraciones, de forma que solo es necesario procesar aquellos hechos que se han modificado, una vez hecho se actualiza la información guardada con los hechos añadidos o eliminados. Conforme la lista de hechos se reduce el proceso se hace más rápido.

## 4.2. Entradas del sistema experto

### 4.2.1. Conjunto de hechos

El conjunto de hechos que se le dará al clasificador serán las palabras de cada noticia que se quiera clasificar eliminando aquellas que no aportan información y están listadas en un fichero StopWords.

### 4.2.2. Conjunto de reglas

Para la generación de las reglas que utilizaremos junto a JESS hemos generado conjuntos de reglas con distintos métodos: análisis formal de conceptos, árboles de decisión y sistemas de producción de reglas. Utilizamos como fuente para generar estas reglas aquellas noticias de las que conocemos su categoría. En las siguientes páginas se documenta la metodología utilizada con mayor detalle.

## 4.3. Método propuesto

### 4.3.1. Clasificación de noticias

La fase de clasificación comienza con una selección en la base de datos de aquellas noticias que no están clasificadas, estas son aquellas noticias asignadas a la categoría *Sin Categoría*. Una vez hecho esto se van recorriendo una a una, y se toma el texto de la noticia para extraer una lista de palabras que usaremos como conjunto de hechos en el clasificador. El siguiente paso es lanzar desde Python un proceso que ejecute el archivo .jar que contiene el código de clasificador, pasando como parámetros los conjuntos de reglas y hechos. Una vez el clasificador termina se recupera la salida y se almacena el resultado en la base de datos. Si el clasificador no devuelve ningún

resultado, que se puede dar si la noticia no cuenta con un texto a parte del titular o resumen, se mantiene la categoría *Sin Categoría*.

#### 4.3.2. Aplicación de la jerarquía

Una vez se determina la categoría de una noticia, se comprueba si siguiendo la jerarquía que hemos creado podemos sugerir alguna etiqueta más. Para ello hacemos un recorrido del grafo que contiene la jerarquía (Ver figura 2.1) tomando como punto de partida la categoría que queremos asignar. Siguiendo el recorrido añadimos las etiquetas que corresponda. Si la categoría asignada no aparece en la jerarquía, o la noticia extraída no cuenta con una categoría en la web de su municipio, no se añaden etiquetas nuevas.

#### 4.3.3. Resultados obtenidos

Tras pasar todo este proceso, damos una o varias etiquetas a aquellas noticias que no tenían una categoría asignada en su web. A continuación se muestran unos ejemplos de noticias clasificadas:

##### Ejemplo 1

###### **Titular<sup>1</sup>**

La cabeza de la Liga se mantiene igual a la espera del Corinthians-Boomerang de este viernes

###### **Cuerpo de la noticia:**

Tras su derrota con el líder en la primera vuelta, para Boomerang este viernes 27 será la última oportunidad de poder acortar su distancia de siete puntos con Corinthians y poder seguir siendo alternativa al campeonato de liga regular. De lo contrario, será complicado que el líder pierda la cómoda distancia que posee de aquí a la llegada de los playoffs, plazas que ambos conjuntos tienen ya prácticamente aseguradas. Seguridad que aún no tienen el resto de aspirantes a esos primeros puestos que dan lugar a las eliminatorias por el título, ya que sólo existen ocho puntos de diferencia entre el tercero (Matadero) y el noveno (Casas Viejas Independiente), por lo que pa-

---

<sup>1</sup>[http://www.benalupcasasviejas.es/opencms/opencms/benalup/actualidad/noticias/noticia\\_2473.html](http://www.benalupcasasviejas.es/opencms/opencms/benalup/actualidad/noticias/noticia_2473.html)

san las jornadas y ninguno de los equipos de ese pelotón se descuelgan.-En segunda división, al jugarse la liga regular a una sola vuelta, Calle Nueva Veteranos depende de sí mismos para conquistar ese título honorífico, incluso podría perder un partido y seguir siendo campeón, ya que el enfrentamiento directo con Casas Viejas Veteranos, su inmediato perseguidor, lo tiene ganado.-Se adjuntan resultados de esta pasada semana de campeonato y clasificaciones.

**Etiquetas sugeridas:**

	RIPPER	FCA	C4.5
Algoritmo	Deportes	Grupos municipales	Asuntos sociales
	-	Juventud	-
	-	Deportes	-
	-	Desarrollo	-
	-	Cultura	-
	-	Ayuntamiento	-
	-	Educación	-
Jerarquía	-	Economía	Servicios sociales
	-	-	Servicios
	-	-	Social

**Ejemplo 2**

**Titular<sup>2</sup>**

Donación de Sangre el próximo 1 de diciembre en Bornos

**Cuerpo de la noticia:**

La Delegación de Sanidad del Ayuntamiento de Bornos informa que el próximo LUNES 1 de diciembre el Centro Regional de Transfusión Sanguínea de Cádiz, se va a desplazar a Bornos para realizar una campaña de recogida de sangre.- -Todo el que quiera podrá donar sangre a partir de las 17:30 horas y hasta las 21:30 horas, en el Centro de Mayores, Plaza I de Mayo.- -Recordar que con este simple gesto se pueden salvar vidas.- -Las condiciones para donar sangre son tener más de 18 años al menos 50 kilos de peso y gozar de buena salud.- -Los donantes habituales deben esperar dos meses entre una donación y otra y pueden realizar un máximo de cuatro

<sup>2</sup>[http://www.bornos.es/opencms/opencms/bornos/actualidad/noticias/noticia\\_1389.html](http://www.bornos.es/opencms/opencms/bornos/actualidad/noticias/noticia_1389.html)

donaciones por año los varones y tres las mujeres.- -DONACIÓN EN BORNOS- -LUNES 1 DE DICIEMBRE- -HORARIO: DESDE LAS 17:30 HORAS A LAS 21:30 HORAS- -LUGAR: CENTRO DE LOS MAYORES. PLAZA I DE MAYO

<b>Etiquetas sugeridas:</b>	RIPPER	FCA	C4.5
	Economía	Salud	Salud

### Ejemplo 3

#### **Titular**<sup>3</sup>

Concierto de 'Laura Gallego'

#### **Cuerpo de la noticia:**

Día: 10 de Septiembre de 2.011-Hora: A partir de las 23:00 h. Puntos de Venta de Entradas-Ayuntamiento:-Lunes a Viernes de 9:00 hº a 14:00 hº. Sábados de 10:00 hº a 14:00 hº. Excmo. Ayto. de Algar-Concejalía de Fiestas

<b>Etiquetas sugeridas:</b>	RIPPER	FCA	C4.5
	Cultura	Patrimonio	Generales

<sup>3</sup>[http://www.algar.es/opencms/opencms/algar/actualidad/noticias/noticia\\_0009.html](http://www.algar.es/opencms/opencms/algar/actualidad/noticias/noticia_0009.html)



# **Parte III**

# **Publicación**



# Django

---

Django es un framework de código abierto para el desarrollo web en python y sigue el patrón de diseño Modelo Vista Controlador (MVC).

Django es una herramienta que incita a la reutilización y la capacidad de interaccionar con otras herramientas, también al desarrollo rápido. Ofrece una interfaz administrativa de usuarios.

Django empezó al final de 2003 cuando los programadores web del **Lawrence Journal-World** empezaron a crear aplicaciones con Python. Fué lanzada en Julio de 2005 bajo una licencia **BSD** (licencia de software libre permisiva con un mínimo de restricciones) y en Junio de 2008 se anunció que la formación **Django Software Foundation (DSF)** se encargaría del mantenimiento de esta herramienta.

Django, drupal, ruby on rails, son frameworks parecidos pero queríamos algo sencillo y rápido y ya que el proyecto lo estábamos desarrollando en python, elegimos Django ya que hasta los archivos de configuración de ésta herramienta están escritos en python. Django facilita el desarrollo de páginas orientadas a contenidos y es muy sencilla la creación de la base de datos *MySQL* y mostrar esos datos en HTML. Django es también capaz de ejecutar un servidor web en nuestra máquina. La facilidad que nos aporta django para crear un servidor web se ve complementada además por varios plugins ofreciéndonos una 'puesta en escena' de nuestra aplicación más que aceptable en pocos pasos. Por todas estas razones, nos decidimos a usar django.

## 5.1. Instalación

Lo primero que debemos hacer es borrar cualquier instalación anterior de django si lo hemos instalado, si no podremos saltarnos este primer paso. Si hemos instalado django anteriormente en nuestra máquina, depende de la forma en la que lo hayamos instalado, pip o ejecutando el setup.py.

**instalado con pip o easy\_install** Si hemos instalado django con pip o easy\_install, instalando **Django** de nuevo con esta herramienta, ésta se hará cargo de como manejar las versiones antiguas.

**instalado con setup.py** Sin embargo, si hemos instalado **Django** a través del comando `python setup.py install` debemos borrar el directorio de django del **site-packages** de python. Para encontrar el directorio que debemos borrar, debemos ejecutar el siguiente comando.

```
1 python -c "import sys; sys.path = sys.path[1:]; import django; print(django.__path__)"
```

### 5.1.1. Ubuntu, MAC, Distribución Unix

Para instalar **Django** en estos S.S.O.O podemos hacerlo de dos maneras

pip

1. Instalamos la herramienta pip, la forma más sencilla es usar esta distribución independiente de pip <https://pip.pypa.io/en/latest/installing.html#install-pip>.
2. (**OPCIONAL**) Descargar las herramientas *virtualenv* y *virtualenvwrapper*. Estas herramientas proveen de entornos de Python que es más práctico que instalar los paquetes. Estas librerías también instalan paquetes sin necesitar los privilegios del administrador. Es opcional si se desea aprender a usarlo o no.
3. Usar el comando `pip install Django` Esto instalará **Django** en la carpeta **site-package** de *Python*. **si estamos en una distribución UNIX o MAC, debemos usar el comando sudo para instalarlo como superusuario**

Manualmente

1. Descargamos la última versión oficial de **Django**
2. Descomprimir el .tar.gz
3. Cambiar el directorio descomprimido por Django-X.Y Donde X.Y es el número de la versión.
4. Usamos el comando **python setup.py install** Esto instalará **Django** en la carpeta **site-package** de *Python*. si estamos en una distribución **UNIX o MAC**, debemos usar el comando **sudo** para instalarlo como superusuario

## 5.2. Creación del proyecto

Antes de empezar vamos a ver la diferencia que existen entre un proyecto y una API.

- Diferencia entre proyecto y API
  - Un proyecto tiene una o más APIs
  - API es una aplicación web que se encarga de hacer algo específico

Cuando instalamos la herramienta django en nuestra computadora, debemos iniciar el proyecto de django, con el siguiente comando.

```
1 $ python django-admin.py startproject mysite
```

Donde *mysite* será el nombre que tendrá nuestro proyecto.

Una vez que tengamos el proyecto creado debemos crear nuestra API con el siguiente comando.

```
1 $ python manage.py startapp polls
```

En este caso, *polls* es el nombre de la api que queremos crear.

Y una vez que ejecutamos este comando se nos crea una carpeta llamada *polls* (pueblos) en nuestro caso, con varios archivos con la extensión de *python*.

1. *admin.py*
2. *models.py*

	djangotest.pueblos_usuario
1	id : int(11)
2	dsnombre : varchar(100)
3	dsusuario : varchar(100)
4	pueblo_id : int(11)
5	token : varchar(200)
6	correo : varchar(100)
7	dsapellido1 : varchar(100)
8	dsapellido2 : varchar(100)

Figura 5.1: Tabla Usuarios de la base de datos Semandal

3. test.py

4. views.py

De estos archivos los más importantes son el `models.py` y el `views.py`. Explicaremos a continuación estos archivos y otros de los más importantes, `urls.py`.

### 5.3. Creación de la base de datos - /polls/models.py

Para la creación de la base de datos, debemos modificar el `models.py` que es donde escribiremos las tablas y sus correspondientes atributos. En lugar de la creación con sentencias sql, django nos permite escribir en código nuestra estructura de base de datos para posteriormente con la ejecución de un simple comando nos crea la estructura en nuestra base de datos mysql.

A continuación se adjunta un ejemplo de como crear una tabla en django.

```

1 class Usuario(models.Model):
2     dsusuario = models.CharField(max_length=100, unique=True)
3     dsnombre = models.CharField(max_length=100)
4     dsapellido1 = models.CharField(max_length=100)
5     dsapellido2 = models.CharField(max_length=100)
6     pueblo = models.ForeignKey(Pueblo)
7     token = models.CharField(max_length=200)
8     correo = models.CharField(max_length=100, null=True, unique=True)
9 
```

Esto crearía la tabla (Ver figura 5.1)

A continuación explicaremos los tipos de datos, que hemos usado en nuestro `models`, se pueden asignar con django y las opciones de éstas.

