ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA
GRADO EN INGENIERÍA INFORMÁTICA
Mención en Sistemas de Información

SISTEMA DE RECOMENDACIÓN BASADO EN LA EXTRACCIÓN DE CONCEPTOS
RECOMMENDER SYSTEM BASED ON CONCEPT INFERENCE

Realizado por
CHRISTIAN CINTRANO LÓPEZ
Tutorizado por
MANUEL ENCISO GARCÍA-OLIVEROS
Co-tutorizado por
CARLOS ROSSI JIMÉNEZ
Tutor coordinador
MANUEL ENCISO GARCÍA-OLIVEROS
Departamento
LENGUAJES Y CIENCIAS DE LA COMPUTACIÓN

UNIVERSIDAD DE MÁLAGA
MÁLAGA, Julio de 2014

Fecha defensa:
El Secretario del Tribunal

Resumen: Este Trabajo Fin de Grado (TFG) tiene como objetivo la creación de un framework para su uso en sistemas de recomendación. Este TFG se ha realizado por dos personas en la modalidad de trabajo en equipo. El trabajo se dividió en dos partes, una realizada conjuntamente y la otra de manera individual. La parte conjunta tiene como objetivo construir un sistema que sea capaz de, a partir de comentarios y opiniones sobre *puntos de interés* (POIs) y haciendo uso de la herramienta de procesamiento de lenguaje natural AlchemyAPI, construir *contextos formales*. Éste es el eje principal de la teoría del *análisis formal de conceptos* (FCA) propuesta por Bernhard Ganter. Además será el punto de partida de la segunda parte (individual), que consistirá en aplicar otra parte de la teoría de FCA para obtener, a partir del *contexto*, el *retículo de conceptos* mediante la implementación en Java del algoritmo Titanic. Estos *conceptos* podrán ser usados para obtener grupos de usuarios y tendencias. El sistema se ha implementado como una aplicación web Java EE versión 6 y una API para trabajar con *contextos formales*. Para el desarrollo web se han empleado tecnologías actuales como Spring y jQuery. Este proyecto se presenta como un trabajo inicial en el que se expondrán, además del sistema construido, diversos problemas relacionados con los sistemas de recomendación y se propondrán líneas para futuros TFGs.

Palabras clave: Sistemas de recomendación, Descubrimiento de conocimiento, Procesamiento del lenguaje natural, Análisis formal de conceptos, Contexto formal

Abstract: This Degree Thesis (TFG) aims to create a framework for use in recommender systems. The TFG has been done by two people in the form of teamwork. The work was divided in two parts, one joint and other individually. The joint part aims to build a system that is able to, from and comments about *points of interest* (POIs) and using the tool of natural language processing AlchemyAPI, build *formal contexts*. This is the main axis of the theory of *formal concept analysis* (FCA) proposed by Bernhard Ganter. It will also be the starting point of the individually part. It will implement another part of the FCA theory to obtain the *concept lattice* from the *context*. This implementation will be carried out by implementing the Titanic algorithm in the programming language Java. These *concepts* may be used to obtain user groups and trends. The system has been implemented as a Java EE version 6 web application and an API to work with *formal contexts*. In development have been used current technologies such as Spring and jQuery. This project is presented as an initial work in which, several issues related to recommender systems and lines for future TFGs have been proposed.

Keywords: Recommender systems, Knowledge discovery, Natural language processing, Formal Concept Analysis, Formal Context

Índice

1. Introducción

Este es un Trabajo Fin de Grado desarrollado en la línea general Social CRM. Esta línea busca la creación de un sistema de recomendación, desde la extracción de la información de las fuentes de datos: comentarios, opiniones, reseñas, etc. hasta dar una recomendación al usuario. Principalmente se centra en recomendaciones para el sector turístico aunque las teorías presentadas son perfectamente aplicables a otras áreas de interés.

Al tratarse de un trabajo fin de grado de la modalidad de equipo se propuso un reparto de las tareas a realizar. Una primera parte ha sido desarrollada conjuntamente, y se distinguieron otras dos tareas a realizar por cada uno de los miembros del equipo de manera individual. Estas dos tareas individuales estaban bien diferenciadas pero relacionadas entre sí.

En la parte común del TFG se estudió como, desde unas opiniones sobre *puntos de interés* (POIs), extraer información sobre atributos o características de los mismos. Esta información debía de ser tratada y recogida en una tabla binaria en la que se relacionasen a los autores de los comentarios (objetos) con las características más relevantes (atributos). A esta tabla se le denomina *contexto formal* y será empleada en las partes específicas del TFG.

La parte específica del TFG presentada en esta memoria consistió en el descubrimiento de grupos de usuarios y de características sobre los POIs. Partiendo del *contexto formal* se aplicaron unos operadores de cierre definidos por Ganter [55] para obtener una serie de agrupaciones entre un conjunto de atributos y objetos de la tabla. A estas agrupaciones se les denomina *conceptos* y podrían ser interpretadas a posteriori por un experto para descubrir tendencias o grupos de los usuarios. Un ejemplo que ilustra esta teoría podría ser el siguiente.

Supongamos que tenemos como objetos un conjunto de comentarios de distintos usuarios sobre un determinado hotel y como atributos una serie de características del mismo como: calidad de la habitación, staff, comida, piscina, playa, etc. Un *concepto* podría ser el conjunto de usuarios en cuyos comentarios habla sobre la piscina y la playa. Entonces, un experto podría ver esta información y nombrar esta agrupación con un nombre "turista veraniego". De esta forma quedaría definido este grupo de usuarios mediante esas características. Además podría ser objetivo de futuras campañas de marketing.

Para la construcción de este sistema se ha realizado un desarrollo web aplicando la tecnología Java EE versión 6 haciendo uso de frameworks y librerías como Spring, jQuery, Bootstrap, etc. Además para la parte de minería de datos y obtención de atributos a partir de los textos se empleó la API de AlchemyAPI que funciona mediante un servicio web. La aplicación se presenta en términos de la teoría de FCA permitiendo trabajar con *contextos formales* y *conceptos* de manera sencilla. De esa forma un experto podría

fácilmente interpretar los datos obtenidos del sistema.

En las posteriores secciones se desarrollarán todos los aspectos relativos al TFG. En la sección 2 se expondrá la motivación que llevó a la elaboración de este trabajo. La sección 3 explica la metodología seguida durante todo el desarrollo del proyecto. A continuación, en la sección 4 describe cual es el sistema a construir, mientras que en la sección 5 se nombran los requisitos que debe de cumplir dicho sistema. Seguidamente, en la sección 6, se expondrá el estado del arte y los fundamentos teóricos necesarios para la construcción del sistema. En la sección 7 se analizarán uno a uno los requisitos enunciados en la sección 5. La sección 8 describe el entorno tecnológico del sistema y la sección 9 los aspectos relativos al desarrollo de la aplicación. En la sección 10 se muestra una serie de capturas de pantalla de la aplicación final. A continuación se exponen y analizan una serie de experimentos desarrollados a partir del sistema construido (Sección 11). También se presentan unas conclusiones generales de todo el TFG (Sección 12). Además de estas secciones, en la memoria se incluye la bibliografía y una serie de anexos. Estos anexos se componen de: Un desglose a alto nivel del tiempo empleado en cada tarea del TFG, una lista de los algoritmos descritos en la memoria, un conjunto de programas, APIs y plugins para trabajar sobre aspectos de la teoría de FCA, y los *contextos formales* empleados en el desarrollo del TFG.

2. Motivación del TFG

La finalidad del TFG es la creación de un sistema de recomendación. Hoy en día, este tipo de tecnologías son muy importantes para las empresas. Además el sector en el que se ha enfocado es en el sector turístico tan importante en la provincia de Málaga. Tal importancia han adquirido estos sistemas que existen numerosos grupos de investigación trabajando en estos temas. Un ejemplo puede ser el grupo SICUMA (Sistemas de Información Cooperativos de la Universidad de Málaga)[50].

Particularmente para el autor de esta memoria ha supuesto el aprendizaje de una serie de tecnologías actuales y muy útiles para la formación. Además de poder profundizar en aspectos sobre los CRM que no se abarcan durante el desarrollo del grado.

3. Metodología

Se siguió el modelo de desarrollo ágil Scrum [45] por los dos miembros del equipo. A continuación se detallan algunas particularidades:

- Se definieron *sprints* de una semana de duración. De esta forma se conseguía un seguimiento más exhaustivo del proyecto.
- Al final de cada *sprint* se realizaba una reunión con el tutor, que haría las veces tanto de *Scrum Master* como de cliente.
- En esa reunión se realizaba el *Sprint Retrospective* con los productos, pruebas y conclusiones obtenidos del anterior *sprint*, y el *Sprint Planning* en el que se decidían que *historias de usuario* entrarían en la siguiente iteración.

Generalmente se dividió cada *sprint* en cuatro fases: recolección de información y estudio, búsqueda de herramientas, construcción y, verificación y pruebas. En las primeras iteraciones se dedicó un mayor esfuerzo en tareas de estudio, mientras que en las finales se enfocaron más las partes de construcción y, verificación y pruebas. A continuación se explica brevemente cada una de estas fases.

- Recolección de información y estudio: Para cada *historia de usuario* del *sprint* se realizó un estudio previo del estado del arte. Para ello se revisaron artículos relacionados con la materia y páginas web dedicadas al tema en concreto.
- Búsqueda de herramientas: Una vez afianzado el conocimiento se buscaron herramientas o frameworks que pudieran solucionar el problema o parte de él. En este TFG se intentó dar prioridad al uso de herramientas ya creadas antes que al desarrollo propio, principalmente para el procesamiento del lenguaje natural.
- Construcción: Se construyeron los códigos necesarios para probar los frameworks y para el desarrollo del sistema.
- Verificación y pruebas: Todo el código creado se testeó y comprobó que los datos generados fueran correctos, empleando para ello una serie de casos de prueba.

Además para el desarrollo de la aplicación se siguió un modelo de procesos incremental. Estas se explicarán con mayor detalle en la sección 9.4.

4. Objetivos

Este TFG posee dos objetivos bien definidos para cada una de las partes (común e individual):

- El principal objetivo de la parte común del TFG es la construcción de un sistema que, a partir de comentarios sobre POIs sea capaz de crear un *contexto formal*.
- La parte individual persigue construir un sistema que, a partir de un *contexto formal* sea capaz de extraer el *retículo de conceptos*. Además se desea que sea integrable con el sistema general.

5. Requisitos

Para la creación del sistema se tuvieron en cuenta una serie de requisitos. A continuación se expondrán los requisitos relativos a la parte común del TFG:

RNF-01	**Desarrollo como aplicación web**
Tipo de requisito	No Funcional
Dependencias	
Descripción	La aplicación debe de seguir una arquitectura web.
Importancia	Alta
Prioridad	Baja
Asignado a:	Christian Cintrano López y David Barrientos Brenes

RNF-02	**Interfaz sencilla**
Tipo de requisito	No Funcional
Dependencias	
Descripción	La aplicación debe tener una UI minimalista y sencilla de usar.
Importancia	Baja
Prioridad	Baja
Asignado a:	Christian Cintrano López y David Barrientos Brenes

RF-01	**Obtener el conjunto de atributos para el contexto formal**
Tipo de requisito	Funcional
Dependencias	
Descripción	Extraer los atributos o *keywords* de los comentarios.
Importancia	Alta
Prioridad	Alta
Asignado a:	Christian Cintrano López y David Barrientos Brenes

RF-02	**Procesar los comentarios para rellenar la tabla que representa al *contexto formal***
Tipo de requisito	Funcional
Dependencias	▪ RF-01
Descripción	Necesidad de un procesamiento automatizado de las entradas del sistema (comentarios) para incluir la información necesaria dentro de la tabla del *contexto formal*.
Importancia	Alta
Prioridad	Alta
Asignado a:	Christian Cintrano López y David Barrientos Brenes

RF-03	**Almacenamiento del *contexto formal* con un formato concreto**
Tipo de requisito	Funcional
Dependencias	▪ RF-02
Descripción	Generar un fichero con la información del *contexto formal*.
Importancia	Baja
Prioridad	Baja
Asignado a:	Christian Cintrano López y David Barrientos Brenes

Y estos corresponden a la parte individual:

RF-04	**Empleo de técnicas para obtener todos los conceptos a partir de una tabla de FCA**
Tipo de requisito	Funcional
Dependencias	▪ RF-01 ▪ RF-02
Descripción	A partir de un *contexto formal* obtener todos los conceptos posibles.
Importancia	Alta
Prioridad	Alta
Asignado a:	Christian Cintrano López

RF-05	**Representación de los conceptos**
Tipo de requisito	Funcional
Dependencias	▪ RF-04
Descripción	Visualizar el *retículo de conceptos* mediante alguna herramienta, siguiendo la representación jerárquica de retículos.
Importancia	Media
Prioridad	Media
Asignado a:	Christian Cintrano López

RF-06	**Integración con la parte común**
Tipo de requisito	Funcional
Dependencias	
Descripción	Sería deseable que los conceptos pudieran extraerse automáticamente a partir del *contexto formal* generado a partir de los comentarios.
Importancia	Alta
Prioridad	Baja
Asignado a:	Christian Cintrano López

En la sección 7 de esta memoria se analizarán con un mayor nivel de detalle estos requisitos.