#### CharField

Éste método crea un atributo de tipo Texto

**BooleanField**

Crea un atributo de tipo boolean,

**PositiveIntegerField**

Crea un entero mayor que cero.

**DateTimeField**

Crea un atributo de tipo fecha.

**5.3.1. Manejo de la base de datos con django**

En esta sección realizaremos una enumeración de los comandos importantes en django, que hemos utilizado para nuestro proyecto.

**5.3.2. Creación de registro en una tabla**

Para crear un nuevo registro en la tabla, en primer lugar debemos crear el objeto. Si en el models, no hemos especificado que un campo puede ser null debemos agregarlo obligatoriamente, en caso contrario al intentar insertarlo django nos dará error.

```
1 sigP = SigP(id_user=u, id_p=p[0])
```

Cabe recordar que si la tabla tenía una clave foránea, como es el caso de nuestro ejemplo, *u* y *p[0]* deben ser objetos de tipo usuario y pueblo respectivamente por tanto debemos o crearlos antes o extraerlos de una consulta

**5.3.3. Consulta de un registro en una tabla**

Antes que nada, en las consultas de los registros explicamos que, cuando se iban a buscar objetos que tenían claves foraneas, debíamos pasar en la búsqueda el objeto o una referencia. En el siguiente ejemplo, en la primera linea pasamos objetos, y en la segunda, decimos que de la tabla SigP, el objeto usuario tiene un id = *u.id*.

```
1 sigue = SigP.objects.filter(usuario=u, pueblo = p)
2 sigue = SigP.objects.filter(usuario__id=u.id, pueblo__id = p.id)
```

De manera general, sería algo parecido a:

```
1 sigue = tabla1.objects.filter(atributo(tabla1)=Objeto)
2 sigue = tabla1.objects.filter(atributo(tabla1).__atributo(tabla2)=Atributo(tabla2))
```

En el proyecto usamos las dos, para demostrar que funciona de cualquier manera, pero la manera más correcta sería la segunda.

Vamos a explicar a continuación las dos maneras que existen de consultar registros de una tabla con django.

### **filter**

Filter se trae un QuerySet. Es decir, una lista de objetos de la tabla que consultemos. Si la consulta no devuelve ningún valor nos devolverá una lista vacía.

```
1 sigue = SigP.objects.filter(usuario=u, pueblo=p)
```

### **get**

Lo que nos devuelve get es un único objeto de tipo SigP (En este caso). De hecho, si intentamos realizar el *Where* “ y esa consulta nos da más de un resultado, django nos dará un error.

```
1 SigP.objects.get(id=x)
```

También tienen conversión los operandos clásicos de ayuda en las búsquedas

### **order\_by**

Éste quizas sea el operador que tiene la sintaxis más simple de todos los operadores auxiliares de búsqueda. Simplemente debemos escribir el **Order\_By()** y entre los paréntesis el campo por el que queremos ordenar.

```
1 rez = Comentarios.objects.filter(id_not=n_id).order_by('fecha')
```

### **reverse**

Este operador es el ASC o DSC del SQL clásico. Por defecto el order by los ordena ascendentemente y el reverser le da la vuelta al resultado de la consulta.

### **distinct**

Este es uno de los más complicados ya que tienen dos formas de invocarlo y dos resultados distintos.

**values** Nos devuelve un diccionario. A continuación pondremos una consulta y una imagen con el resultado que devuelve.

```
1 Noticias.objects.order_by().values('id').distinct()
```

**values\_list** Este método devuelve una lista de tuplas.

```
1 Noticias.objects.order_by().values_list('id').distinct()
```

```
>>> n = Noticias.objects.order_by().values('id').distinct()
>>> n
[{'id': 9297L}, {'id': 9298L}, {'id': 9299L}, {'id': 9300L}, {'id': 9301L}, {'id': 9302L}, {'id': 9303L}, {'id': 9304L}, {'id': 9305L}, {'id': 9306L}, {'id': 9307L}, {'id': 9308L}, {'id': 9309L}, {'id': 9310L}, {'id': 9311L}, {'id': 9312L}, {'id': 9313L}, {'id': 9314L}, {'id': 9315L}, {'id': 9316L}, '...(remaining elements truncated)...']
```

Figura 5.2: Diccionario devuelto

```
>>> n = Noticias.objects.order_by().values_list('id').distinct()
>>> n
[(9297L,), (9298L,), (9299L,), (9300L,), (9301L,), (9302L,), (9303L,), (9304L,), (9305L,), (9306L,), (9307L,), (9308L,), (9309L,), (9310L,), (9311L,), (9312L,), (9313L,), (9314L,), (9315L,), (9316L,), '...(remaining elements truncated)...']
```

Figura 5.3: Lista de Tuplas de valores devueltos

También podemos agregar en la consulta de values el parámetro flat a True, para que este nos devuelva una lista.

```
1 Noticias.objects.order_by().values_list('id', flat=True).distinct()
```

```
>>> n = Noticias.objects.order_by().values_list('id', flat=True).distinct()
>>> n
[9297L, 9298L, 9299L, 9300L, 9301L, 9302L, 9303L, 9304L, 9305L, 9306L, 9307L, 9308L, 9309L, 9310L, 9311L, 9312L, 9313L, 9314L, 9315L, 9316L, '...(remaining elements truncated)...']
```

Figura 5.4: Lista de valores devueltos

### 5.3.4. Actualizar registro en django.

El método update de django se realiza sobre un QuerySet, por tanto debemos realizar un filter para obtener el objeto en el queryset que queremos y posteriormente escribir el siguiente comando

```
1 Objeto.update(Atributo = NuevoValor)
```

### 5.3.5. Mysql con django

Django es un entorno de programación que da un paso más en cuanto al manejo de la base de datos. Django lo que hace es estar en un punto intermedio entre las bases de datos clásicas y las bases de datos orientadas a objetos. Desde el punto de vista del programador las consultas ya no se hacen de forma clásica, si no que debemos trabajar con objetos. Cuando realizamos una consulta, lo que django nos devuelve

es un array de objetos. Y cuando trabajamos con claves foráneas, en el manejo de bases de datos clásico, hacíamos un inner join y consultábamos el id de la tabla que necesitábamos en el where. Con el manejo de bases de datos con objetos, debemos o consultar el objeto que tiene una series de características y después pasarlo en otra búsqueda, o hacer una referencia en la búsqueda. A continuación realizaremos un ejemplo del empleo de ambas consultas.

1. Busqueda de un objeto referenciando el objeto a buscar en la búsqueda

```
1     pueblos = SigP.objects.filter(id_user_id = id_u).order_by("id_pueblo")
```

2. Busqueda de un objeto buscando el objeto que referenciaba antes

```
1     usuario = Usuario.objects.filter(id=id_u)[0]
2     pueblo = Pueblo.objects.filter(id=id_p)[0]
3     u = SigP(id_user=usuario, id_p=pueblo)
```

Este apartado quizás sea uno de los más extensos porque

En django, las direcciones accesibles se expresan en un fichero (urls) y se definen estas con una expresión regular. Si una dirección a la que intentan acceder a nuestro servidor django no cumple una de las expresiones regulares que tenemos definidas en el urls.py, nos saltará una excepción en el navegador. Además de la expresión regular, el código que se ejecuta en nuestro servidor web es un método definido y desarrollado en el views.py el cual explicaremos a continuación.

En definitiva el urls.py, es como el funcionamiento de un *switch-case*. Si la url cumple una expresión regular, ejecuta el método en el views.py que esté especificado en la misma linea.

```
1     url(r'\textasciicircum RegularExpresion/$', views.metodo, name='metodo')
```

## 5.4. Ejecución de código - /polls/views.py

Este realmente es nuestro quid de la cuestión. Cada método escrito en el views.py debe devolver un `HttpResponce()`. esta respuesta http, puede ser escrita en un fichero html a parte o bien (Nosotros únicamente queremos respuestas json para nuestra aplicación) enviar en una respuesta http, una cadena de texto en formato json. Al estar todo basado en un lenguaje de programación (Python) podemos usar los métodos de python para transformar las respuestas en json o bien podemos parsear nosotros el propio json para obtener lo que queremos en éste. La última opción será por la que

nos decantemos ya que por defecto nos da los datos y nombres de una consulta y nosotros queremos únicamente ciertos datos para no sobrecargar la respuesta json. También podemos pasar parámetros a estos métodos por la url, especificandolos en la expresión regular de urls.py.

## 5.5. Definición de parámetros en el proyecto - /mysite/- settings.py

En este fichero, tenemos los atributos del proyecto. A continuación pondremos los atributos más importantes pero no todos los que están especificados.

**INSTALLED\_APPS** Esta son las APIS que usan nuestra aplicación, debemos especificar aquí todas aquellas que vamos a crear. Por defecto vienen especificadas algunas aplicaciones nativas de django como la de administrador o la de autenticación.

**DATE\_INPUT\_FORMATS** Como su propio nombre indica, muestra como se manejarán las fechas y con qué formatos, en nuestro proyecto

**SECRET\_KEY** Usado para la codificación a la hora de entrar en la aplicación,loguearse como usuario, por ello, debe ser un valor no predecible y único. También es creado de forma automática aunque podemos cambiarlo.

**LANGUAGE\_CODE** El idioma que vamos a usar para nuestra aplicación.

**TIME\_ZONE** La zona horaria que usaremos para obtener fechas.

**DATABASES** Las bases de datos que usaremos en nuestro proyecto. En éste debemos especificar el motor, el nombre de la tabla, el nombre de usuario y su contraseña y por último, la dirección.

```
1 DATABASES = {  
2     'default': {  
3         'ENGINE': 'django.db.backends.mysql',  
4         'NAME': 'tabla',  
5         'USER': 'user',  
6         'PASSWORD': 'userpass',  
7         'HOST': 'host.uhu.es',  
8     }  
9 }
```

## 5.6. Creación de comandos personalizados

La ejecución de scripts en Django se realiza a través de la shell ejecutando el fichero manager.py. Para la ejecución de tareas periódicas se recomienda la creación de comandos para facilitar la ejecución automática de las tareas. Para la creación de los comandos se debe modificar la estructura de archivos creada durante la instalación de Django, para permitir la ejecución de comandos personalizados se debe modificar la estructura de la siguiente manera:

```

1 polls/
2     __init__.py
3     models.py
4     management/
5         __init__.py
6         commands/
7             __init__.py
8             _private.py
9             closepoll.py
10    tests.py
11    views.py

```

Siguiendo el ejemplo de la creación de la aplicación polls, la modificación incluye la creación de los ficheros *management* y *commands*. Dentro de esta última carpeta se incluyen los ficheros con el código que se ejecutará con el comando, el nombre del comando es el mismo que se le da al fichero. Si el nombre del fichero comienza con guión bajo (\_) el comando no se podrá ejecutar.

El código que incluye el fichero que contiene el código del comando es el siguiente:

```

1 import sys
2 sys.path.insert(0, '... django/mysite/Programas/Update/')
3 import mi_modulo
4 from django.core.management.base import NoArgsCommand
5
6 class Command(NoArgsCommand):
7     def handle_noargs(self, **options):
8         mi_modulo.run()

```

Con este código se crea un comando sin argumentos. Con el módulo sys se agrega a la variable PATH de Python la ruta al script que realiza la extracción de noticias, de esta forma luego podemos exportarlo como un módulo y ejecutarlo.

Para ejecutar el comando, lo utilizamos como argumento al ejecutar el fichero manage.py

```

1 python ./manage.py mi_comando

```

## 5.7. Uso de Django-extensions

Django-extension es un conjunto de comandos y extras que se añaden a la funcionalidad que ofrece Django. Una vez instaladas las extensiones, hay que agregarlas a la aplicación que se está creando. Para agregarla hay que modificar la variable `INSTALLED_APPS` que se encuentra en el fichero `settings.py`:

```
1 INSTALLED_APPS = (
2     ...
3     'django_extensions',
4 )
```

Una vez instalado, puede generarse el fichero dot que contiene un grafo creado a partir del modelo creado en django con el siguiente comando:

```
1 manage.py graph_models -a > my-project.dot
```

Con el fichero dot podemos obtener el grafo en formato PNG con el siguiente comando:

```
1 dot -Tpng my-project.dot > my-project.png
```

## 5.8. Anotaciones

Cuando investigamos no sabemos a priori los datos que vamos a necesitar de un objeto (Municipio en nuestro caso), y por tanto a medida que van surgiendo nos encontramos con que debemos modificar la tabla. Para esto hay un plugin llamado *South* cuya función consistía en migrar la base de datos. Esta opción quedó descartada debido a la gran cantidad de datos así que se optó por agregar el campo al models y de la misma forma agregarlo a partir de la herramienta de MySQL *PHPMyAdmin* ya que los cambios se iban haciendo para un campo o dos, pero al final recurrimos muy a menudo a esta solución ya que la mayoría de cambios se hicieron en la tabla de pueblo y esta tabla no era conveniente borrarla ya que los datos no eran reproducibles en un corto plazo debido a que algunos de los campos eran llenados a partir de datos buscados con un script que usaba la API de búsqueda de google como hemos referenciado en este capítulo. Otra solución para este problema que se nos presentaba era borrar la tabla (Si no tenía este tipo de datos y volver a llamar al comando syncdb. Otra herramienta muy potente de django que no hemos usado es *TastyPie*. Esta herramienta es la que nos ayudaba en la creación de la RestAPI. Pero decidimos no usarla por la descentralización y poder devolver los errores de la forma que queríamos. En los estudios realizados sobre esta herramienta no hemos visto la posibilidad de

realizar estas acciones y se hicieron varias pruebas de decodificación de los datos que nos devolvía la *API* con *TastyPie* y el método de codificación que nos daba *Python*, y no funcionaron correctamente por lo que decidimos realizar nosotros la conversión en *JSON*

Por último una de las cosas más importantes a la hora de la preparación de los datos es la paginación. Es una funcionalidad que agregamos al proyecto una vez que éste estaba con las funcionalidades básicas implementadas y con un correcto funcionamiento. Posteriormente nos dedicamos a pulir las funcionalidades y una muy importante era ésta. No debíamos traernos por ejemplo todas las noticias ya que esto hacía que la aplicación tardara mucho a la hora de realizar esta consulta porque el pueblo que más noticias tiene son 3800. Entonces teníamos que diseñar el sistema de paginación y trabajar con él. Al principio creamos uno manual en el que le íbamos indicando de qué a qué noticia me traía, pero si un futuro queremos poner publicidad en nuestra API en la página web, y la gente puede entrar y traerse toda la información, no tendrá muchas visitas. Del mismo modo, tenemos información que hemos recopilado y tratado. Este proceso nos ha llevado mucho tiempo, y si un usuario puede obtener todos los datos de la base de datos en un sólo click sería contraproducente para nosotros ya que todo nuestro trabajo se podría copiar con mucha facilidad.

# API

---

El segundo método de publicación de la información que usaremos es una API Rest, así el usuario únicamente con visitar nuestra web podrá obtener los datos que hemos recopilado.

Una API es un conjunto de llamadas a ciertas bibliotecas y ofrecen un cierto servicio. Existen muchas API conocidas como las de Google (*Geolocalización, Custom Search, etc ...*), Wikipedia, etc ... Una API se convierte en una API REST cuando nuestra API puede ser utilizada por cualquier dispositivo o cliente que entienda HTTP, así que podríamos decir que REST es un tipo de arquitectura más natural y estandar para crear APIs

Existen tres niveles de calidad a la hora de aplicar REST en el desarrollo de una aplicación web que se recoge en un modelo llamado **Richardson Maturity Model** Y los tres niveles son:

**Uso correcto de las URI** La estructura de la URI debe ser la siguiente:

```
1 {protocolo}://{hostname}[:puerto]/ {ruta del recurso} ?{consulta de filtrado}
```

También existen varias restricciones a la hora de crear las URIs:

1. Los nombres de la URI deben implicar una acción por tanto deben evitarse usar verbos en ellos.
2. Deben ser únicas, no debemos tener más de una URI para identificar un mismo recurso.
3. Deben ser independiente de formato.
4. Deben mantener una jerarquía lógica.

5. Los filtrados de información de un recurso no se hacen en la URI.

**HTTP** Existen varios puntos que tenemos que tener en cuenta a la hora de la creación API REST, como los métodos de HTTP (GET, POST, PUT, DELETE, PATCH) los códigos de estado para consultar cual es el problema y la aceptación de tipos de contenido. Nosotros hemos dado una vuelta más al uso de la API en este sentido y en lugar de estos métodos, desde nuestra API únicamente dejamos hacer llamadas GET. Y éstas serán las llamadas públicas a la API. Digamos que también le hemos dado una vuelta al uso de estos recursos y hemos intentado hacer que la API sea mucho más sencilla para el usuario.

**Hypermedia** El lenguaje en el que está escrito el resultado de la llamada a la API, Nosotros usaremos JSON.

## 6.1. JSON

Para la parte de la publicación de los datos debíamos elegir un formato y nos decidimos por JSON.

JSON, **Java Script Object Notation**, está siendo uno de los formatos más usados por todas las API's debido a la simplicidad de éste.

Aunque XML es más extensible, dependiendo del tipo de información que vamos a transferir, ésta cualidad nos puede hacer más o menos falta. Para nosotros, por nuestra forma de trabajar, utilizamos únicamente datos clásicos así que no nos hace falta la extensibilidad de XML y por tanto podemos aprovecharnos de otras cualidades de JSON como por ejemplo legibilidad.

La superioridad de JSON a la hora de tratar datos básicos, es el almacenamiento en vectores y registros en lugar de árboles como hace XML. Por tanto aprovechamos la facilidad a la hora de importar estos datos a objetos en lenguajes de programación. Para hacer lo mismo con xml, deberíamos transformar los datos antes de importarlos. Por eso JSON es el formato más utilizado en las APIs y por ese motivo lo usaremos nosotros.

## 6.2. Llamadas

Todas las llamadas tendrán la estructura definida anteriormente con protocolo http, dirección y puerto 8000 (Es el que django abre por defecto aunque podemos cambiarlo si fuera necesario). Como hemos dicho también, existen unas URIs Públicas y otras privadas, aquí pondremos todas las llamadas públicas en la url seguido de su valor de retorno. No se hará así para las privadas ya que no es necesario porque nos devuelve en JSON, un objeto con una variable booleana con valor False si se ha producido algún error y True si la operación se ha completado sin fallos.

A continuación pondremos dos de las llamadas más comunes y la estructura que deberían devolver junto a un ejemplo.

**/api/pueblos/:id** Esta llamada muestra todos los atributos de un pueblo con el id que le indiquemos. Existe un número total de 8118 pueblos, por tanto el id debe estar entre 1 y 8118 en un intervalo cerrado.

```

1      {
2      "id": 11,
3      "dsprovincia": "Cádiz",
4      "npueblos": 1,
5      "pueblos": [
6          {
7              "id": 1766,
8              "dsueblo": "Alcalá de los Gazules",
9              "coordenadas": {
10                  "longitud": -5.7245344,
11                  "latitud": 36.4617479
12              },
13              "url": "http://www.alcaladelosgazules.es/",
14              "opencms": true,
15              "habitantes": 5439,
16              "deuda": 5697.33,
17              "deudaxhab": 502.41005291,
18              "fecha_inscripcion": "1986-11-26",
19              "superficie": 479.59,
20              "wiki": "http://es.wikipedia.org/wiki/Alcalá_de_los_Gazules",
21              "cp": 11001,
22              "n_noticias": 228
23          }
24      ]
25  }
```

**/api/noticias/:id** Este método nos muestra la noticia con el ID que le indiquemos.

Los saltos de linea se sustituyen por el carácter - al no ser compatible con json. tanto en el campo cuerpo como en el campo titular. Se recomienda volver a cambiarlo a la hora de capturar la consulta json.

```

1  {
2      resultado:[
3          {
4              "id_noticia": 1,
5              "titular": "string",
6              "fecha": "string",
7              "url": "string",
8              "cuerpo": "string"
9          }
10     ]
11 }
```

```
8     liked :"integer",
9     dspueblo :"string",
10    ncomentarios :"integer",
11    categoria:[
12      {
13        id_categoria :"integer",
14        dscategoria :"string"
15      }
16    ],
17    vista :"boolean"
18  }
19 ]
20 }
```

## Android

---

Android es un sistema operativo basado en el núcleo Linux. Fue diseñado para dispositivos móviles con pantalla táctil. Fue desarrollado por *Android Inc.* empresa que en un principio respaldó Google y terminó comprando en 2005. Android fue presentado en 2007 con la colaboración de **Open Handset Alliance**.

Tiene una gran comunidad de desarrolladores de aplicaciones y ha llegado al 1.000.000 de aplicaciones de las cuales dos tercios son gratuitas y más baratas que las mismas en la App Store.

El lenguaje de desarrollo de Android es Java

A la hora de decidirnos por android para desarrollar la aplicación en una plataforma en concreto, hicimos un estudio de mercado y como podemos ver en XakataAndroid Android en 2014 ya superó a iOS en usuarios, por tanto Android al igual que hizo google en su época se está expandiendo y de momento no tiene techo. También podemos ver en la imagen (Ver figura 7.1 de la fuente RedUsers el porcentaje de usuarios de sistemas operativos en dispositivos móviles.

Otro aspecto que hemos tenido en cuenta es la diferencia del desarrollo de aplicaciones para Android y para iOS. Es cierto que la programación en Android no es algo trivial, pero aun así es más sencilla que en iOS, aunque veríamos satisfechos los frentes de actuación de la aplicación englobando estas dos tecnologías. La implementación en iOS es parte de los trabajos futuros.

### 7.0.1. Aspectos a tener en cuenta

**¿Manejo de base de datos?** Realmente en la aplicación de Android no viene ningún manejo de las bases de datos propias de la API. Por tanto los datos se obtienen todos por llamadas a ésta. El único manejo de base de datos que debemos saber, únicamente por optimizar tiempos de respuestas y no hacer perder el tiempo al usuario, es SQLite.

**Actividades asíncronas** Todas las actividades, tienen dentro su correspondiente actividad asíncrona, es decir, a una clase siempre le corresponde una actividad asíncrona. A continuación explicamos el por qué hemos optado por este tipo de implementación

**Llamadas a API** Como hemos dicho, no manejamos ningún tipo de base de datos externa, y únicamente hacemos uso de las llamadas a la API. Una llamada a una dirección web. La gestión de este tipo de llamadas en Android debe hacerse en una función asíncrona.

**Tiempo de espera** Cada llamada a la API tiene un delay, por tanto si no queremos que aparezca de la nada datos en la pantalla, debemos poner un mensaje de sincronización y de espera. Esto se hace creando una actividad asíncrona.

**Extensión de adaptadores** Debido a las necesidades de la aplicación no podíamos utilizar un adaptador para listas predeterminado por tanto teníamos que crear uno específico para cada tipo de dato mostrado. Nosotros hemos creado 5 adaptadores distintos:

1. Pueblos (Pueblos + escudo o bandera)
2. Noticias (Varios textView para titular, fecha, nº comentarios, etc ... )
3. Inicio (Parecido al de noticias, pero este muestra el contenido de los comentarios si los tiene)
4. Categorías (Un String con un botón de eliminación)
5. Comentarios (Igual que el de noticias pero con otra estructura y formato)

Además de estos adaptadores, hemos creado uno para gestionar la correcta visión de la funcionalidad Amigos y aunque funcional, está en revisión para agregar la funcionalidad completa en trabajos futuros.

## 7.1. Funcionalidades de la aplicación

En el apartado de la publicación de la información, decidimos que, además de una simple página web para consultar resultados, en el tema de clasificación la ayuda de un usuario podría ser más que útil. Pero para que un usuario nos ayude en la clasificación, debemos darle acceso a ello de una manera sencilla e intuitiva. A continuación pondremos las funciones básicas que un usuario puede realizar en nuestra aplicación y explicaremos el porqué.

**Loguearse/registrarse** Aún pudiendo usar una técnica de logueo como OAuth2, decidimos hacerla con un usuario y una contraseña, pos sencillez en el desarrollo de la aplicación. En los trabajos futuros se especifican los cambios respecto a estos apartados.

Debido a unas especificaciones iniciales que cambiaron a posteriori, a la hora de realizar el registro, se debe indicar el nombre, y los apellidos, el resto de campos son el nombre de usuario, el pueblo principal y el correo electrónico.

A la hora de loguearse es necesario especificar el usuario y la contraseña.

**ISSUE.** Existe un problema a la hora de enviar y la contraseña y es que esta no se manda encriptada, si no que lo hace en texto plano. La solución se explica en trabajos futuros.