6. Fundamentos y estado del arte

En esta sección se describe el estado del arte relativo a los sistemas de recomendación así como los fundamentos teóricos relativos a la teoría del *análisis formal de conceptos*.

6.1. Introducción

Existe una gran cantidad de bibliografía sobre los sistemas de recomendación. Esto es debido al gran auge del comercio electrónico, que emplea estos sistemas, y a las redes sociales, fuente para la extracción de información. Esta información está formada básicamente por opiniones sobre las empresas y sus productos. Estas críticas son muy valiosas para los directivos ya que les permite conocer los gustos de sus clientes y así poder enfocar mejor sus líneas de negocio. Pero esta información no es nada si no es tratada y transformada en conocimiento útil. En esta línea se reúnen áreas tan relevantes hoy en día como la minería de datos, el análisis de sentimientos, big data, etc. Todas estas técnicas permiten pasar de los comentarios hasta información útil para el directivo. Estos sistemas también son muy beneficiosos para los usuarios porque reciben información más precisa y por lo tanto ofreciéndole una mayor satisfacción.

6.2. Big data en sistemas de recomendación

Como ya se ha mencionado las redes sociales son una gran fuente de información, principalmente sobre opiniones. Cada vez más y más personas hacen uso de redes sociales como Facebook y Twitter, entre otras, para expresar sus gustos y opiniones de manera asidua. Esto ha provocado que las empresas que quieran sacar partido de esta información tengan que recabar el mayor número de opiniones posibles. Pero la masividad con la que se hace uso de estas redes implica que haya que tratar grandes cantidades de datos. Para solucionar estos problemas nació el conjunto de técnicas y tecnologías conocidas como big data. De una manera sencilla, el big data no es más que el tratamiento de grandes cantidades de información. Uno de los objetivos del big data es realizar operaciones sobre estos conjuntos de datos de la forma más eficiente posible. La base del big data es la técnica del MapReduce que permite computación paralela sobre grandes colecciones de datos.

Big data es una tecnología aplicable a casi cualquier área de los sistemas de información. Debido a ello es un campo muy extenso con una gran cantidad de teorías, herramientas y tecnologías. Yendo a lo esencial se podrían dividir las tecnologías relacionadas con big data en tres líneas distintas:

Frameworks de computación paralela La principal ventaja del big data consiste en el MapReduce. Sin embargo para obtener el máximo beneficio de ello es necesario

realizar computación paralela con un gran número de equipos. Existen muchos frameworks en el mercado para esta tarea. Entre todo este abanico de ofertas, Hadoop [26] ha tenido una especial buena acogida, y gracias a ella se han desarrollado muchos frameworks sobre él. Hadoop es un framework de software libre para aplicaciones distribuidas y big data.

Herramientas de recolección de información de redes sociales Un aspecto importante del big data es que necesita de grandes cantidades de información, fundamentalmente de las redes sociales. Por este motivo existen múltiples herramientas para poder obtener, de manera sencilla, esta información. Por ejemplo, Apache Flume [5] es un framework para Apache que nos permite establecer un canal por el cual nuestra aplicación tenga acceso a tweets de Twitter de manera automática. Además es perfectamente integrable con Hadoop [3].

Almacenamiento de la información El problema de las tecnologías de big data es que requieren de un enorme almacenamiento. Además los sistemas gestores de bases de datos relacionales (SGBD relacionales) no son eficientes a la hora de recuperar o agrupar grandes cantidades de información. Para solucionar esto se crearon las bases de datos no relacionales y los almacenes de datos. Estas bases de datos se basan en el concepto de cubo de datos y permiten realizar más eficientemente la instrucción *select*, así como las agrupaciones o filtrado de datos. Las principales empresas de bases de datos, como Oracle y su Data Warehouse, ofrecen bases de datos no relacionales así como múltiples herramientas para interactuar con ellas.

6.3. Opiniones como eje principal de los sistemas de recomendación

Todos estos sistemas de recomendación basados en redes sociales, y muchos otros más, tienen como base las opiniones de los usuarios. Los autores suelen definir una opinión como una quíntupla $(o_j, f_{jk}, oo_{ijkt}, h_i, t_t)$ [59]. Siendo o_j el objeto sobre el que se habla (p. e. un Smartphone), f_{jk} la característica o atributo del objeto que se está valorando (p. e. duración de la batería o tamaño de la memoria), h_i el *holder* o autor de la opinión, t_t la fecha en el que se realizó el comentario y oo_{ijkt} que es la valoración positiva, negativa o neutral de la opinión sobre la característica del objeto realizada por el holder en ese momento. Dependiendo del autor y las características del sistema a desarrollar la quíntupla puede variar, eliminando el t_t, empleando una taxonomía de atributos, o el uso de sinónimos. Estos dos últimos casos serán descritos a continuación.

Taxonomía de atributos La mayoría de los objetos poseen una gran cantidad de características. Además sus componentes también poseen una serie de atributos. Un

ejemplo puede ser el caso de un Smartphone que puede ser bonito o resistente (atributos), mientras que si nos referimos a la pantalla puede ser grande o pequeña y que tenga más o menos resolución. Estos siguen siendo atributos del objeto Smartphone, pero son específicos de un conjunto de sus componentes. Esto nos permite definir un objeto como $O = (T, A)$ [58], siento T una taxonomía de partes con la relación *parte de*; y A un conjunto de atributos. Esto permite crear una jerarquía de tributos y componentes dentro de un mismo objeto. Con esta técnica se consigue una gestión de la información más eficiente. Además hay que tener en cuenta que un mismo atributo puede pertenecer a varios objetos o partes y valorarse de manera distinta para cada uno de ellos. Otras de las aplicaciones de las ontologías de este tipo son en el campo del análisis de sentimientos en textos [60]. Ejemplos de aplicaciones que pueden gestionar esta información puede ser FCAView Tab Plugin [20] u OntoComp [41] aplicaciones diseñadas para Protégé.

Sinónimos Este es uno de los principales problemas de la extracción de opiniones y análisis de sentimientos. Los sinónimos son de vital importancia a la hora de interpretar la opinión del usuario. Existen una gran cantidad de artículos dedicados a solucionar este problema, sin embargo, no existen técnicas realmente eficientes para ello. La mayoría de las reseñas hablan sobre el uso de diccionarios de términos o tesauros para agrupar los conjuntos de sinónimos [61]. Una buena característica de éstos es que, al ser las aplicaciones de este tipo enfocadas a algún sector concreto, el vocabulario es más reducido y por lo tanto se mejora mucho el tiempo de respuesta [56] y la calidad de los sinónimos obtenidos.

6.4. Sistemas de recomendación en el sector turístico

Toda esta teoría se presenta de forma general sin entrar en ningún campo de estudio. Sin embargo, uno de los sectores en los que posee especial relevancia es en el sector turístico [57], que es el tratado en este TFG. En las opiniones sobre destinos turísticos (restaurantes, hoteles, parques, museos, etc.) existen dos elementos especialmente relevantes: el turista[1] y los Puntos de Interés, usualmente abreviado como POIs [57]. En este tipo de sistemas obtiene una especial relevancia las características de estos usuarios. Estas pueden ser intrínsecas como lugar de procedencia, gustos, etc. o aspectos extrínsecos, por ejemplo el modo de realizar la visita turística (solo, en familia, con amigos, etc.) o las condiciones climáticas del día en cuestión.

Este último tipo de características son muy interesantes para los sistemas de recomendación sensibles al contexto. Estos sistemas emplean factores externos para dar la recomendación, por ejemplo recomendar ferias para salidas nocturnas o proponer ir a centros comerciales en días lluviosos.

Para la mayoría de entornos, la recomendación se realiza a un individuo concreto, sin

[1]Hace referencia a cualquier usuario final del sistema

embargo, el turismo es una actividad que comúnmente se realiza en grupo. Para mejorar la experiencia de estas agrupaciones existen varias técnicas llamadas filtrado en grupo. Ellas intentan obtener una recomendación que pueda satisfacer al grupo en general más que a algunos individuos en concreto.

Por otro lado, los POIs pueden ser restaurantes, hoteles, paseos, parques, centros de ocio, etc. en resumen cualquier lugar que pueda ser objeto de interés para el turista. Estos forman el principal objeto de estudio en la recomendación y son la base para construir los sistemas.

6.5. Filtrado de información

Existen una gran cantidad de algoritmos y técnicas distintas en el campo de la recomendación. Gracias a ellas se consigue el filtrado de la información menos relevante para el usuario. Estos sistemas de filtrado, aunque suelen ser muy distintos entre sí, se pueden catalogar en:

- Colaborativos: Se basan en que los usuarios dan una serie de calificaciones a un conjunto de objetos. Estas calificaciones se asignan mediante un rango de valores y describe de manera subjetiva como de "bueno" es el objeto. De esta forma el sistema recomienda al usuario una serie de objetos que fueron valorados positivamente por otros usuarios con calificaciones similares a las del recomendado.

- Basados en contenido: Similar al anterior. Recomiendan objetos similares que los que se valoraron positivamente en el pasado.

- Demográficos: La recomendación se enfoca a conjuntos de usuarios, agrupándolos por geografía, sexo, edad, etc.

- Basados en conocimiento: Tienen información sobre como un objeto satisface a un usuario, y establece una relación entre sus necesidades y la recomendación.

6.6. Análisis Formal de Conceptos

En la línea de los sistemas de recomendación existen diversas teorías que persiguen la construcción de un modelo que permita obtener conocimiento a partir de opiniones sobre POIs o sus características. Una de estas teorías que ha dado muy buenos resultados en este campo es la teoría matemática del Análisis Formal de Conceptos (Formal Concept Analysis, FCA en adelante).

Esta teoría modela el termino de *concepto* en términos de la teoría de retículos. Fue ideada por Bernhard Ganter [55] y ha sido muy útil en los sistemas de recomendación por sus buenos resultados y simplicidad. FCA parte de lo que Ganter denomina un *contexto*

formal $\mathbb{K}$ que se define como una tupla (G, M, I). G es un conjunto de objetos de estudio, M un conjunto de atributos e I una relación binaria entre G y M, $I \subseteq G \times M$. En el ámbito turístico los objetos representan a los usuarios que dan las opiniones y los atributos a los POIs o sus características.

Un ejemplo de representación en forma de tabla de un *contexto formal* puede ser el siguiente:

Hotel	habitación	servicio	vistas
Usuario 1	×		
Usuario 2		×	
Usuario 3		×	×

Además del *contexto formal*, Ganter también definió un par de operadores de cierre: Dado un $A \subseteq G$:

$$A' := \{m \in M | \forall g \in A : (g, m) \in I\}$$

Y dado un $B \subseteq M$:

$$B' := \{g \in G | \forall m \in B : (g, m) \in I\}$$

Y también definía las siguientes propiedades [55, 52]:

Proposición *Dados los conjuntos* $A, A_1, A_2 \subseteq G$ *tenemos que:*

1. $A_1 \subseteq A_2 \Rightarrow A_2' \subseteq A_1'$,
2. $A \subseteq A''$,
3. $A = A'''$

Dualmente, para los conjuntos $B, B_1, B_2 \subseteq M$ *tenemos que:*

1. $B_1 \subseteq B_2 \Rightarrow B_2' \subseteq B_1'$,
2. $B \subseteq B''$,
3. $B = B'''$

Los operadores constituyen una relación entre el conjunto potencia de los conjuntos G y M. A estas relaciones se les denominan **Conexiones de Galois** y son la base para la definición formal de *concepto*.