**No Logueado página inicio** Un usuario que no esté logueado, verá como pantalla principal cinco noticias elegidas al azar entre las últimas 20. Esto se hace para que si no existen noticias nuevas, el usuario pueda ver algún cambio en las noticias principales, ya que al usuario sin registrar no se les permite ver todas las noticias de las que disponemos.

**busqueda de noticias** Un usuario no registrado también puede buscar noticias igual que un usuario logueado. No le queremos quitar mucha funcionalidad al usuario que no esté registrado para hacer la aplicación más atractiva y así poder lograr que se registren y nos ayuden a clasificar noticias. La búsqueda, puede hacerse por titular, fechas o categoría.

**Visor de noticias** Un usuario podrá ver el contenido de una noticia y su clasificación. También podrá ver la noticia en su página web, pero no podrá votar una noticia y aunque pueda ver las categorías de una noticia, tampoco podrá acceder a la página de edición de categorías.

**Visor de comentarios** Al igual que con las noticias, un usuario podrá ver

todos los comentarios que existen mas no podrá ni denunciar comentarios ni realizar comentarios nuevos en una noticia.

**Acceder a la página de información** Es algo que no hemos visto necesario poner una vez el usuario está logueado ya que, además de que este apartado nunca se ve en las aplicaciones, o muy pocas veces, la gente suele revisarla al principio. Nosotros hemos puesto un correo de contacto, y el porqué de esta aplicación y donde tuvo su origen.

### Actualizar categorías

**Logueado** Además de todas las funciones a las que puede acceder un usuario no logueado.

**Página de inicio** A diferencia del usuario no logueado, el usuario logueado podrá ver en su pantalla principal, un cartel de Bienvenida junto a su usuario, su municipio principal, (Con acceso a ver su propio perfil) y las últimas 5 noticias de su municipio principal, con un '+' para ver más noticias de este municipio.

**Página de perfil** En principio esta página se creó para que un usuario pueda cambiar su municipio principal. El resto de atributos en principio no creímos convenientes cambiarlos salvo quizás el correo electrónico cuyo posible cambio está en los trabajos futuros.

**Búsqueda de municipios y su información** Si un usuario está logueado, puede realizar una búsqueda de municipios y todos los datos que almacenamos en nuestra base de datos sobre éste. Una vez que accede a la pantalla de información del municipio, el usuario si así lo desea, puede agregar el pueblo a favoritos, ver su geolocalización en Google Maps, buscar otro pueblo y por último ver sus noticias (si tuviera), si no tiene, la búsqueda de las noticias se hará sobre los pueblos colindantes a éste.

**Navegación de noticias** Un usuario logueado, a parte de ver las noticias del pueblo al que sigue, existe un navegador que le permite ver Todas las noticias disponibles en nuestra BDA, todas las noticias los pueblos a los que sigue y las noticias de estos pueblos separadas. Además, para el usuario registrado guardamos un histórico de noticias vistas para una función que explicaremos en trabajos futuros.

**Votación de noticias** Además ver las noticias, si un usuario está registrado podrá votar una noticia positivamente, Este voto también lo guardamos para estadísticas y posibles recomendaciones futuras, pero ante todo lo

dejamos implementado para tenerlo preparado cuando hagamos la integración con **Facebook**

**Denuncias e inserción de comentarios** Un usuario registrado podrá además de ver los comentarios, denunciar comentarios que ya existan. Si un comentario obtiene más de 20 denuncias, éste será eliminado. Hemos decidido poner un número fijo en lugar de un porcentaje debido a que si se incrementa el número de usuarios demasiado, se necesitarían muchos votos para eliminar un comentario. Si un usuario denuncia su propio comentario, éste será borrado avisando al usuario previamente.

**'Semantizador'** Esta quizá sea la parte más importante del resultado de la interacción entre el usuario y la APP. Aquí es donde un usuario puede votar que una categoría es errónea, para realizar la validación de la votación sobre una categoría errónea, se seguirá el mismo método que con los comentarios. Esto nos ayudará a ver que clasificador de los tres utilizados obtiene mejores resultados a la hora de clasificar una noticia en distintas categorías, ya que se almacena un registro de que categoría se asignó a que noticia y quien fué el que la categorizó. Además de votar categorías erróneas, un usuario podrá clasificar una noticia si éste opina que ésta corresponde a una categoría a la cual no ha sido asignada, por tanto, podrá o bien agregarla a partir de nuestra lista de categorías o por otra parte, crear una nueva.

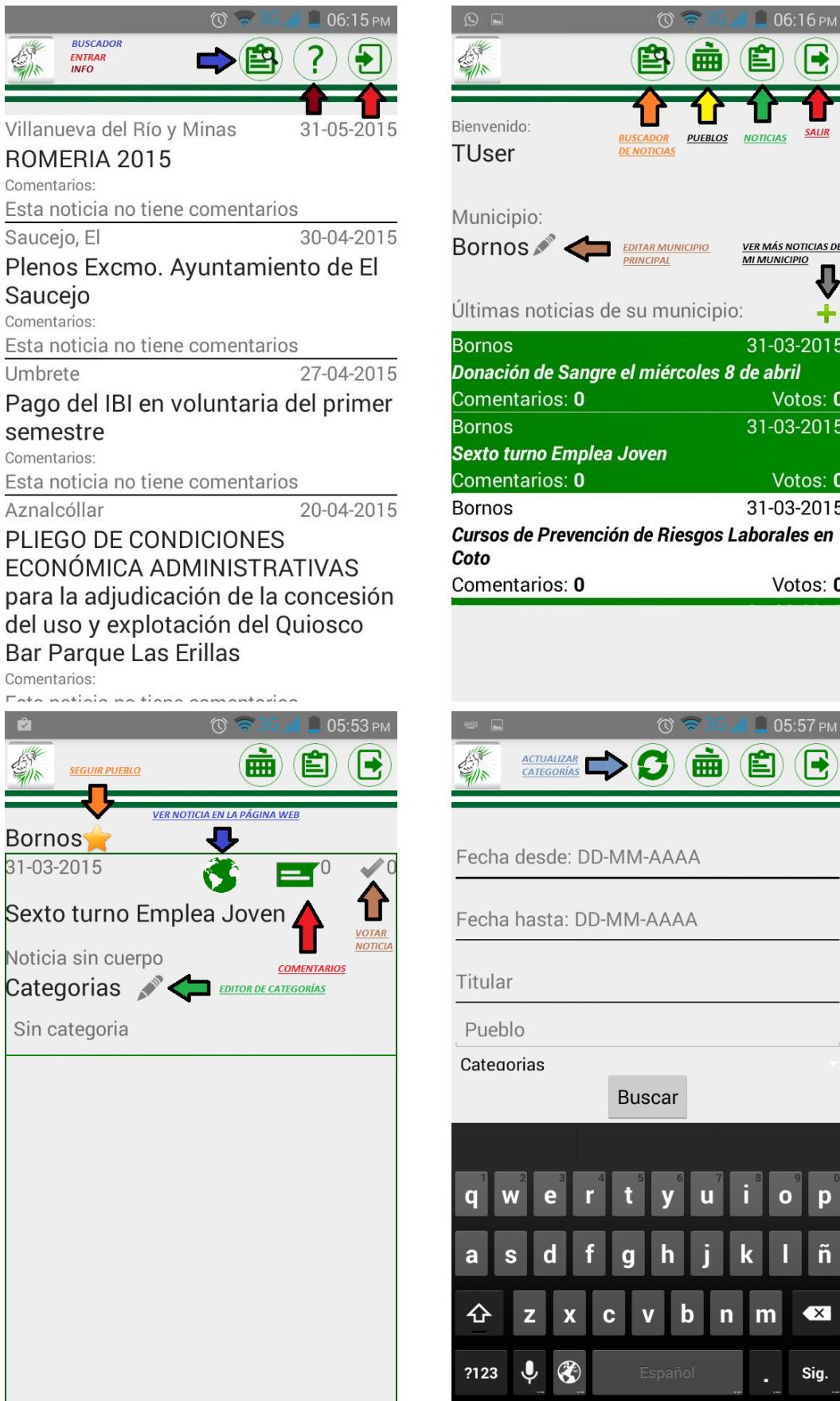
Para terminar el apartado de funcionalidad de la aplicación, también debemos decir que el último usuario logueado se guarda cuando nos deslogueamos. Únicamente guardamos un usuario.

## 7.2. Funcionalidades auxiliares

En la decisión de la estructura de la aplicación Android, habíamos abierto una vertiente que quedó en ser una parte futurible del proyecto pero no era el enfoque principal de éste. La funcionalidad de la que hablamos es un manejo social de los usuarios. Que estos puedan buscarse entre sí y ver los pueblos a los que siguen. Pero como hemos dicho antes, esta opción se desestimó por desviarse demasiado del objetivo principal de la aplicación.



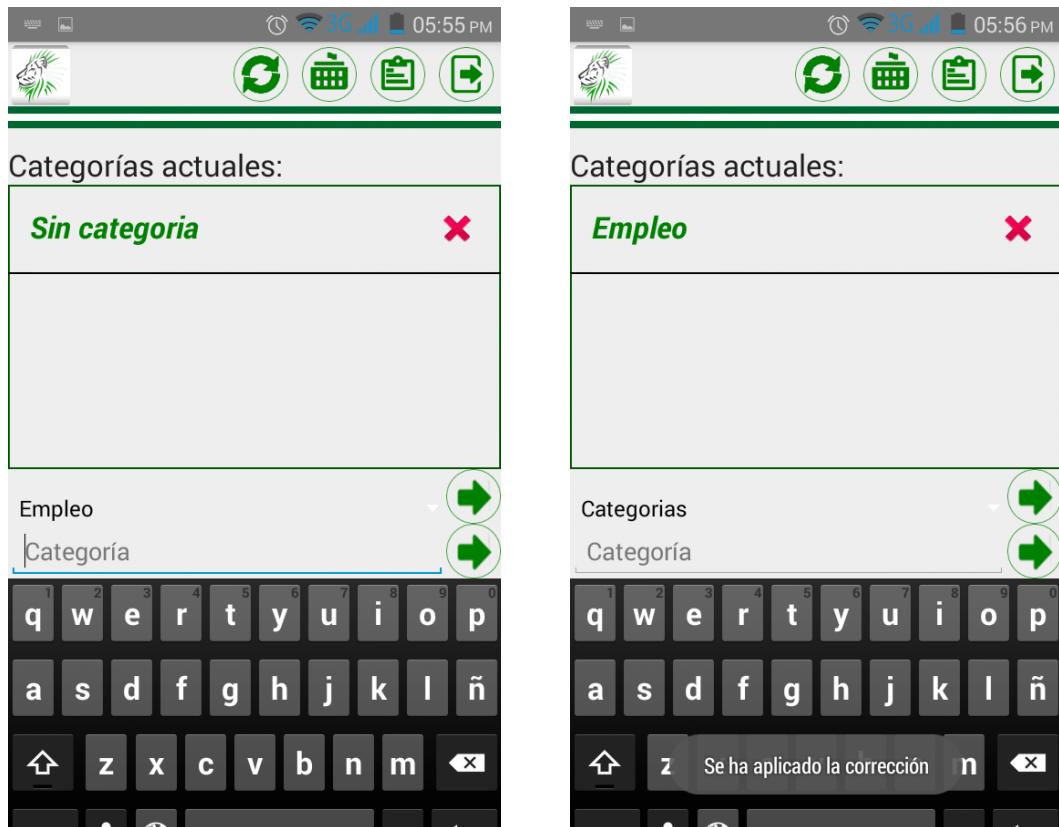
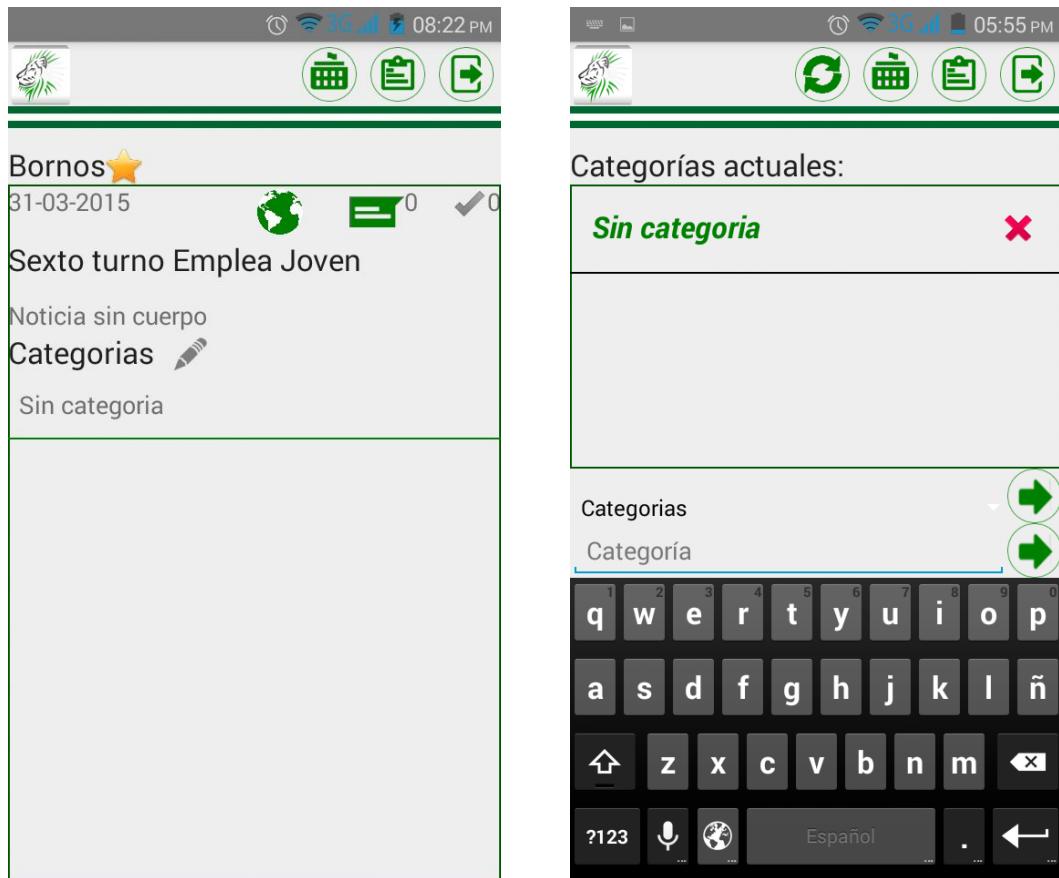
### 7.3. Elementos de navegación

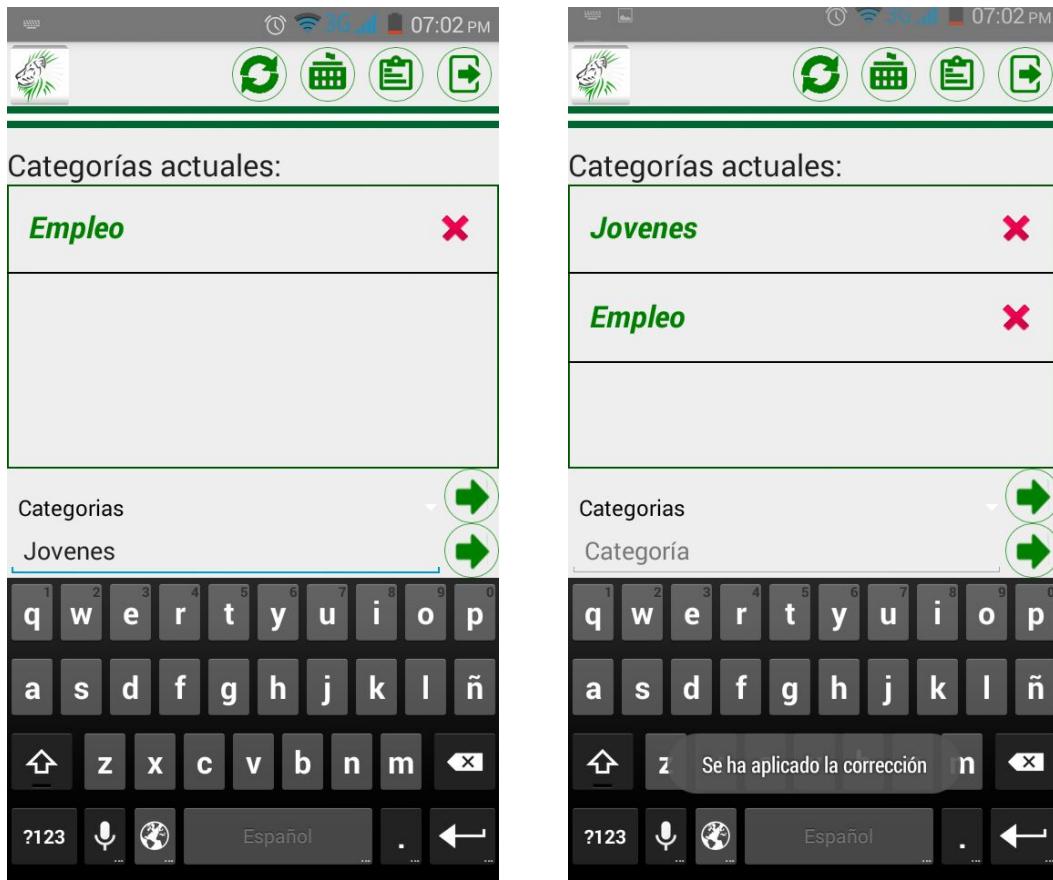




## 7.4. Casos de uso

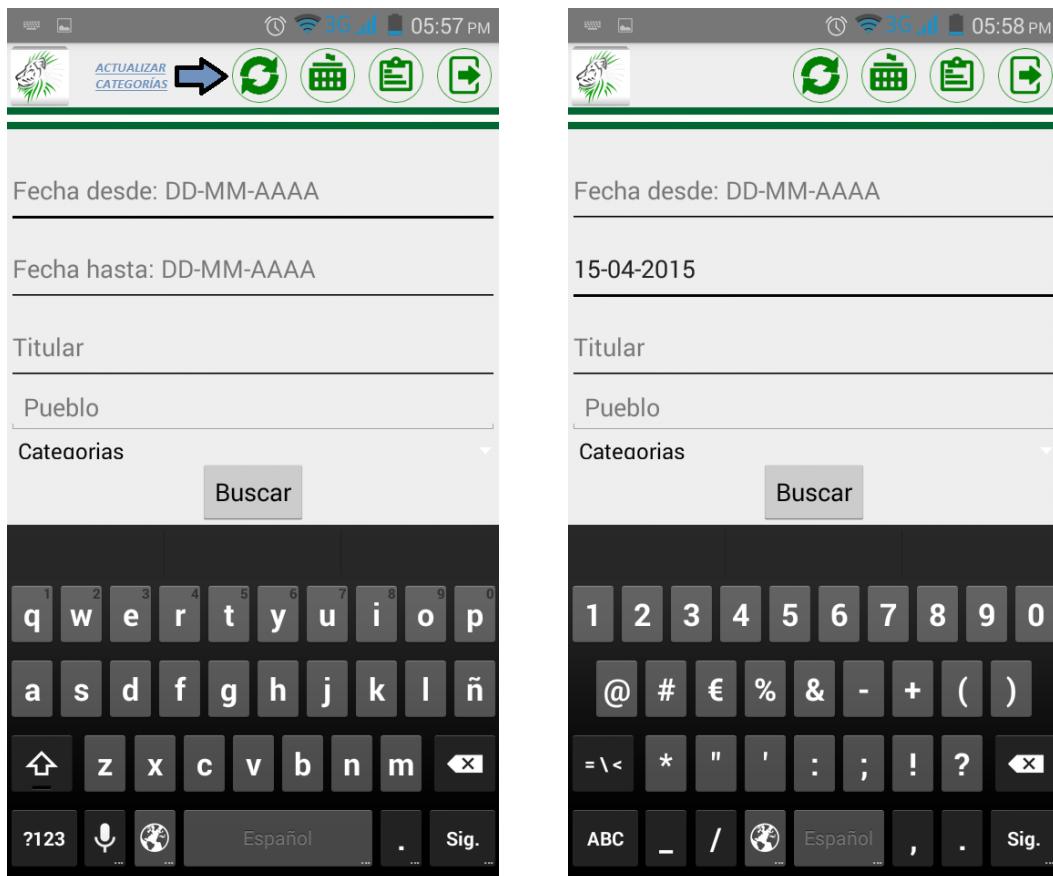
### 7.4.1. Categorizar noticia







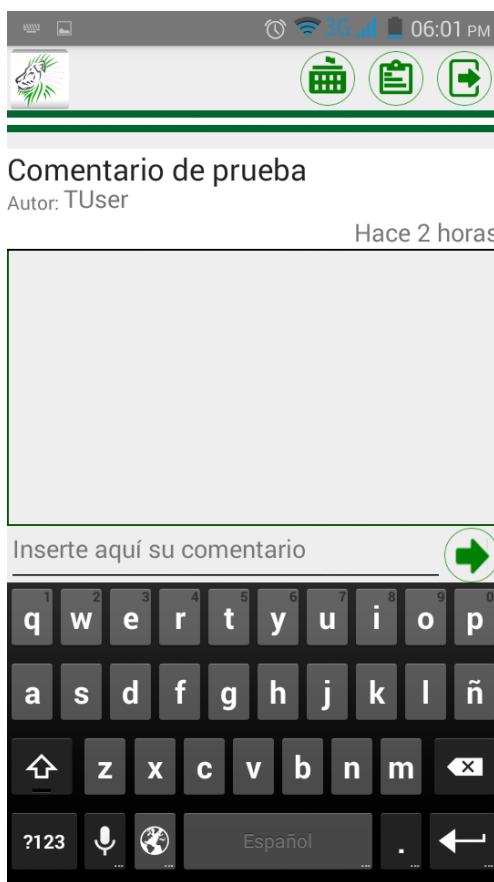
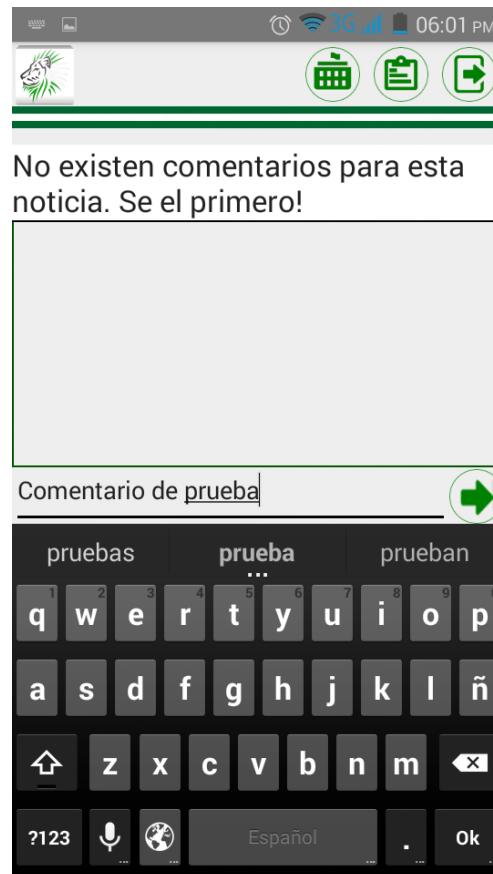
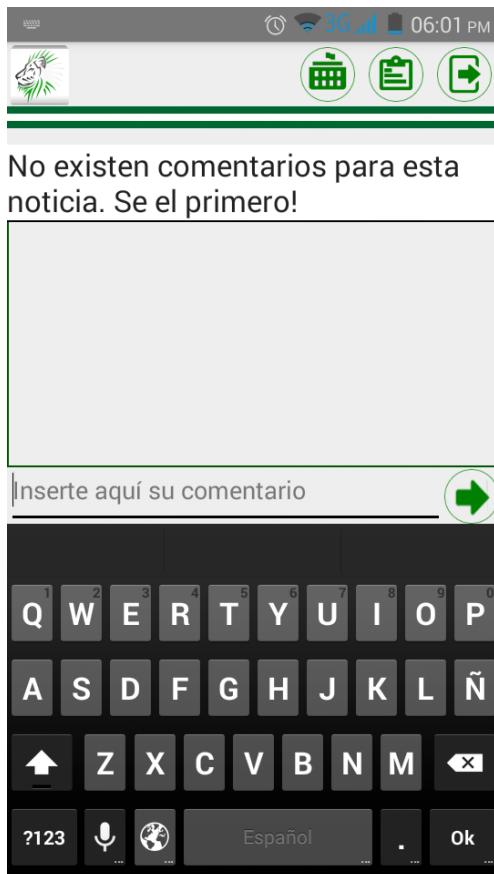
### 7.4.2. Busqueda Noticias



*Esta consulta no tiene más noticias*



### 7.4.3. Realizar comentario



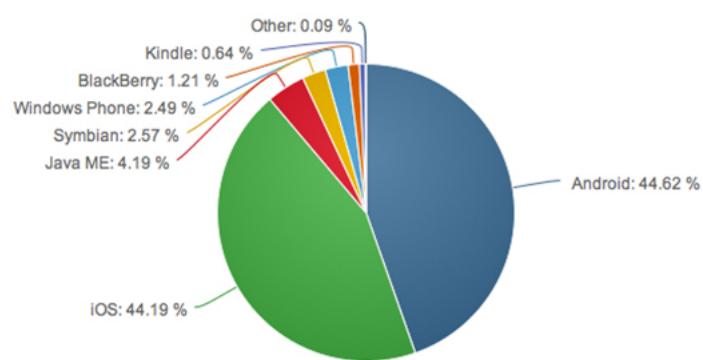


Figura 7.1: Porcentaje de usuarios de los distintos SO móviles



## **Parte IV**

# **Arquitectura de la solución**



# Servidor

---

## 8.1. Tareas programas en el servidor

### 8.1.1. Actualización automática de los datos

Una vez tenemos un primer conjunto de datos es posible que alguno de ellos cambie con el tiempo por lo que debemos de, periódicamente, lanzar un proceso que compruebe si los datos almacenados están actualizados. Para realizar esta tarea crearemos en el servidor que mantiene la aplicación una serie de tareas que se ejecutarán de manera automática.