Definición *Un concepto de un Contexto Formal* (G, M, I) *es un par* (A, B) *tal que:*

$$A \subseteq \text{G, B} \subseteq \text{M, } A' = B \text{, y } A = B'$$

Otra definición puede ser la siguiente:

$$A \subseteq G \text{ y } B \subseteq M \text{ siendo máximo } A \times B \subseteq I$$

Este término posee un gran potencial en el ámbito turístico. Si los objetos G representan a usuarios, los atributos M características de POIs, e I que el usuario $g \in G$ comentó la característica $m \in M$; cada *concepto* representará a un grupo de usuarios con los mismos intereses u opiniones. Esta es la base de las técnicas de recomendación tales como el *filtrado colaborativo*. Sin embargo tiene un interés extra, ya que estas agrupaciones podrían servir para descubrir tendencias en los gustos de los usuarios.

El conjunto de todos los conceptos $\underline{\mathfrak{B}}(G, M, I)$ posee una estructura de retículo con la relación de orden $\subseteq$. Esto permite generar una estructura jerárquica de conceptos.

Ganter no solo propuso la teoría del FCA, sino que además propuso un algoritmo para obtener el retículo de conceptos $\mathfrak{B}(G, M, I)$. Este algoritmo se llama NextClosure [55]. A continuación se describe una de las implementaciones de este algoritmo [54]:

Algorithm 1 All Closures

input : Un conjunto M
output: Todos los cierres de M
begin
  First_Closure;
  repeat
    Output A;
    Next_Closure;
  until *not success*;

Algorithm 2 First Closures

input : Un conjunto M
output: El cierre A del conjunto vacío.
begin
  return $A \leftarrow \emptyset''$;

Algorithm 3 Next Closures

input : Un conjunto M, una lista de conjuntos de objetos A, una lista de conjuntos de atributos B, y un índice r. Inicialmente $r = 0 = i_0$, $A_0 = G$, $B_0 = G'$

output: El cierre (A_r, B_r)

```
begin
    foreach i = m, m − 1, ..., 1 do
        if m_i ∉ B_r then
            while i < i_r do
                r ← r − 1;
            i_{r+1} ← i;
            A_{r+1} ← A_r ∩ {m_i};
            upsets ← true;
            foreach j = 1, 2, ..., i − 1 do
                if {m_j}' ⊇ A_{r+1} then upsets ← false;
                exit for; ;
            if upsets = true then
                B_{r+1} ← B_r ∪ {m_i};
                foreach j = i + 1, i + 2, ..., m do
                    if j ∉ B_{r+1} then
                        if {m_j}' ⊆ A_{r+1} then
                            B_{r+1} ← B_{r+1} ∪ {m_j};
                r ← r + 1;
            return
        i − −;
    Stop:;
```

Este algoritmo muestra de una forma sencilla como obtener todos los *conceptos* de un *contexto formal*. Los mayores problemas de este algoritmo son:

- Necesita valores binarios en el Contexto Formal para funcionar. Esto limita el poder emplear otros algoritmos sobre la tabla para obtener otro tipo de información, p. e. obtener las implicaciones entre los atributos.
- La eficiencia es muy baja debido a la gran cantidad de operaciones de cierre que hay que realizar. Esto provoca que para grandes cantidades de datos sea poco útil.

El problema de la eficiencia ha llevado a muchos investigadores a tratar de mejorar el algoritmo. Una de esas mejoras es el algoritmo TITANIC [63].

Este algoritmo fue propuesto por Gerd Stumme, Rafik Taouil, Yves Bastide, Nicolas Pasquier y Lotfi Lakhal. Se basa en la obtención de una función de peso que reduce en gran medida el número de cierres a realizar. TITANIC no reduce la complejidad del problema pero si lo vuelve más eficiente. A continuación se presenta una implementación del algoritmo:

Algorithm 4 Titanic

input : Un conjunto M
output: Todos los cierres del conjunto M
begin
  WEIGHT ($\{\emptyset\}$); $\mathscr{K}_0 \leftarrow \{\emptyset\}$;
  $k \leftarrow 1$;
  forall the $m \in M$ **do**
    $\{m\}.ps \leftarrow \emptyset.s$;
  $\mathscr{C} \leftarrow \{\{m\} | m \in M\}$;
  loop begin
    WEIGHT ($\mathscr{C}$);
    forall the $X \in \mathscr{K}_{k-1}$ **do**
      X.closure $\leftarrow$ CLOSURE(X);
    $\mathscr{K}_k \leftarrow \{X \in \mathscr{C} | X.s \neq X.ps\}$;
    if $\mathscr{K}_k = \emptyset$ **then**
      exit loop;
    k++;
    $\mathscr{C} \leftarrow$ TITANIC-GEN($\mathscr{K}_{k-1}$);
  end loop;
  return $\bigcup_{i=0}^{k-1} \{X.\text{closure} | X \in \mathscr{K}_i\}$;

Algorithm 5 Titanic-Gen

input : $\mathscr{K}_{k-1}$, el conjunto de de claves $(k-1)$-sets K con sus pesos $K.s$
output: $\mathscr{C}$, el conjunto de candidatos k-sets C con los valores $C.ps := \min\{s(C \backslash \{m\}) | m \in C\}$
begin
  $\mathscr{C} \leftarrow \{\{m_1 < m_2 < ... < m_k\} | \{m_1, ..., m_{k-2}, m_{k-1}\}, \{m_1, ..., m_{k-2}, m_k\} \in \mathscr{K}_{k-1}\}$;
  forall the $X \in \mathscr{C}$ **do**
    forall the *(k-1)-subconjuntos* S *de* X **do**
      if $S \notin \mathscr{K}_{k-1}$ **then**
        $\mathscr{C} \leftarrow \mathscr{C} \setminus \{X\}$;
        exit forall;
      $X.ps \leftarrow \min(X.ps, S.s)$;
  return $\mathscr{C}$;

Algorithm 6 Closure

input : X, siendo $X \in \mathscr{K}_{k-1}$
output: El cierre de X
begin
  $Y \leftarrow X$; **forall the** $m \in X$ **do**
    $Y \leftarrow Y \cup (X \backslash \{m\})$.closure;
  forall the $m \in M \backslash Y$ **do**
    if $X \cup \{m\} \in \mathscr{C}$ **then**
      $s \leftarrow (X \cup \{m\}).s$
    else
      $s \leftarrow \min\{K.s | K \in \mathscr{K}, K \subseteq X \cup \{m\}\}$
    if $s = X.s$ **then**
      $Y \leftarrow Y \cup \{m\}$
  return Y;

Algorithm 7 Weigh

input : Un conjunto $\mathscr{X}$ de conjuntos de atributos
begin
  forall the $X \in \mathscr{X}$ **do**
    $X.s \leftarrow 0$;
  forall the $g \in G$ **do**
    forall the $X \in$ *subconjuntos(g',*$\mathscr{X}$*)* **do**
      $X.s$++;
  forall the $X \in \mathscr{X}$ **do**
    $X.s \leftarrow \frac{X.s}{|G|}$;

La función subconjuntos(Y,$\mathscr{X}$) devuelve para $Y \subseteq M$ y $\mathscr{X} \subseteq \mathfrak{P}(M)$, todos los $X \in \mathscr{X}$ con $Y \subseteq X$.

La mayoría de los algoritmos que trabajan con conceptos parten de un *contexto formal* binario, como son el caso de NextClosure y Titanic.

La construcción de un sistema capaz de obtener todo el *retículo de conceptos* de un *contexto formal* será la parte específica del TFG desarrollado.

Otro aspecto destacable de la teoría del FCA es la posibilidad de extraer implicaciones entre los atributos. Estas implicaciones son una especie de dependencia funcional entre los atributos, de manera que cada vez que un atributo a_1 se dé (en términos binarios se diría que en esa fila hay un "1") también se dará el atributo a_2. De esta forma se podría definir la implicación $a_1 \Rightarrow a_2$. En las implicaciones pueden intervenir más de un par de elementos, $\{a_1, a_2, a_3\} \Rightarrow \{a_4, a_5\}$.

Estas dependencias son muy importantes, ya que uno de los mayores problemas de los sistemas de recomendación consiste en que para poder dar una propuesta de recomen-

dación con un cierto grado de exactitud se requiere de mucha información por parte del usuario. Gracias a las implicaciones es posible deducir atributos a partir de otros, por ejemplo si A implica a B significa que siempre que se da A se dará B por lo tanto si A es positiva también lo será B. Esto permite reducir el volumen de información que el usuario debe de introducir en el sistema.

El mayor problema de los sistemas basados en implicaciones radica en que el conjunto de éstas suele ser demasiado grande y, por lo tanto, poco eficiente. Sin embargo para solucionar este problema existen técnicas matemáticas para transformar este conjunto en base, es decir, el menor conjunto de implicaciones que mantiene las mismas propiedades. Además es posible usar esta base y los operadores de cierre para obtener *conceptos*.

Al contrario que con los *conceptos* los algoritmos que trabajan sobre implicaciones requieren un *contexto formal* con valores pertenecientes al intervalo [0,1]. Existen algunos algoritmos como TITANIC (ver algoritmo 4) que permite extraer implicaciones sobre *contextos* binarios. Para los *contextos formales* con valores reales existen dos aproximaciones principales:

Reglas de asociación Expresa un casi todas, por ejemplo para $A \overset{90\,\%}{\rightarrow} B$ significa que el 90 % de las tuplas cumplen la implicación.

Algoritmos difusos (fuzzy) Nos indica una similitud de atributos. Un ejemplo podría ser $A \Rightarrow^{75\,\%} B$ que nos muestra que A y B no difieren más del 75 % en todas las tuplas.

Esta será la parte específica del TFG desarrollado por mi compañero y no será objeto de estudio en esta memoria.

7. Análisis de requisitos

En las siguientes secciones se analizarán cada uno de los requisitos funcionales enunciados en la sección 5, dando los supuestos que llevaron hasta la solución elegida.

7.1. RF-01 Obtener el conjunto de atributos para el contexto formal

En los sistemas para obtener palabras clave (*keywords*) a partir de textos existen dos vertientes básicas:

- Los sistemas que tienen los atributos prefijados e inalterables. En estos sistemas a los atributos se les suelen denominar *tags*.
- Los sistemas que generan las *keywords* dinámicamente.

En este TFG se ha preferido optar por la segunda opción. Generar de esta manera los atributos permite crear un sistema más genérico y adaptable a un mayor número de campos. Este aspecto es muy deseable puesto que los atributos de los POIs son muy distintos dependiendo del objeto en cuestión, no es lo mismo un hotel, un restaurante o un museo. Sin embargo esto ocasiona varios problemas añadidos como son la necesidad de entrenamiento previo y el análisis del texto. La realización de un entrenamiento previo no es un gran problema, pero el análisis de texto sí.

Inicialmente se barajaron dos opciones para extraer los atributos a partir de los textos.

- Por un lado se realizó una búsqueda de algoritmos y técnicas para solventar este requisito. En la gran mayoría de los artículos localizados también se expone el problema del análisis de sentimientos (aspecto no necesario para este TFG) dando por sentado que se poseen los términos o *keywords*, que son los posibles candidatos a convertirse en atributos. Por este motivo fue especialmente difícil encontrar documentación útil. Algunos artículos usan teoría probabilística y buscadores para obtener *keywords*.
- También se estudió la herramienta de software libre Gate [24]. Esta herramienta posee una aplicación de escritorio y un API para Java. Al ejecutarse sobre el texto genera una marcación en él en forma de etiquetas, catalogando las palabras. Dicho etiquetado lo realiza gracias a una serie de diccionarios internos. El principal problema que ofrece Gate es su limitado diccionario, el cual no contenía términos aptos para el sector turístico. Por este motivo, ante la necesidad de utilizar algún tesauro para ampliar el diccionario propio se desechó esta herramienta.