Esta acción es de especial relevancia en el caso de las noticias, ya que a partir de nuevas noticias podremos intentar los sistemas de clasificación creados. El proceso de búsqueda de nuevas noticia y la clasificación de aquellas noticias de las que no conocemos es el siguiente.

La primera tarea es la búsqueda de noticias que no tengamos almacenadas por haber sido publicadas después de la última búsqueda realizada, como el iterar entre los distintos municipios es una operación que requiere tiempo se realizará esta tarea cada 48 horas.

Junto a la tarea anterior debemos ejecutar otra tarea que consideramos una continuación de la anterior, esta es el asignar una categoría a aquellas noticias que hayamos extraído y no tengan una categoría en la web de su municipio y la aplicación de la ontología que hemos creado.

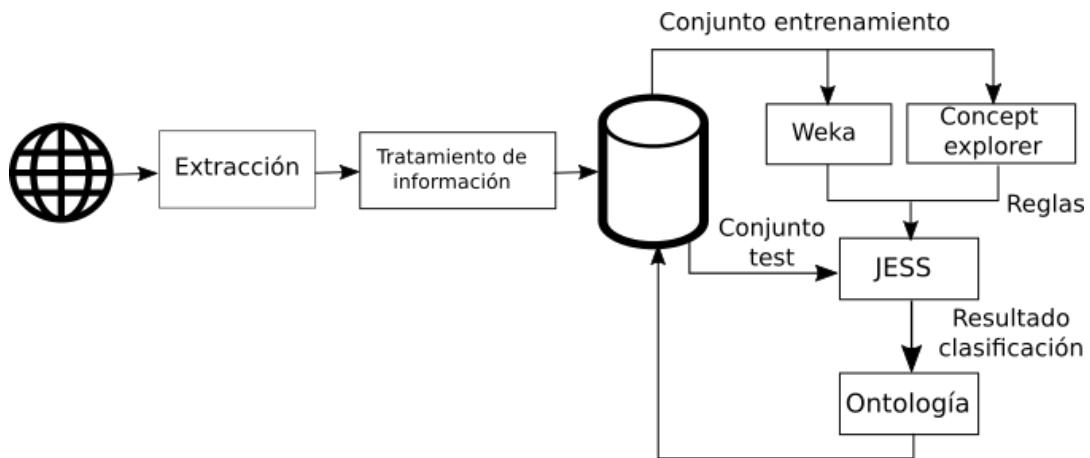


Figura 8.1: Infraestructura del sistema

### 8.1.2. Realización de copias de seguridad de la base de datos

En prevención de un fallo en el gestor de la base de datos u otro fallo del propio servidor, una vez por semana se hará una copia completa de la base de datos para poder recuperar la aplicación en caso de producirse algún fallo.

### 8.1.3. Eliminación de comentarios considerados negativos

Una vez cada 24 horas, aquellos comentarios que los usuarios, por votación, consideren negativos serán eliminados si un comentario tiene el número mínimo de votos negativos que consideramos para proceder a su eliminación

### 8.1.4. Corrección de categorías erróneamente asignadas

De la misma forma que la tarea anterior, una vez al día, se harán correcciones sobre las categorías asignadas a las noticias si un número mínimo de usuarios, a través de la aplicación, marcan que una o varias de las categorías asignadas a una noticia no son correctas.

## 8.2. Base de datos

A continuación se describe como hemos almacenado la información extraída en la base de datos. Para una vista en detalle de las tablas véase el anexo A

### 8.2.1. Noticias y municipios

(Ver figura 8.4) Con respecto a las noticias y pueblos almacenamos la siguiente información. Una noticia queda definida por los siguientes parámetros: Municipio al que pertenece, titular, resumen, cuerpo, fecha, dirección a la web donde se encuentra y una categoría, como información asociada a la noticia tenemos los comentarios que los usuarios pueden realizar sobre ella.

En cuanto a los municipios guardamos una serie de datos como son, nombre, provincia a la que pertenece, número de habitantes, etc. Los parámetros que definen un municipio se encuentran detallados en el capítulo Obtención de municipios y sus datos. Para mostrar noticias en el caso de que no tengamos noticias de un municipio que pueda seleccionar el usuario, guardamos para cada municipio sus cinco municipios más cercanos para poder mostrar, en este caso, noticias cercanas. Además almacenamos información adicional extraída desde el Instituto de estadística y cartografía de Andalucía<sup>1</sup>. Para poder llevar un seguimiento de la actualización de las noticias guardamos información sobre cuando hacemos un intento de buscar nuevas noticias en los distintos municipios y saber el resultado del proceso para poder, reiniciar el proceso desde el municipio que no se pudo completar.

## 8.3. Clasificación y categorías

(Ver figura 8.3) En la parte de clasificación guardamos la lista de categorías creadas en el capítulo Búsqueda de categorías que categoría o categorías se le asignan a una noticia, qué método de clasificación o usuario ha sugerido una o varias categorías para una determinada noticia, el grafo creado para la jerarquía en 2.1 y las equivalencias entre las etiquetas que podemos encontrar en las páginas de los municipios y nuestras categorías.

---

<sup>1</sup><http://www.juntadeandalucia.es/institutodeestadisticaycartografia>

## 8.4. Usuarios

(Ver figura 8.5) De los usuarios almacenamos una serie de datos básicos como un nombre de usuario y una contraseña. Por la funcionalidad que ofrece la aplicación Android, almacenamos los pueblos que quiere seguir el usuario, los comentarios que realiza y las denuncias que puede hacer sobre comentarios de otros usuarios que considere negativos, las noticias que visita y aquellas que además las valore de forma positiva y las correcciones que haga sobre las categorías asignadas a una noticia.

### 8.4.1. Control

Las tablas que almacenan información de control son las siguientes:

Llamadas		Versions	
<b>id</b>	<b>AutoField</b>	<b>id</b>	<b>AutoField</b>
contabilizacion	PositiveIntegerField	tabla	CharField
llamada	CharField	version	PositiveIntegerField

Figura 8.2: Tablas de control

En entras tablas almacenamos información estadística sobre las llamadas que se hacen a la API y las modificaciones hechas a las tablas para que desde la aplicación Android se puedan detectar estás modificaciones y actualizar algunos datos.