Continuando con la idea de emplear algún producto para solucionar esta necesidad se realizó una búsqueda de nuevas herramientas. Se comprobó que existen numerosos

productos que ofrecen extracción de *keywords*, sin embargo la mayoría no cubrían las necesidades del TFG [39, 46]. Posteriormente se probó R [43] herramienta inicialmente pensada para el campo de la estadística pero que se está empleando en minería de datos y procesamiento del lenguaje natural. Sin embargo, al final se optó por AlchemyAPI [2]. Esta es una herramienta que funciona como un servicio web con licencia. Tras descargar los códigos fuentes necesarios para su uso (se encuentran en múltiples lenguajes de programación incluido Java) se puede realizar la extracción de *keywords* de manera muy sencilla. La salida de este análisis del texto es un documento XML lo cual facilita enormemente su tratamiento. La licencia de uso gratuita está limitada a 1000 peticiones diarias, mientras que la versión académica son 30000 peticiones. Estos números son más que suficientes para el sistema que se desea construir.

Una vez obtenidas todas las *keywords* de los textos solo serán atributos un número reducido de ellas. Los atributos deben de ser términos relevantes para los POIs por lo tanto se debe de realizar algún tipo de filtrado que elimine las palabras poco relevantes. El tipo de filtrado que se empleará en este TFG será un filtrado por frecuencia. Las palabras más relevantes serán aquellas que aparezcan en un mayor número de comentarios. Una vez obtenidas todas estas frecuencias solo es necesario cribarlas fijando un umbral para obtener el conjunto de atributos M.

7.2. RF-02 Procesar los comentarios para rellenar la tabla que representa al *contexto formal*

Sigue la misma filosofía que la sección 7.1, extraer *keywords* para su posterior procesamiento. Para obtener estos términos a partir de los textos se emplea la misma herramienta AlchemyAPI. Para cada palabra de este conjunto habrá que comprobar si se encuentra o no en el conjunto de atributos M y de esa forma ir rellenando las filas de la tabla.

Para los valores de la tabla se debatió entre asignar valores binarios (0 o 1) o en el intervalo real [0,1]. Se tomó la decisión de elegir los binarios debido a que la mayoría de los algoritmos para extraer *conceptos*, necesarios para la parte individual del sistema, trabajan sobre *contextos formales* de este tipo.

Un aspecto importante de esta sección y de la 7.1 es el problema de los sinónimos. AlchemyAPI no realiza ningún tipo de tratamiento de sinónimos en los textos por lo que al agrupar por frecuencias no se tiene en cuenta atributos que deberían de ser uno solo. El problema de los sinónimos es un problema complejo que depende enormemente del idioma en el que se escribe el texto. Existe muchísima literatura sobre este tema [56][58] siendo la principal aproximación el uso de tesauros. Estos diccionarios o tesauros están diseñados mayoritariamente para el idioma inglés existiendo herramientas menos potentes en otros idiomas. Debido a la dificultad para obtener una herramienta útil y sencilla que solucione este aspecto se ha dejado esta parte para futuros proyectos y se ha realizado el

tratamiento de los sinónimos manualmente.

7.3. RF-03 Almacenamiento del *contexto formal* con un formato concreto

Gracias a la popularidad de la teoría de Ganter existen multitud de herramientas y programas sobre FCA. La página web del grupo Uta Priss [62] contiene una gran cantidad de referencias a software relacionado con esta teoría. El principal inconveniente de estas herramientas es que cada una de ellas requiere de un formato concreto para la información del *contexto formal* que reciben como entrada. Las extensiones más utilizadas son: .cxt, .tlb, .xls y .csv. Cada uno requiere los datos de manera diferente por lo que es muy difícil establecer un formato general que permita al *contexto formal* ser leído por todas las aplicaciones. Este es el motivo por el que en este TFG solo se hará uso de la extensión .csv al tener una estructura muy sencilla de entender y ser interpretada por programas de hojas de cálculo.

7.4. RF-04 Obtener todos los conceptos a partir de una tabla de FCA

Se estuvo barajando si emplear herramientas ya desarrolladas, como en la parte común, o realizar manualmente la implementación de algún algoritmo. Debido a la similitud con la cual se describen los algoritmos de obtención de *conceptos* se seleccionó la segunda opción.

También se analizó cual algoritmo implementar. Existen una gran cantidad de algoritmos de múltiples autores que tratan de solucionar el problema de generar el *retículo de conceptos* [53].Hay diversos tipos de algoritmos, algunas soluciones solo generan partes del retículo mientras que otros, para casos más concretos, lo generan completos y además siendo relativamente eficientes. En base a estos datos se decidió realizar la implementación del Titanic (ver algoritmo 4) frente al NextClosure (ver algoritmo 3). Se decidió decantarse por el primero debido a que, aunque no se reduce la complejidad exponencial, se reduce la complejidad del NextClosure

$$O(\underline{\mathfrak{B}}(\mathbb{K}) \cdot (|G| \cdot |M|^2))$$

frente a la de Titanic

$$O\left(|M| \cdot \left(db + \binom{|M|}{\lfloor \frac{|M|}{2} \rfloor} \cdot |G| \cdot |M|\right)\right)$$

El número de accesos al *contexto* de NextClosure es $2^{|M|}$ mientras que Titanic accede

a lo máximo $|M|$. Esto hace que se mejore enormemente la eficiencia sobre todo para *contextos* muy grandes (en ese caso *db* será el campo dominante de la complejidad).

7.5. RF-05 Representación de los conceptos

Se realizó una búsqueda de plugins para representar los *conceptos* de manera jerárquica. Tras no encontrar ninguno que fuera fácilmente integrable se optó por realizar una implementación de algún algoritmo para representar retículos. Sin embargo, estos algoritmos entrañan una cierta complejidad saliéndose del alcance de este TFG, por lo que también fue desestimada. Finalmente, tras analizar varias herramientas sobre FCA [62] se decidió emplear la herramienta externa Graphviz [25] para la representación del *retículo*.

7.6. RF-06 Integración con la parte común

Al ser un sistema que funciona a partir de los resultados de la parte común y es un desarrollo no existen inconvenientes de integración, el sistema generado en este TFG se enmarca dentro de la creación de un mismo framework. Esto provoca que no haya ningún problema para acceder a los ficheros de ambos subsistemas. Además, al ser este sistema un desarrollo se ha tenido este requisito en cuenta a la hora de construirlo para que sea interoperable. En la sección 9 se desarrollará en mayor detalle esta integración.

8. Entorno tecnológico

Se ha desarrollado una aplicación web Java EE (antes conocido como J2EE) versión 6, implementada haciendo uso de los siguientes frameworks y librerías

- JBoss 6 [28]: Servidor de aplicaciones. Fue elegido debido a la familiaridad previa que se tenía con el mismo.
- Spring v4 [47]: Framework de desarrollo de entornos web. Es una de las mejores tecnologías de modelo vista-controlador. Es fiable, posee una gran cantidad de módulos fácilmente configurables y es fácil de usar. Se ha empleado de controlador de la aplicación web.
- Bootstrap 2.3.1 (look & feel) [6]: Hojas de diseños css originalmente diseñada por Twitter. Se ha vuelto muy popular debido a los buenos resultados que se obtienen y a su sencillez. Concretamente se ha utilizado la plantilla gratuita Flatly [21] basada en Bootstrap.
- JSP: Para el diseño de la interfaz web. Son rápidos, potentes, dinámicos, sencillos y altamente extendidos. Su simplicidad fue la clave para su selección.
- JTLS: Etiquetas para JSP que añaden funcionalidad extra. Concretamente se han empleado para sentencias de control de flujo <c:if> y <c:forEach> además de acceso a los ficheros .properties para implementar el multilenguaje.
- Apache Ant (Solo en front-end) [4]: Librería para la compilación del código.
- Maven (Solo en back-end) [38]: Herramienta para el control de configuración. Ampliamente extendido, potente y fácil de usar.
- AlchemyAPI [2]: Librería para el procesamiento del lenguaje natural que ofrece una amplia gama de funcionalidades. Se usa bajo licencia de compra, aunque posee licencias de prueba y académica (limitada a un número máximo de peticiones).

Además se han empleado una serie de plugins para mejorar la funcionalidad y la apariencia de la web.

- jQuery [30]: Biblioteca de JavaScript que permite simplificar la manera de interactuar con los documentos HTML, manipular el árbol DOM, manejar eventos, desarrollar animaciones y agregar interacción con la técnica AJAX a páginas web. Es la librería de JavaScript mas utilizada. Además de JQuery se hace uso de una serie de plugins concretos de su biblioteca.
    - JQuery BlockUI [31]: Bloquear la pantalla mientras se procesa la información.

    - jQuery jGrowl [33]: Mostrar alertas.
    - jQuery Upload File [34]: Subir archivos de manera eficiente, integrable con AJAX y permitiendo Drag & Drop.
    - jQuery Validate [35]: Validar campos del formulario. Además es altamente personalizable
    - jQuery File Download [32]: Descargar archivos.
- AJAX [1]: Empleado para recargar solo lo necesario de la página. Permite además gestión de eventos y control de las peticiones al servidor.

La parte específica del TFG se desarrolló como una ampliación de la funcionalidad del sistema desarrollado en la parte común, por lo que solo se añadió al código de back-end la siguiente librería y herramienta:

- JGraphT [29]: Librería para el tratamiento de grafos. Ofrece un cierto número de algoritmos y funcionalidades fáciles de usar.
- Graphviz-2.38 [25]: Herramienta para la construcción de gráficos. Ofrece lectura de archivos .dot y transformadores a .png.

En el esquema representado en la figura 1 se resume el funcionamiento básico del sistema completo:

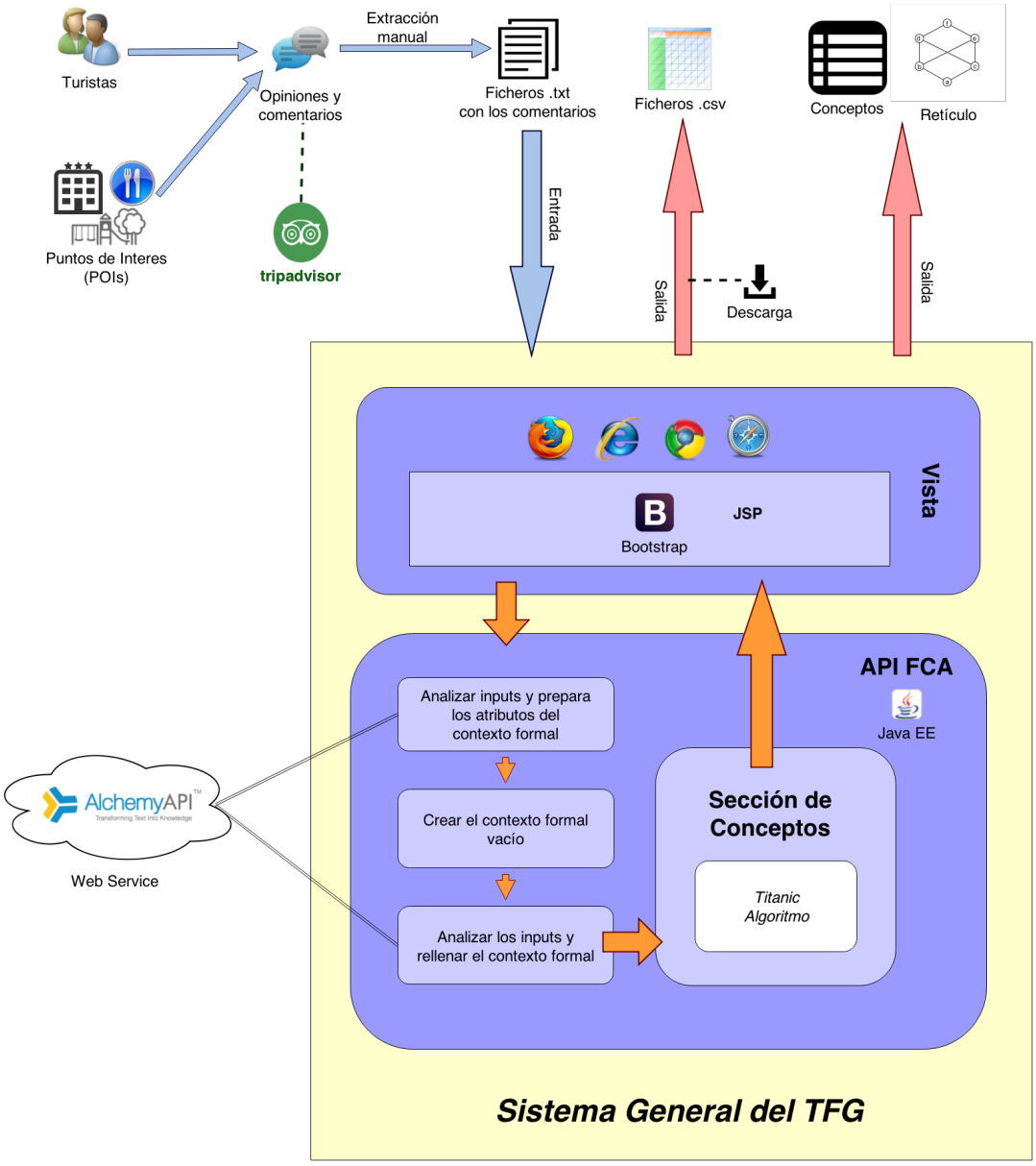

Figura 1: Diagrama de arquitectura del sistema completo

9. Desarrollo

En esta sección se expondrán los principales aspectos el desarrollo realizado para construir el sistema.

9.1. Aspectos básicos

El desarrollo se realizó sobre el sistema operativo Windows 7 empleando el IDE Eclipse [13] para la construcción de la aplicación. El sistema se divide en dos proyectos diferentes:

- FCAApplication: Interfaz web para poder visualizar el trabajo realizado sobre FCA y proporcionar una UI más amigable para su tratamiento. Es el front-end del sistema.
- FCAUtilitiesProject: Una API con todas las funcionalidades relacionadas con FCA desarrolladas para el sistema descrito en esta memoria. Es el back-end del sistema. El desarrollo de la parte específica del TFG se realizó ampliando la funcionalidad de esta API.

Se tomó la decisión de realizar esta separación para mejorar su integración con otras aplicaciones. Estos sistemas externos solo tendrán que importar la API y podrán emplear todas las funcionalidades descritas en la sección 9.3.

9.2. Estructura de los proyectos

A continuación se presentan la estructura de los proyectos desarrollados.

9.2.1. FCAApplication

La extructura es la de un proyecto Java Web siguiendo el patrón modelo-vista-controlador (ver figura 2). En él se distinguen las siguientes partes:

- src.controllers, en donde se encuentra el controlador de la aplicación
- build, carpeta en la que se almacena el FCAApplication.war
- Libraries, librerías externas necesarias para la aplicación. En esta carpeta va incluido el código compilado de la API de FCA: FCAUtilitiesProject.jar
- web, en donde se encuentra el código relativo a la vista de la aplicación. Se compone de los directorios:
  - bootstrap, que contiene los archivos css, imagen y javascript.
  - WEB-INF, que se compone a su vez de:
    - jstl: contiene los archivos .tld necesarios para el multi-idioma.

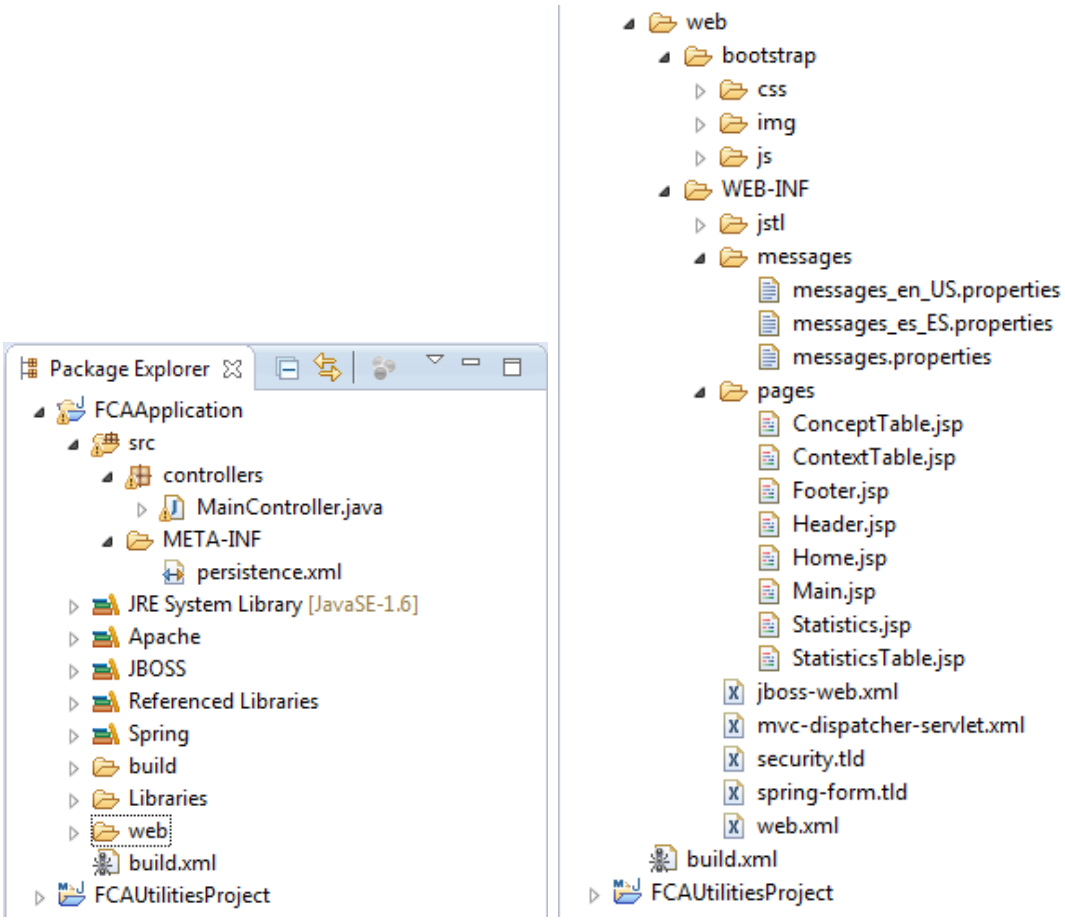

Figura 2: Proyecto de front-end: FCAApplication

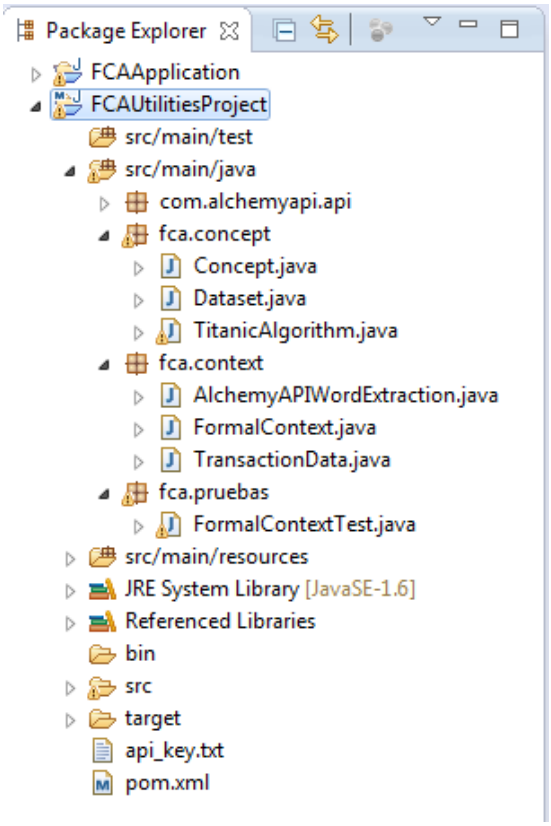

Figura 3: Proyecto de back-end: FCAUtilitiesProject

- messages: en él se encuentran los archivos .properties para las traducciones. Solo se han implementado los idiomas español e inglés (el idioma por defecto es inglés).
- pages, que contiene las páginas JSP de la aplicación.

- build.xml, archivo Ant para la compilación.

9.2.2. FCAUtilitiesProject

Este proyecto se estructura como un proyecto Maven y se divide básicamente en 4 paquetes (ver figura 3):

- com.alchemyapi.api: contiene los códigos fuentes de AlchemyAPI.
- fca.concept: en él se encuentra la implementación de TITANIC y clases auxiliares (ver secciones 9.3.2, 9.3.2 y 9.3.2).
- fca.context: paquete principal en el que se encuentran las clases básicas para trabajar con *contextos formales* (ver secciones 9.3.2, 9.3.2 y 9.3.2).
- fca.pruebas: en su interior se encuentra la clase de prueba que se empleó en el desarrollo beta del front-end.

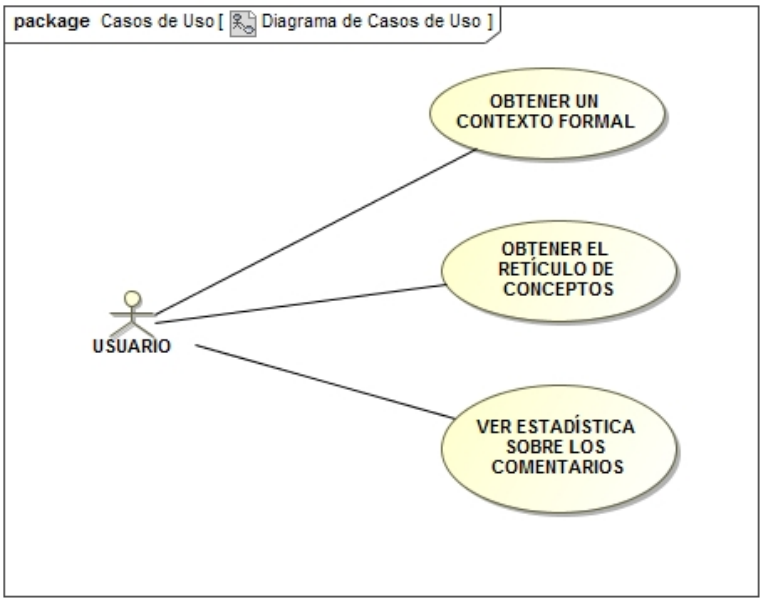

Figura 4: Diagrama con los casos de uso

9.3. Modelos

9.3.1. Casos de Uso

En la figura 4 se ilustran los tres casos de uso principales del sistema.

- Obtener un contexto formal: Indica la construcción y obtención de la tabla con la información del *contexto formal* a partir de los comentarios introducidos por el usuario.
- Obtener el retículo de conceptos: Partiendo del *contexto formal* obtener el *retículo de conceptos*.
- Ver estadística sobre los comentarios: Ver información de ayuda para saber que umbral aplicar al *contexto*.

9.3.2. Diagramas de Clases

En la definición del dominio del problema se diseñó el diagrama de clases descrito en la figura 5.

Para la implementación de este diagrama en el proyecto FCAUtilitiesProject se realizaron algunas modificaciones. A continuación se definen las clases y métodos principales de este proyecto. Estas clases se ilustran en el diagrama de clases de la figura 6.