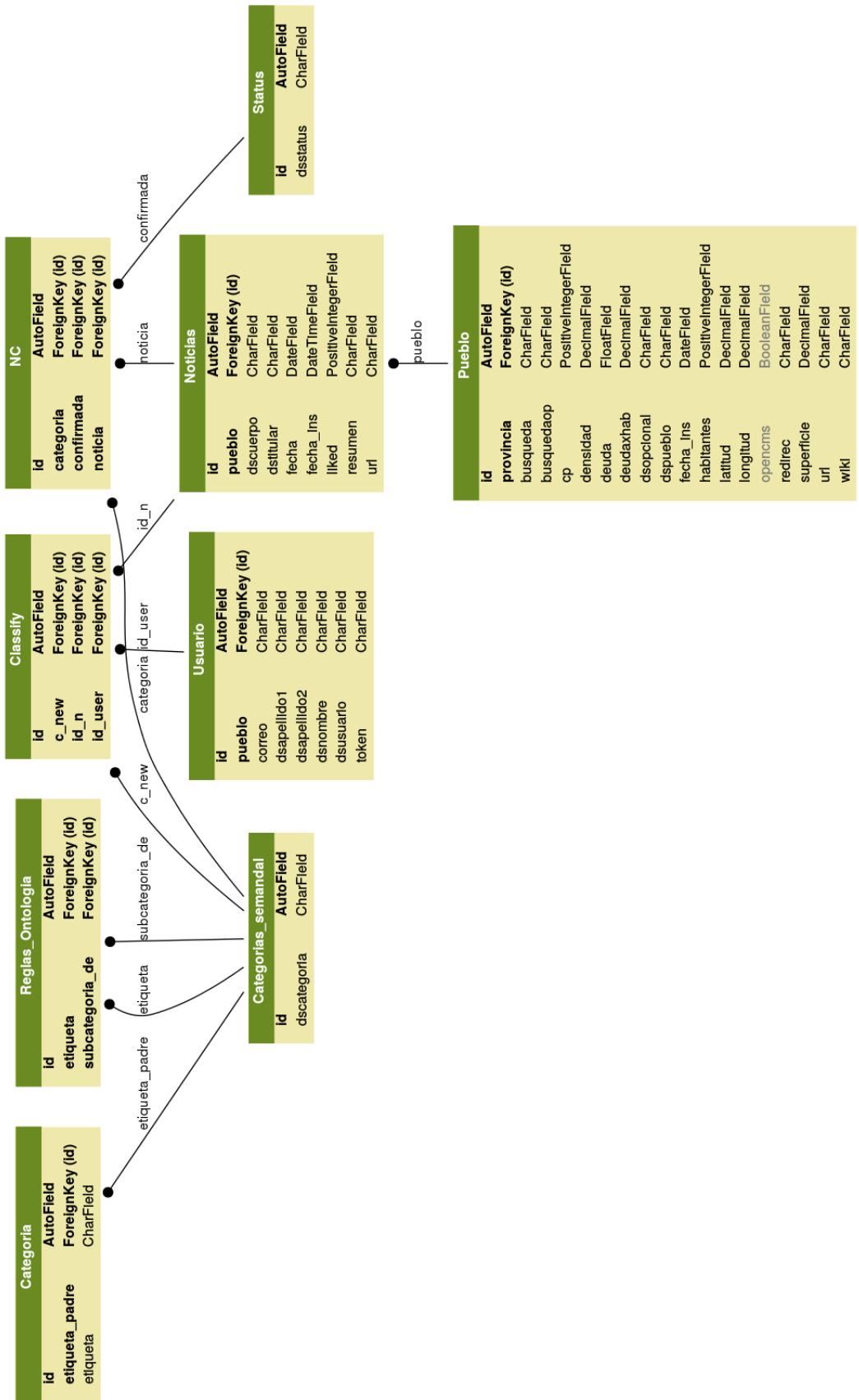


Figura 8.3: Tablas de clasificación y categorías

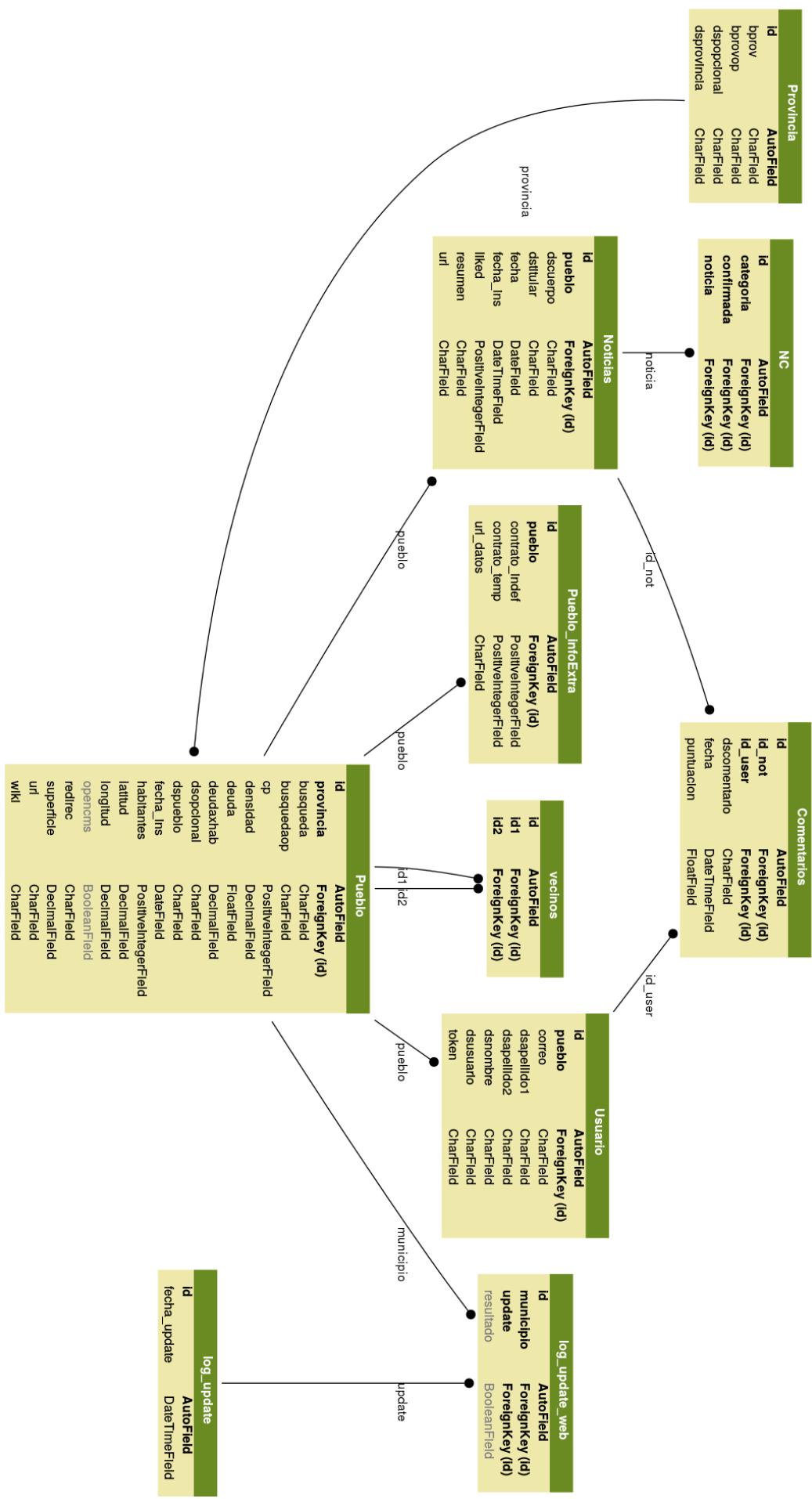


Figura 8.4: Tablas de noticias y pueblos

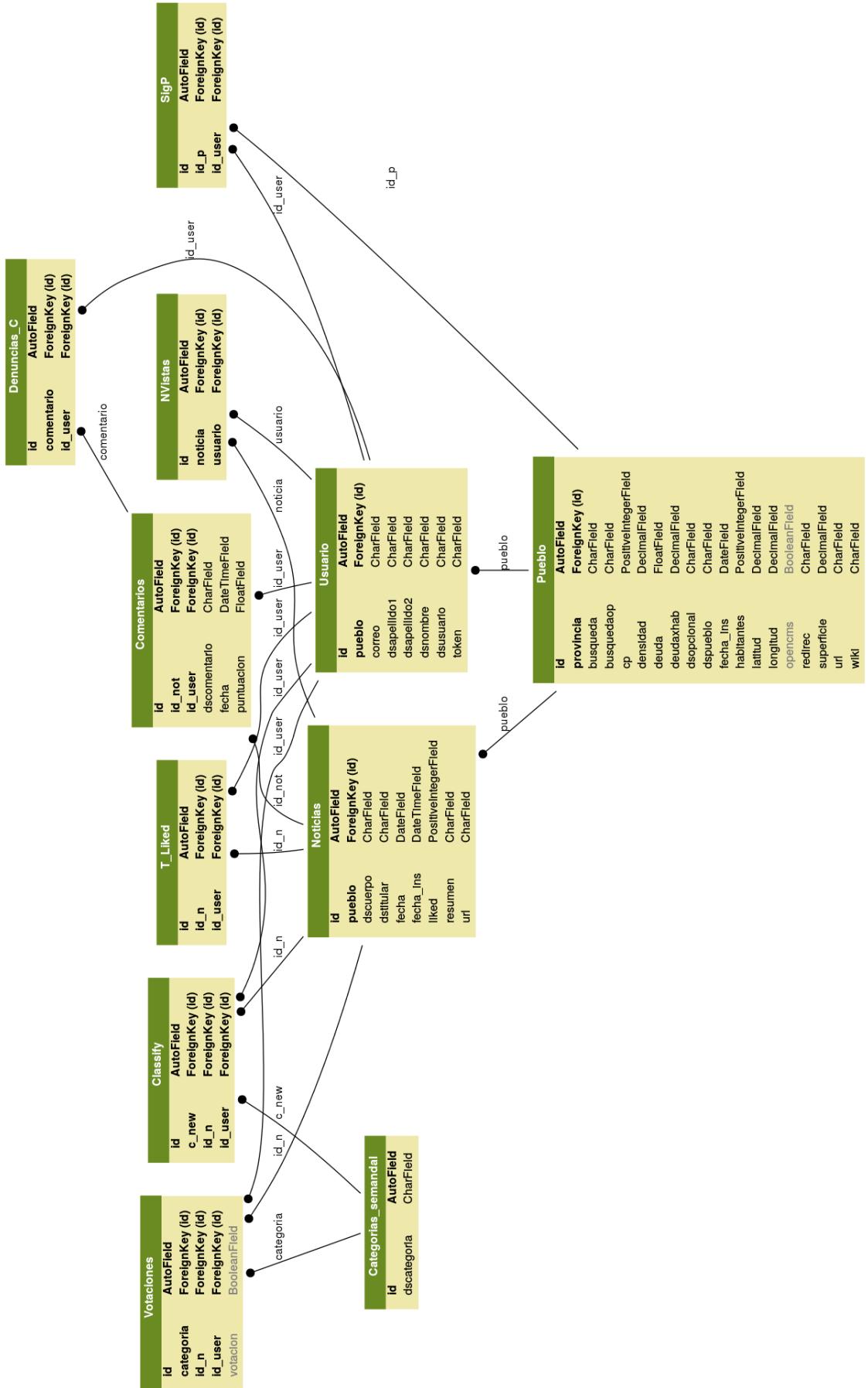


Figura 8.5: Tablas de usuarios



# **Parte V**

# **Conclusiones**



# Conclusiones

---

## 9.1. Obtención de datos

Íbamos a centrarnos en un primer intento a todos los municipios españoles, después a todos los andaluces y por último a todos los andaluces cuyo sitio web estuviese desarrollado bajo el gestor de contenidos OpenCMS. Incluso usando este mismo gestor de contenido existen 6 estilos diferentes de desarrollar el sitio. Además la información que encontramos en las páginas no es del todo correcta (como por ejemplo las noticias encontradas del año 13 o del 2019). Para facilitar la consulta de datos deberían o bien tener una API cada municipio en su página web o usar una estructura parecida. Hemos podido comprobar que los que guardan una estructura similar son aquellos desarrollados por empresas públicas, pero en un primer momento supusimos que al trabajar bajo una plantilla que la junta ofrecía gratuitamente, la gente desarrollaría su página web bajo esta plataforma y de manera parecida. El trabajo con las APIs de terceros ha sido satisfactorio y el único inconveniente que vimos fue el trabajo con el Custom Search de Google que te deja 100 consultas al día únicamente.

## 9.2. Clasificación

Para la clasificación, comenzamos utilizando análisis formal de conceptos y árboles de decisión, posteriormente se repitieron los experimentos con el algoritmo RIPPER. En cuanto a las herramientas tanto Concept Explorer como WEKA son sencillas de utilizar y permiten integrarlas en un proyecto Java o utilizarlas mediante una interfaz gráfica, lo que da bastante flexibilidad para trabajar con ellas. En cuanto a los métodos empleados, FCA y árboles de decisión parecen ofrecer mejores resultados

que RIPPER, por lo que de volver a crear un sistema de clasificación solo utilizaría los dos primeros.

### 9.3. Android

La elección de android en contrapartida a django si fue acertada, por tanto volveríamos a desarrollar para esta plataforma. Su desarrollo es en un lenguaje con el que estamos familiarizados, java, y existe mucha información en internet por si nos surge una duda, casi siempre el problema que vas a resolver ya existe. Cuando empezamos a desarrollar en android quizás no es una tecnología sencilla ya que hay muchas funcionalidades que se deben realizar de una manera específica como por ejemplo, el acceso a un contenido en internet debe hacerse desde una actividad asíncrona. Pero una vez que tenemos cierto manejo con ella, sobre todo cuando empezamos a desarrollarla desde 0, y por tanto coger mucha más destreza que si nos los dan hecho, las funcionalidades son casi inmediatas. La parte del desarrollo android que nos llevó más tiempo, una vez que acabamos el proceso de estudio e investigación sobre la herramienta, fue hacer que la parte visual fuera atractiva. También hemos aprendido mucho en este sentido ya que hemos tenido que investigar diseños aunque al final apostamos por uno propio.

### 9.4. API

El desarrollo de la API ha sido en HTML con Javascript y JQuery. Quizá hubiera sido mejor utilizar django para crear el entorno web. A nivel educativo ha sido útil por aprender otros lenguajes y las tecnologías que usa Javascript tan exigidas hoy en día en el mundo laboral, pero si tuviésemos que realizar un proyecto de estas características, lo que haríamos sería desarrollar el entorno web con django explotando así todas sus características.

### 9.5. Django

Django te permite un manejo sencillo de la base de datos, de hecho podríamos decir que es una de las herramientas más potentes y que la hemos exprimido a la mitad debido a que hemos decidido descentralizar la aplicación. Porque como hemos dicho podríamos haber desarrollado la interfaz gráfica de la API como otro proyecto

de django usando sus plugins para desarrollo web que son bastante potentes, pero pensamos en que no estuviesen en la misma cpu la interfaz y la API en sí. La herramienta es bastante potente, pero la mayoría de esta potencia reside en la capacidad de instalar plugins y aprender el manejo de estos para crear algo más profesional ya que por si sola carece de algunas características bastantes útiles a la hora de investigar. Volveríamos a usar Django para un proyecto de estas características



## **Parte VI**

# **Trabajos futuros**



## 9.6. Trabajos futuros

### 9.6.1. Extracción

#### **Extracción de contenidos desde otros gestores de contenidos**

Hasta ahora todo el proceso de extracción se ha realizado sobre municipios cuyas web se han generado haciendo uso del gestor de contenidos OpenCMS, una ampliación del proyecto puede ser la extracción de otros gestores de contenidos que usan los municipios de Andalucía como Joomla!, TYPO3 o WordPress.

#### **Extracción completa de los datos disponibles en el Instituto de estadística y cartografía de Andalucía**

Extracción de todos los datos que ofrece el Instituto de estadística y cartografía de Andalucía, actualmente solo obtenemos los datos relativos al número de contratos de trabajo creados en el municipio.

#### **Extracción de otros datos de la web de un municipio**

En la web de un municipio hay mucha más información a parte de las noticias, una posible ampliación de Semandal podría ser la extracción de esta información. En ella se incluyen enlaces a trámites que se pueden realizar a través de Internet, calendarios de eventos, etc.

#### **Extracción de elementos adjuntos a las noticias**

Junto con el texto de una noticia muchas veces se pueden encontrar elementos que lo acompañan, estos elementos son: imágenes, documentos (.pdf o .doc), enlaces a otros lugares de Internet, etc. En futuras versiones de la aplicación podrían ofrecerse enlaces a estos elementos o en el caso de las imágenes mostrarse junto al texto de la noticia.

#### **Búsqueda y creación de extractores para nuevas webs que usen OpenCMS**

Con el paso del tiempo es posible que uno o varios municipios hayan modificado sus páginas web. Una tarea futura sería buscar si existen nuevos pueblos que usen

OpenCMS y extraer de ellos las noticias creando un nuevo script, si fuese necesario.

### **9.6.2. Clasificación**

#### **Utilización de nuevos algoritmos de clasificación**

El uso de otros algoritmos aplicables a la clasificación de texto como las máquinas de vector soporte (SVM), junto con los algoritmos que usamos actualmente puede ayudar a, en general, obtener mejores resultados.