FormalContext Esta clase representa a un *contexto formal* y es la clase principal del proyecto. Se compone de una serie de constructores y métodos, los cuales se describen a continuación

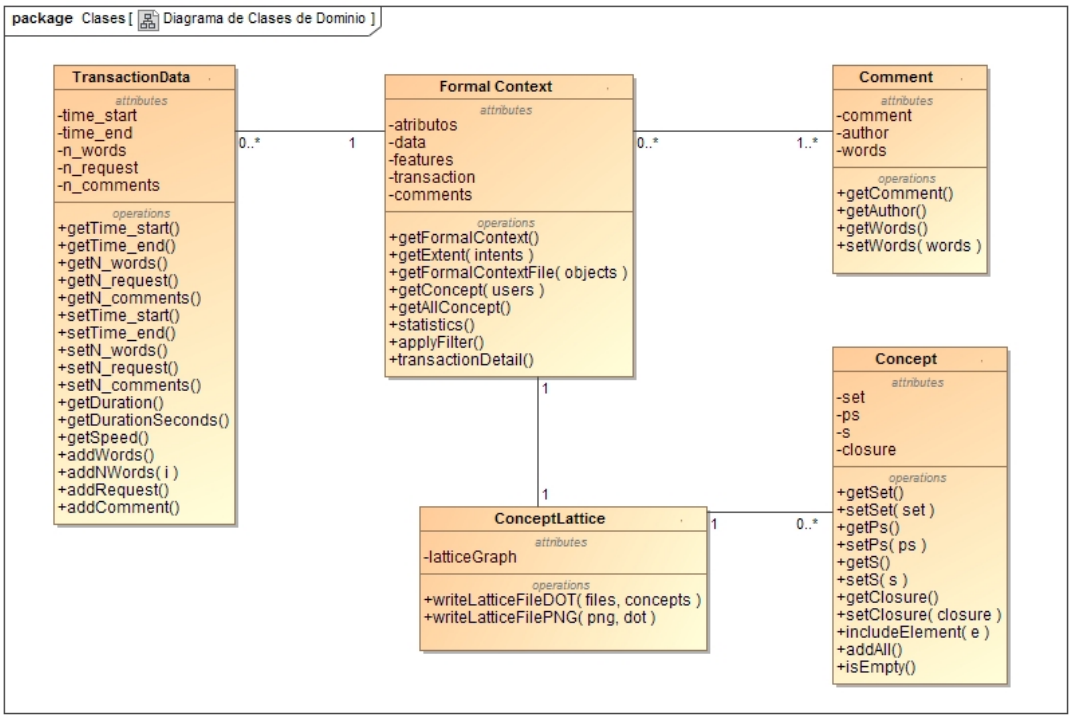

Figura 5: Diagrama de clases del dominio

Constructores

```
public FormalContext();
public FormalContext(List<File> files);
public FormalContext(List<File> files , int threshold); // Constructor
 principal
public FormalContext(File file); // Constructor principal
```

Estos constructores reciben como parámetros un archivo .csv con la información del *contexto formal* o una lista de comentarios en .txt y el umbral por el cual cribar los atributos. Este último es el que tiene implementada la llamada al código de AlchemyAPI (ver sección 9.3.2) para procesar los textos. El resto de constructores son auxiliares y sirven para facilitar otras tareas. Cabe destacar que el *contexto* se rellena de la siguiente forma:

- Si el comentario contiene el *keyword* colocará un "1".
- En caso contrario un "0".

Métodos sobre el contexto formal Los métodos principales para interactuar con el *contexto formal* son los siguientes:

```
public List<List<String>> getFormalContext() {};
public List<String> getExtent(List<Set<String>> intents) {};
public File getFormalContextFile(List<String> objects) {};
```

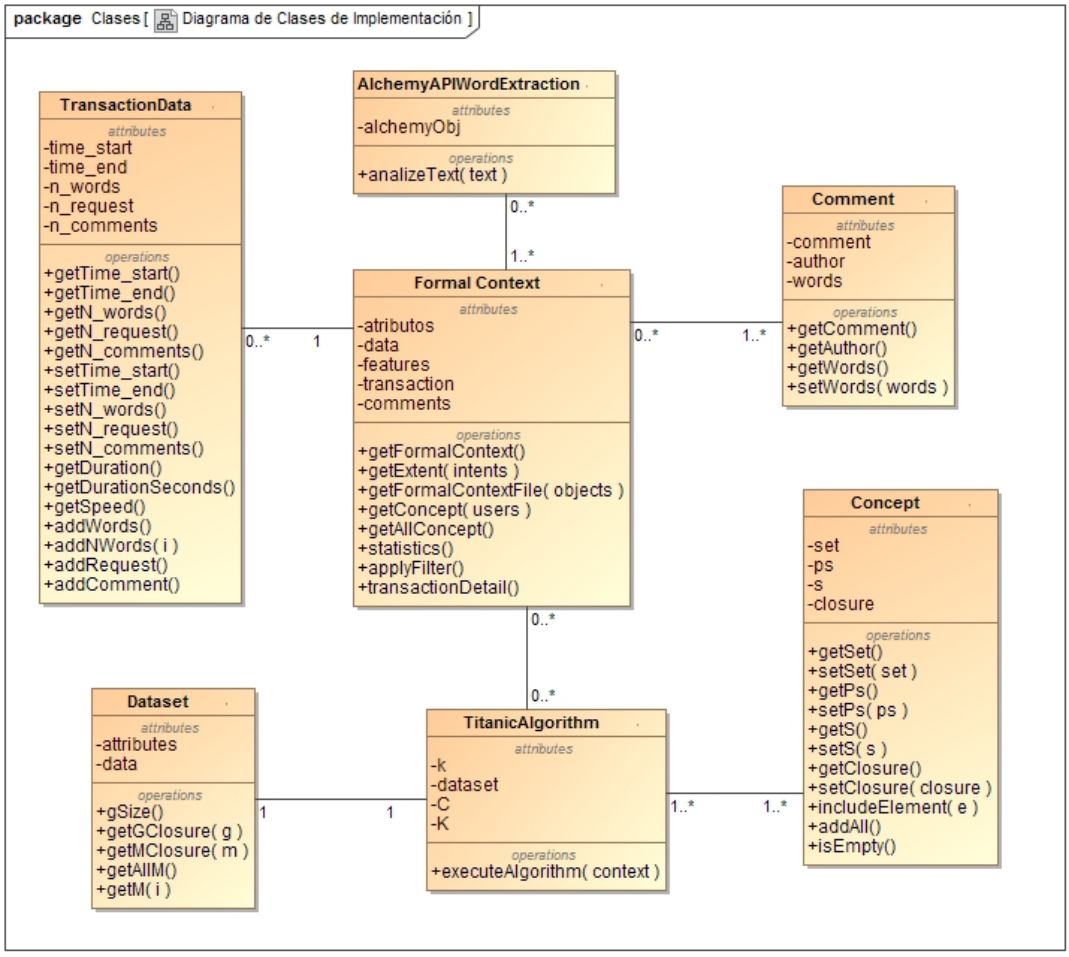

Figura 6: Diagrama de clases de implementación

El primer método sirve para obtener el contexto formal mientras que el segundo nos proporciona una lista que podrá ser usada para filtrar los objetos que cumplan los conjuntos del parámetro. La List<Set<String>> intents realmente representa un concepto pero se ha preferido usar conjuntos de String en lugar de una clase para dar más sencillez a la aplicación web. El último método es usado para poder descargar el *contexto* como un archivo .csv.

Métodos sobre conceptos La parte específica del TFG trata sobre los *conceptos*. Estos son los métodos que proporcionan las funcionalidades necesarias:

```
public List<Set<String>> getConcept(List<String> users) {};
public List<Set<String>> getAllConcept() {};
```

Estos métodos nos devuelven los *conceptos*. Ambos poseen llamadas a la implementación del algoritmo TITANIC la cual se describe en la sección 9.3.2.

Además existen dos métodos para poder visualizar el retículo mediante un archivo .png

Métodos para obtener el retículo

```
public void writeLatticeFileDOT(File file, List<Set<String>> concepts){};
public void writeLatticeFilePNG(File png, File dot) {};
```

Gracias al primero obtenemos un archivo .dot. Para ello se emplea JGraphT que genera el grafo que representa al retículo. El segundo realiza una llamada al sistema para ejecutar Graphviz y generar un archivo .png con el gráfico descrito en el fichero .dot.

Métodos para la obtención de estadísticas Desde el comienzo de la construcción de la aplicación se tuvo la necesidad de generar algún tipo de ayuda para la construcción de los *contextos formales* así como mostrar información relativa al rendimiento de cada transacción. Estos son los métodos dedicados a estas funciones:

```
public List<List<String>> statistics() {};
public void applyFilter(int threshold) {};
public TransactionData transactionDetails() {};
```

El primer método nos devuelve un pequeño estudio sobre cuántos atributos se obtienen al aplicar cada umbral posible. El uso de este método requiere usar el constructor especial public FormalContext(List<File> files).

El método applyFilter(int threshold) sirve para aplicar un umbral al *contexto formal* y sería la segunda parte del constructor public FormalContext(List<File> files).

El último método nos devuelve parámetros de la última transacción (método) del *contexto formal*. Estos datos se devuelven como la instancia de una clase la cual será explicada en la sección 9.3.2.

AlchemyAPIWordExtraction Esta clase es la encargada de realizar la extracción de las palabras claves de los textos que posteriormente serán cribadas para obtener los atributos del *contexto*. Es una abstracción de la API de AlchemyAPI. Se compone principalmente de dos instrucciones:

- Un constructor public AlchemyAPIWordExtraction() que crea una instancia de la clase AlchemyAPI (incluida en el paquete com.alchemyapi.api) clase que se encargará de realizar las peticiones al servicio web. Un aspecto importante es que para poner instanciar esta clase es necesario que en la raíz de nuestro proyecto tengamos un archivo llamado api_key.txt que contenga una clave de uso suministrada por AlchemyAPI.
- Un método List<String> analizeText(String text) que es el encargado de realizar la petición de análisis y prepara la información obtenida de manera que sea fácilmente tratable por el *contexto formal*. En esta información no se devuelve ninguna cantidad relativa a la frecuencia de las palabras en el texto. Se tuvo en cuenta este supuesto por dos motivos:
  - Simplificar el código
  - El procesamiento se realizaría a nivel de comentario y sin emplear análisis de sentimientos o ningún tipo de relevancia en las *keywords*. Por ese motivo solamente se tuvo en cuenta la aparición de la palabra y no el número de veces que se presenta.

TransactionData Para almacenar la información relativa al tiempo de ejecución y número de operaciones realizadas en el procesamiento de los métodos principales de la clase FormalContext se emplea esta clase Java. Contiene una serie de parámetros, con sus correspondientes métodos get y set, para almacenar esta información:

```
// Time of start and end of the transaction
private long time_start, time_end;

// Number of words processed
private int n_words;

// Number of requests processed
private int n_requests;

// Number of comments processed
private int n_comments;
```

Además posee una serie de métodos para incrementar el número de palabras, peticiones y comentarios, y así facilitar su uso:

```
public void addWord() {};
public void addWord(int i) {}; // add i to n_words
public void addRequest() {};
public void addComment() {};
```

Y métodos que devuelven una información un poco más elaborada como la duración:

```
public long getDuration() {}; // duration in milliseconds
public double getDurationSeconds() {}; // duration in seconds
```

O el número de comentarios por segundo (se devuelve un String que representa un double)

```
public String getSpeed() {};
```

TitanicAlgorithm Es una implementación directa del algoritmo TITANIC (ver algoritmo 4). Consta de un único método público

```
public static List<Set<String>> executeAlgorithm(List<List<String>>
 context) {};
```

Este método recibe como entrada el *contexto formal* en formato List<List<String>>, y devuelve una lista de conceptos. Se tomó la decisión de emplear Set<String> para representar los conceptos para abstraer lo máximo posible este código y que sea fácilmente integrable con otras aplicaciones.

Para facilitar la lectura del código se mantuvieron tanto los nombres de los métodos como los de las variables que aparecen en la definición de TITANIC. Además se crearon algunos métodos auxiliares y una clase para representar los *conceptos* (sección 9.3.2). Estos métodos son:

```
private static Set<Concept> titanic() {};
private static Set<Concept> titanicGen(Set<Concept> Kant) {};
private static Concept closure(Concept x) {};
private static void weight(Set<Concept> Xset) {};
private static Set<Concept> subsets(Concept x, int i) {};
private static boolean cCandidate(Concept i, Concept j) {};
```

Concept Contiene toda la información que TITANIC tiene que almacenar sobre cada uno de los conceptos que genera. Sus parámetros son:

```
// Set of attributes. Represent the concept
private Set<String> set;

// Titanic variable: ps
private float ps;

// Titanic variable: ps
```

```
private float s;

// Closure of set. closure:: A -> A''
private Concept closure;
```

Además se definen métodos para añadir elementos al conjunto, comprobar si el conjunto es vacío o si un concepto está incluido en él.

```
public void add(String string) {};
public void addAll(Concept i) {};
public boolean isEmpty()  {};
public boolean includeElement(Concept e) {};
```

Dataset Representa un *contexto formal* para TitanicAlgorithm. Esta diferencia con respecto a la clase FormalContext es debido a la facilidad de integración con otros sistemas que se persigue en este desarrollo.

Define esta serie de métodos que emplea la implementación de TITANIC.

```
// Return G size
public int gSize() {};

// For g included in G, return g'
public Concept getGClosure(int g) {};

// For m included in M, return m'
public Set<Integer> getMClosure(String m) {};

// Return all m included in M
public Set<String> getAllM() {};

// Return the ith element in M
public String getM(int i) {};
```

9.3.3. Diagramas de Secuencia

Los diagramas de secuencia representados en las figuras 7, 8 y 9 incluidos en esta sección ilustran los comportamientos básicos y exitosos de los casos de uso descritos anteriormente.

9.4. Fases de desarrollo

A continuación se describe las fases seguidas en la construcción del sistema.

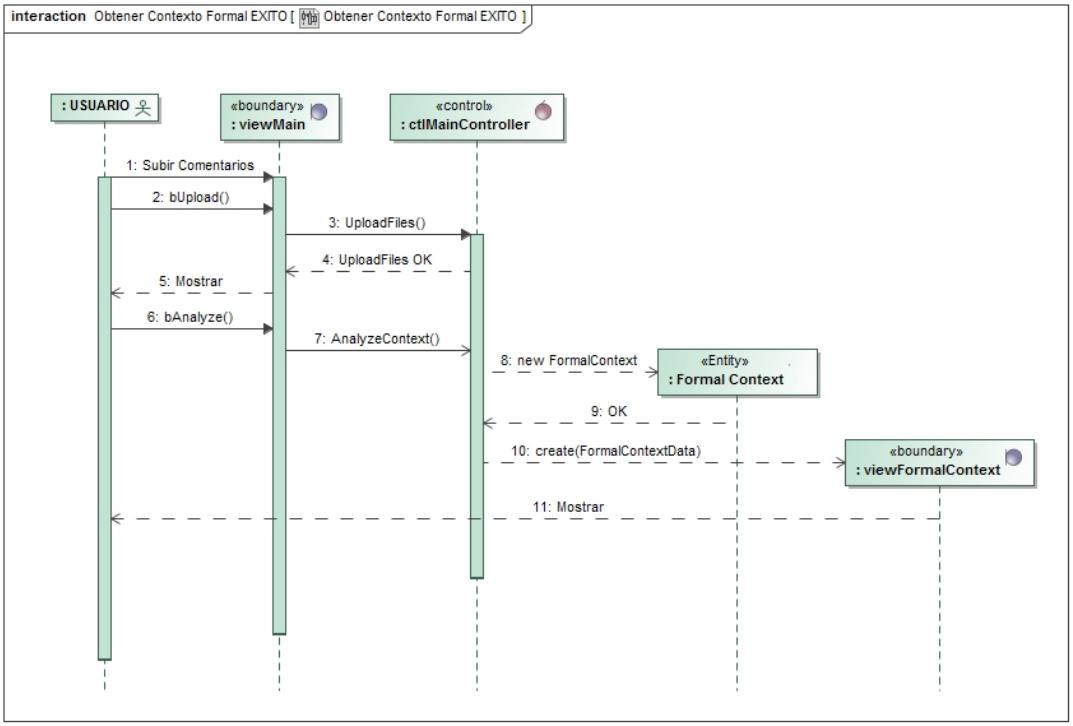

Figura 7: Diagrama de secuencia del caso de uso *Obtener un contexto formal*

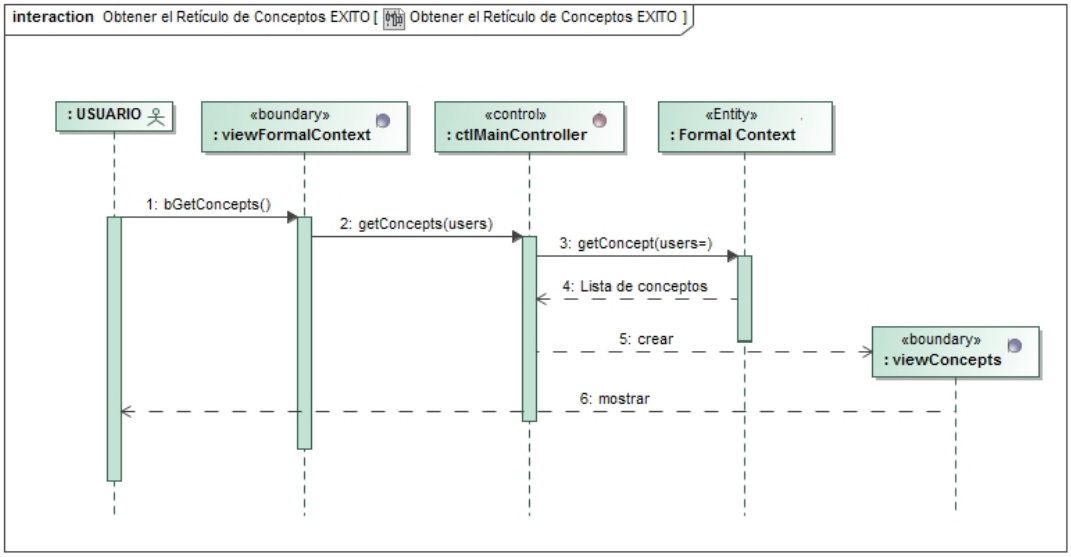

Figura 8: Diagrama de secuencia del caso de uso *Obtener el retículo de conceptos*

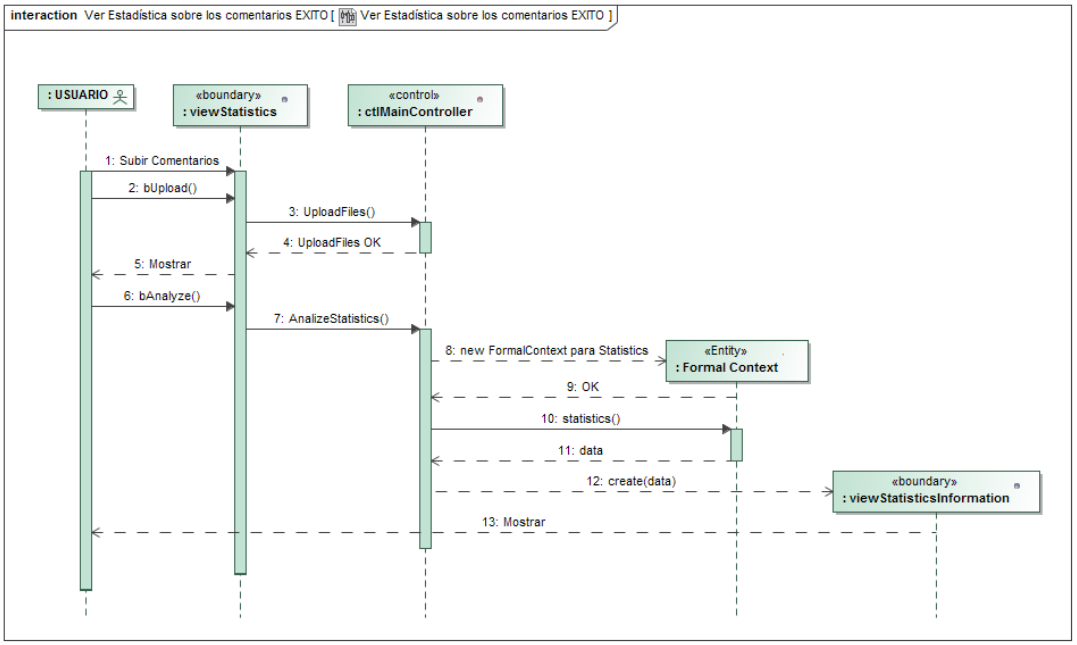

Figura 9: Diagrama de secuencia del caso de uso *Ver estadística sobre los comentarios*

1. Se realizó una primera fase alfa que sirvió como toma de contacto con el problema. Se desarrolló un código de prueba java sin UI para probar los conocimientos adquiridos y crear los primeros esqueletos de las funcionalidades deseadas para el sistema.

2. Tras fijar las funcionalidades deseadas se desarrolló una versión de prueba o prototipo de la API. Esta versión consistía en una interfaz del código final pero cuyos métodos devolvían valores constantes.

3. Empleando la interfaz de prueba se desarrolló una versión beta de la web. Paralelamente se implementaron los métodos finales de la API.

4. Una vez obtenidos los primeros prototipos funcionales se pasó a un modelo de desarrollo incremental hasta la construcción del sistema final. Este modelo comprendía las siguientes fases:

   a) Determinar los objetivos: Nuevas funcionalidades, mejoras y/o correcciones.

   b) Analizar los objetivos: Asignar una prioridad a cada uno de los objetivos siguiendo las siguientes pautas:

   - Prioridad Alta: Corrección de errores.
   - Prioridad Media: Nuevas funcionalidades críticas (necesarias para el sistema final).
   - Prioridad Baja: Mejoras de código y nuevas funcionalidades no críticas.

   c) Planificación: Seleccionar objetivos para la iteración de acuerdo a las prioridades y al tiempo necesario para desarrollarlas.

d) Desarrollo: Construcción y modificación del codigo necesario.

e) Testing: Pruebas del código. Son tanto de las nuevas funcionalidades como del sistema completo.

f) Detección de errores: Se registran los errores detectados para tenerlos en cuenta en la siguiente iteración.

5. Finalmente se realizó una fase de refinamiento, limpieza y refactorización de código.

9.5. Pruebas

Se realizaron una batería de 5 tipos de pruebas para API y 1 para la interfaz web. Las pruebas del código de back-end consistieron en:

Test AlchemyAPI Se testeó su correcto funcionamiento a partir de textos reducidos (entre 10 y 50 palabras) y textos erróneos (vacios o con caracteres extraños). Tras superar estas pruebas se introdujeron comentarios reales obtenidos de Tripadvisor y se comprobó la validez de los resultados obtenidos.

Pruebas de Creación de la Tabla Para crear la tabla que representa al *contexto formal* se siguieron los pasos descritos a continuación:

1. Se construyeron main Java para realizar tests en los cuales se comprobara que la salida del analizador fuera correcta y siguiera la estructura implementada. El conjunto de datos de entrada era controlado.
2. Con la ayuda de otro programa Java se comprobó que se pudiera descargar el *contexto formal* en forma de archivo .csv. Se emplearon las mismas entradas que en el paso anterior.

Extracción de conceptos Se comprobó el funcionamiento de todos los códigos de obtención de *conceptos* con dos tipos de ficheros de prueba.

- MUSHROOM.csv y MUSHROOMusers.csv: Ficheros basados en los datos de ejemplo del algoritmo Titanic (ver apéndices C.1 y C.2 para ver las tablas).
- Un conjunto controlado de comentarios de prueba obtenidos de Tripadvisor.

Visualización del retículo Se testeo los códigos de creación tanto de los archivos .dot como de los .png empleando los ficheros de prueba anteriores (MUSHROOM.csv y MUSHROOMusers.csv).

Pruebas de funcionamiento real Fueron repetidos los anteriores test con comentarios reales como inputs. De esa manera se aseguró el correcto funcionamiento de la API.

La interfaz web fue testeada una vez que la API pasó toda su batería de pruebas. Para estas pruebas se emplearon directamente datos reales y se comprobó tanto su correcto funcionamiento como algunas situaciones anómalas. A continuación se listan dichas situaciones:

- No introducir un umbral.
- No introducir comentarios.
- Introducir comentarios vacios.

9.6. Consideraciones finales

A continuación enumeramos algunas restricciones para el correcto funcionamiento de la aplicación:

- Los comentarios deben de estar en ficheros .txt.
- La primera línea de cada comentario debe de ser el nombre del usuario.
- Los comentarios deben de estar en inglés.
- Para analizar los comentarios previamente hay que subirlos, empleando para ello el botón designado a esa función, y una vez completada esta operación pulsar el botón "Analizar".
- Es favorable que cada comentario aparezca en un fichero diferente.
- Si no se conoce que umbral elegir se recomienda utilizar el apartado de Estadística para ver posibles umbrales.
- Se recomienda utilizar umbrales que generen alrederor de 10 atributos. Esto es especialmente importante si se va a hacer uso del sistema desarrollado para la parte individual de este TFG.
- Emplear el sistema operativo Windows para ejecutar la aplicación.
- No se ha aplicado responsive design debido a que las funcionalidades del sistema no están pensadas para el uso en smartphones.

Consideraciones para hacer uso de las funcionalidades propias de la parte específica del TFG (trabajo con *conceptos*:

- Los *conceptos* crecen exponencialmente con respecto al número de comentarios y atributos. Para obtener datos más fáciles de entender se recomienda usar un número reducido de comentarios y atributos.