#### **Actualización de los modelos de clasificación**

Una tarea de especial interés es la actualización de nuestros modelos de clasificación teniendo en cuenta las nuevas noticias de las que se disponen y las correcciones de las categorías que realicen los usuarios de la aplicación. De esta forma de depura la base de conocimiento y se podría mejorar la tasa de acierto de los modelos de clasificación.

### **9.6.3. Publicación**

#### **Migrar a estructura json por defecto django**

Las estructuras devueltas en JSON son creadas a mano por comodidad y simplicidad. Pero para realizar un uso correcto de django deberíamos estudiar los plugins y métodos para convertir a json estructuras devueltas con django para realizar una API más profesional.

#### **Encriptación de contraseñas**

A la hora de realizar el logueo, la contraseña se manda en texto plano. Un trabajo futuro sería poder enviar datos del usuario encriptados, para poder ofrecer al usuario la seguridad de sus datos a la hora de utilizar la APP.

#### **Implementación de OUATH2**

Existe un plugin para django que realiza la autenticación a través de OUATH2. Aunque en otro trabajo futuro se enfoca la implementación de facebook sdk, tam-

bien sería correcto tener un sistema de logueo propio para aquellos usuarios que no dispongan de cuenta en facebook.

### **Agregar funcionalidad social**

Aunque ya implementada, quitamos la funcionalidad de amigos porque no era un objetivo desde un primer momento del proyecto. Un trabajo futuro sería activar esta funcionalidad y testear su comportamiento. También debemos cambiar la apariencia de dichas pantallas, ya bien sea cambiando su estructura y/o agregand otros elementos visuales como bandera de pueblos a los que siguen los distintos usuarios en su perfil.

### **Implementar Facebook SDK**

Algo que expandiría y mejoraría la usabilidad de nuestra aplicación sería integrar Facebook con ésta. Así mejoraríamos la funcionalidad social descrita anteriormente. También nos ayudaría en la gestión social ya que podría ser más sencillo usar el sdk de facebook que implementar la agregación de amigos y el almacenamiento de estos.

### **Agregar geolocalización embebida en la aplicación android**

Otra mejora, en lugar de poner un enlace a google maps, sería insertarla en la misma pantalla que nos ofrece los datos de los pueblos. Para ello habría que investigar en la utilización más a fondo de la tecnología y realizar un estudio de compatibilidades de dicha función con las distintas versiones de android a ver si quitaría usuarios potenciales.

### **Compartir**

Para la funcionalidad social de la aplicación y para que ésta funcione entre los usuarios debemos implementar un botón para compartir enlaces de las noticias.

#### **9.6.4. Infraestructura**

##### **Migrar a un sistema amazon web service**

Unas de las mejoras más útiles sería migrar a un sistema de amazon web services ya que son los que mejor rendimiento han ofrecido. Este sistema también nos ofrece

balanceadores de cargas y distintas copias de seguridad. La pega de este sistema, es que es un sistema de pago por tanto, deberíamos migrar a esta estructura una vez que tuviésemos un número elevado de usuarios y por tanto se incrementen las ofertas por publicidad.

## **Parte VII**

### **Apéndices**



# Base de datos

---

## A.1. Noticias y pueblos

(Ver figura 8.4) Las tablas que almacenan información sobre municipios y noticias son las siguientes:

### A.1.1. Noticias

Tabla que contiene información sobre las noticias. Guardamos titular, resumen, cuerpo de la noticia, municipio, URL a la noticia y fecha.

### A.1.2. Pueblo

Tabla que contiene información sobre municipios. Guardamos nombre del pueblo, provincia en la que se encuentra, habitantes, URL a la web del municipio, localización GPS, deuda, superficie y densidad de población.

### A.1.3. Pueblo\_InfoExtra

Esta tabla almacena información extraída desde el instituto de estadística de la Junta de Andalucía. La información extraída incluye información relativa al conjunto de contratos creados en cada municipio en el año 2013.

#### A.1.4. vecinos

Esta tabla almacena parejas de pueblos vecinos. En la tabla para cada pueblo se almacenan los 5 pueblos más cercanos a este.

#### A.1.5. Provincia

Esta tabla incluye datos básicos de las 52 provincias españolas.

#### A.1.6. log\_update

Tabla donde almacenamos la fecha de ejecución de las actualizaciones de noticias.

#### A.1.7. log\_update\_web

En esta tabla se almacena para cada pueblo, la última vez que se buscaron noticias nuevas en su web y el resultado, satisfactorio o no, de dicha búsqueda.

### A.2. Clasificación y categorías

(Ver figura 8.3) Las tablas que tienen relación con el sistema de clasificación y las categorías son las siguientes:

#### A.2.1. Categorías

Esta tabla contiene las etiquetas que podemos encontrar en las distintas páginas de municipios que exploramos y con que categorías de las establecidas en 2.0.7 se corresponde.

#### A.2.2. Categorías Semandal

Tabla que contiene las categorías definidas en 2.0.7 y las categorías sugeridas por los usuarios

### A.2.3. Reglas Ontología

Tabla que contiene el grafo de la ontología creado en la figura 2.1, cada registro guarda una arista del grafo.

### A.2.4. Classify

Tabla donde almacenamos las sugerencias de etiquetas que hace el usuario sobre una noticia y los resultados de la fase de clasificación.

### A.2.5. NC

Tabla donde se almacena la categoría o categorías asignadas a una noticia.

## A.3. Usuarios

(Ver figura 8.5) Las tablas que contienen información sobre los usuarios y las acciones que estos pueden realizar son las siguientes:

### A.3.1. Usuario

Esta tabla almacena información sobre el usuario. Se guarda un pueblo principal, una dirección de correo electrónico, una contraseña, un nombre y un nombre de usuario.

### A.3.2. SigP

Esta tabla almacena pueblos que sigue el usuario, además de su pueblo principal almacenado en la tabla Usuario.

### A.3.3. Votaciones

Esta tabla almacena información sobre las correcciones que hacen los usuarios sobre las categorías asignadas a las noticias. Si hay suficientes votos negativos la categoría se considera mal asignada y se elimina de la noticia.

### A.3.4. T\_Liked

Esta tabla guarda información sobre las noticias que los usuarios marcan como favoritas.

### A.3.5. Comentarios

Esta tabla almacena los comentarios que los usuarios hacen sobre las noticias.

### A.3.6. Denuncias\_C

Esta tabla almacena información sobre los comentarios que los usuarios marcan como negativos, si hay suficientes votos negativos se elimina el comentario.

### A.3.7. NVistas

Esta tabla almacena información sobre las noticias vistas por cada usuario.

## A.4. Control

(Ver figura 8.2)

### A.4.1. Llamadas

Esta tabla almacena información estadística sobre las llamadas a la API.

### A.4.2. Versions

Esta tabla lleva un control sobre las modificaciones hechas a distintas tablas para que desde una aplicación se pueda determinar si es necesaria la actualización de algún dato.

# Librerías y software utilizado

---

A continuación se muestran las librerías y software necesarios para hacer funcionar el proyecto Semandal.

## B.1. Librerías

### B.1.1. Python

#### **re**

Librería para resolver expresiones regulares. Esta librería se incluye en la instalación de Python.

#### **BeautifulSoup**

BeautifulSoup es una de las librerías más populares para trabajar con código HTML en Python. Se puede descargar desde la web de BeautifulSoup<sup>1</sup>.

#### **urllib2**

Librería para acceder a un web a través de una URL y obtener el código HTML de la página que se pide. Esta librería se incluye en la instalación de Python.

---

<sup>1</sup><http://www.crummy.com/software/BeautifulSoup/>

**xlrd**

Librería que permite leer ficheros de Excel (.xls) Se puede descargar de forma gratuita desde la web oficial de Python<sup>2</sup>.

**MySQLdb**

Librería que sirve de interfaz con una base de datos MySQL. Se puede descargar de forma gratuita desde la web oficial de Python<sup>3</sup>.

**subprocess**

Librería para generar y lanzar nuevos procesos recuperando la salida de estos. Esta librería se incluye en la instalación de Python.

**datetime**

Librería para trabajar con fechas. Esta librería se incluye en la instalación de Python.

**sys**

Librería para interactuar con el interprete de Python. Esta librería se incluye en la instalación de Python.

**os**

Librería para interactuar con el sistema operativo. Esta librería se incluye en la instalación de Python.

---

<sup>2</sup><https://pypi.python.org/pypi/xlrd>

<sup>3</sup><https://pypi.python.org/pypi/MySQL-python>

### B.1.2. Java

#### JESS

Motor de reglas y framework de razonamiento escrito en Java. Se puede integrar en aplicaciones escritas en Java y Android. Se puede obtener una licencia gratuita con fines educativos en la web de JESS<sup>4</sup>.

## B.2. Software

### B.2.1. Django

Framework de desarrollo escrito en Python, utilizado para la creación y mantenimiento de la API, se puede descargar de forma gratuita desde la web de Django<sup>5</sup>.

#### Django-extensions

Django-extension es un conjunto de comandos y extras que se añaden a la funcionalidad que ofrece Django, en nuestro caso la hemos utilizado para realizar un diagrama de base de datos. Se pueden descargar desde la web oficial de Python<sup>6</sup>.

### B.2.2. Eclipse

Entorno de desarrollo para Java, utilizado para la obtención de reglas y la clasificación. Puede descargarse desde la web de Eclipse<sup>7</sup>.

### B.2.3. Eclipse (Android SDK)

Entorno de desarrollo para Java orientado al desarrollo en Android. Utilizado para el desarrollo de la aplicación en Android. También nos instala los emuladores necesarios para probar nuestro código en Android. Se puede descargar desde la web de Android<sup>8</sup>.

---

<sup>4</sup><http://herzberg.ca.sandia.gov/>

<sup>5</sup><https://www.djangoproject.com/download/>

<sup>6</sup><https://pypi.python.org/pypi/django-extensions/1.5.0>

<sup>7</sup><https://eclipse.org/downloads/>

<sup>8</sup><http://developer.android.com/sdk/index.html>

### B.2.4. Concept Explorer

Aplicación para trabajar con Análisis formal de conceptos. Se puede descargar gratuitamente desde la web de Concept Explorer<sup>9</sup>.

### B.2.5. WEKA

Suite de Inteligencia artificial y minería de datos utilizada para trabajar con algoritmos C4.5 y RIPPER. Se puede descargar de forma gratuita desde la web de WEKA<sup>10</sup>.

### B.2.6. Graphviz

Aplicación para generar grafos a partir del lenguaje dot, utilizada junto a las django-extensions para generar el diagrama de la base de datos. Se puede descargar desde la web de Graphviz<sup>11</sup>.

---

<sup>9</sup><http://conexp.sourceforge.net/>

<sup>10</sup><http://www.cs.waikato.ac.nz/ml/weka/>

<sup>11</sup><http://www.graphviz.org/Download..php>

# **Parte VIII**

# **Bibliografía**



---

---

## Bibliografía

---

- [1] William W. Cohen. Fast effective rule induction. *Machine Learning: Proceedings of the Twelfth International Conference*, 1995.
- [2] Charles L. Forgy. Rete: A fast algorithm for the many pattern/many object pattern match problem. *Artificial Intelligence*, 1982.
- [3] Eibe Frank and Ian H. Witten. Generating accurate rule sets without global optimization. *Machine Learning: Proceedings of the Fifteenth International Conference*, 1998.
- [4] Cèsar Ferri Ramírez José Hernández Orallo, M.José Ramírez Quintana. *Introducción a la Minería de Datos*, chapter 9. Editorial Pearson, Madrid, 2004.
- [5] S. B. Kotsiantis. Supervised machine learning: A review of classification techniques. *informatica* 31:249–268, 2007.
- [6] Oded Maimon Lior Rokach. *Data mining with decision trees*, chapter 3. World Scientific Publishing Co. Pte. Ltd, Singapore, 2008.
- [7] J. Ross Quinlan. *C4.5: Programs for machine learning*. Morgan Kaufman Publishers, San Mateo, California, 1993.
- [8] Peter Norvig Stuart Russell. *Artificial Intelligence: A Modern Approach*, chapter 18. Pearson Education, Inc, Upper Saddle River, New Jersey, 2009.
- [9] Rudolf Wille. Concept lattices and conceptual knowledge systems. *Computers Math. Applic.*, 23, 1992.

- [10] Rudolf Wille. Formal concept analysis as mathematical theory of concepts and concept hierarchies. *Formal Concept Analysis: Foundations and Applications*, 2005.