- A mayor número de *conceptos* la visualización del retículo se hace más difícil de entender, por eso se recomienda emplear datos como los descritos en el anterior punto.
- Generalmente tablas con una mayor densidad de "1" dan un menor número de *conceptos*.
- Si se seleccionan muchos *conceptos* al obtener los usuarios que los cumplan es muy probable que no se obtengan resultados significativos.

10. Interfaz de usuario

A continuación se muestran los principales elementos de la interfaz de usuario, al tiempo que se explica su funcionamiento básico.

10.1. Página principal

La página principal nos presenta enlaces a las funcionalidades principales de la aplicación. Se compone de tres partes:

- Una cabecera con enlaces a las dos secciones principales y un menú desplegable en el que podremos cambiar el idioma de la aplicación como se ilustran en las figuras 10 y 11. El lenguaje por defecto es el inglés.
- El contenido principal que se divide en:
  - Una primera parte con botones que nos redirigen a documentación externa relacionada con FCA.
  - Descripciones y enlaces a las secciones principales de la aplicación.
- Un pie de página en el que aparecen los autores del sistema descrito.

La cabecera y el pie de página son los mismos para todas las páginas de la aplicación.

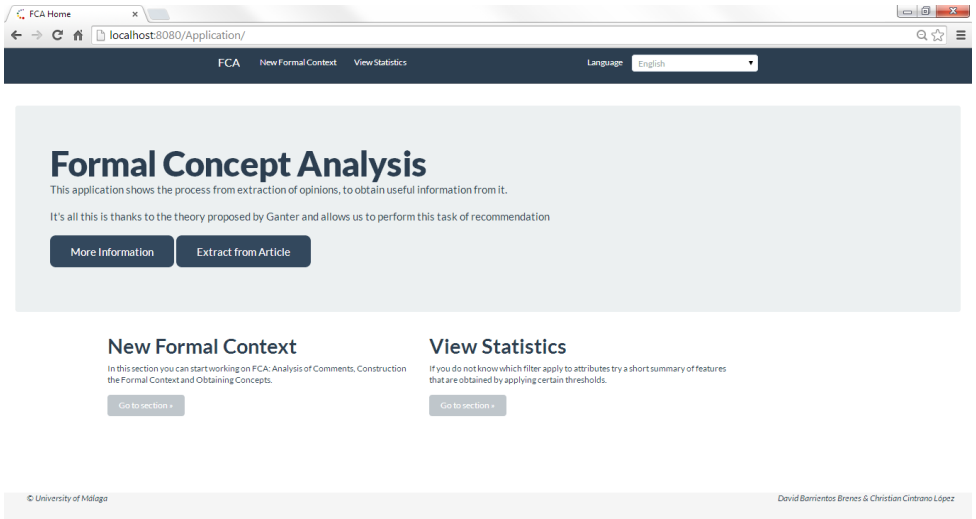

Figura 10: Página principal en ingles

10.2. Estadística sobre los comentarios

En esta página (figura 12) se pueden obtener unas estadísticas sobre los comentarios. Para ello se siguen los siguientes pasos:

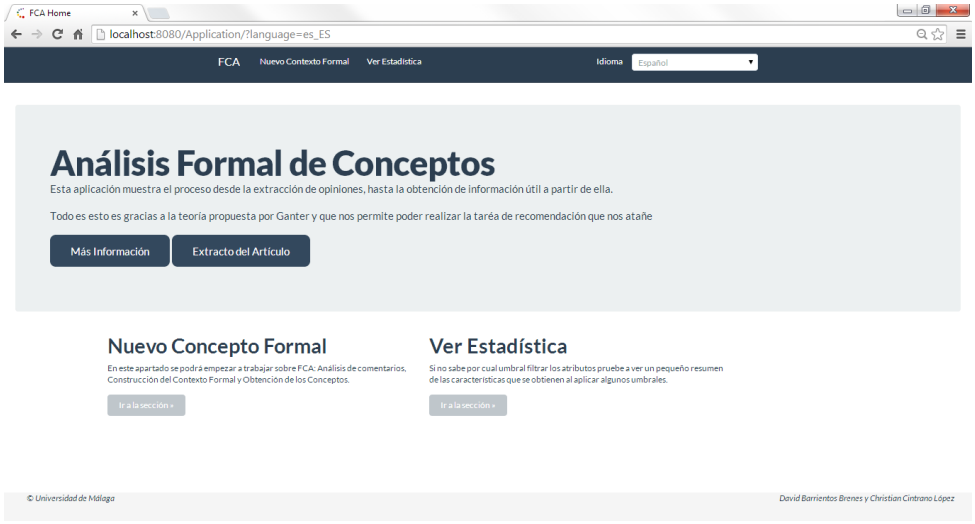

Figura 11: Página principal en español

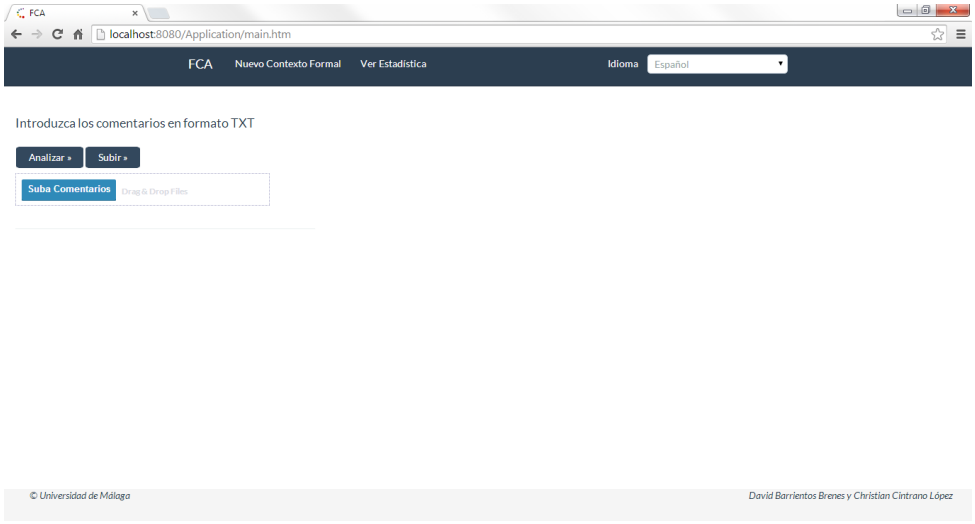

Figura 12: Página inicial de estadísticas sobre los comentarios

1. Presionar el botón *Subir Comentarios* o arrastrar los comentarios hasta el área. Los comentarios deben de ser archivos .txt.

2. Una vez añadidos todos los comentarios (figura 13) pulsar *Subir* y a continuación *Analizar*.

3. En este momento se nos mostrará la información en forma de tabla (figura 14). Estos datos nos enseñan para cada umbral cuantos atributos se obtienen y algunos ejemplos de estos. Además pulsando en la pestaña *Información* podemos ver información relativa a la transacción.

También es posible comenzar a trabajar sobre los comentarios subidos previamente. Para ello si pulsamos en el botón *Ver Contexto Formal* aparecerá una ventana emergente en la que introduciremos el umbral que deseemos.

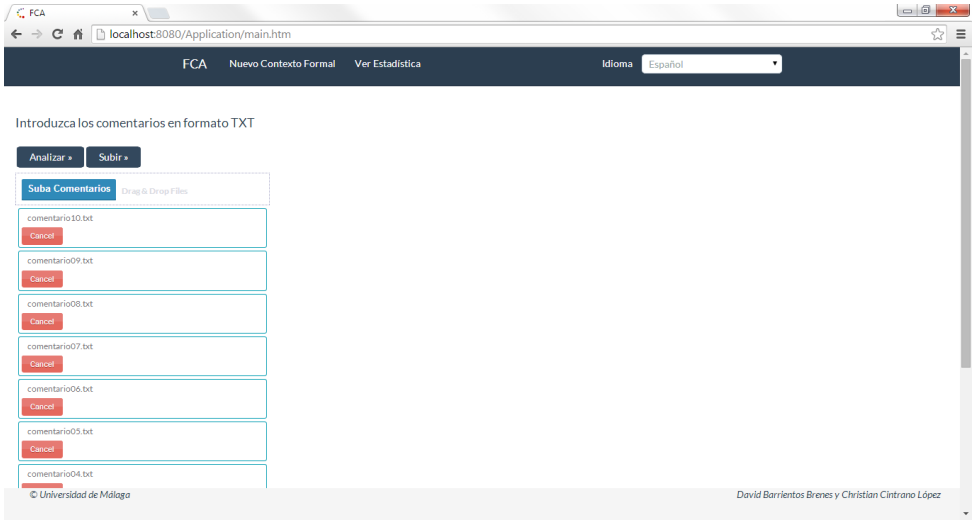

Figura 13: Página de estadísticas tras haber introducido comentarios

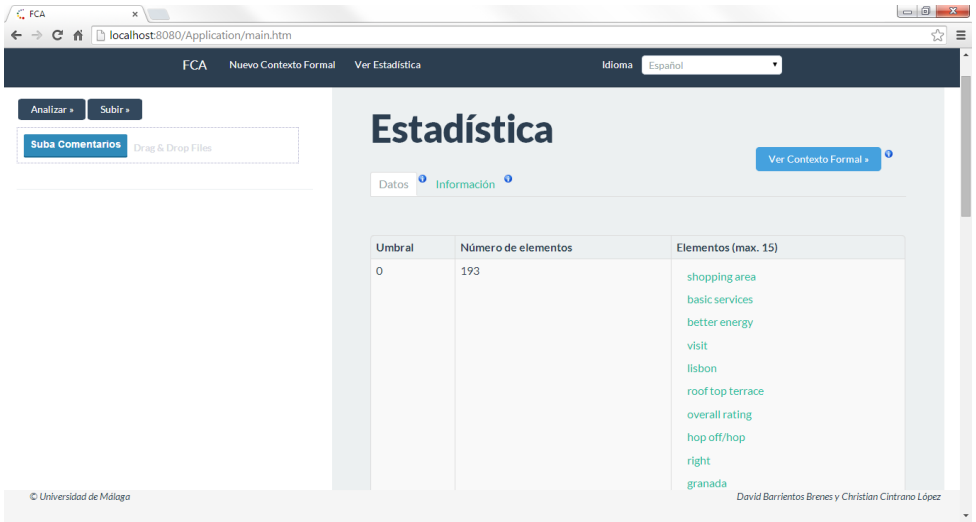

Figura 14: Página de estadísticas mostrando la información obtenida

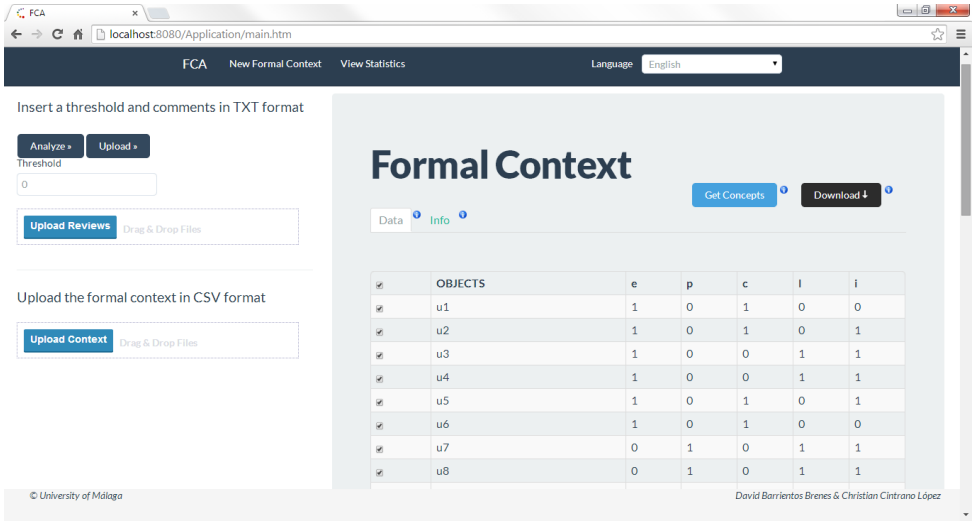

Figura 15: Página que muestra la tabla de doble entrada que representa al *contexto formal*

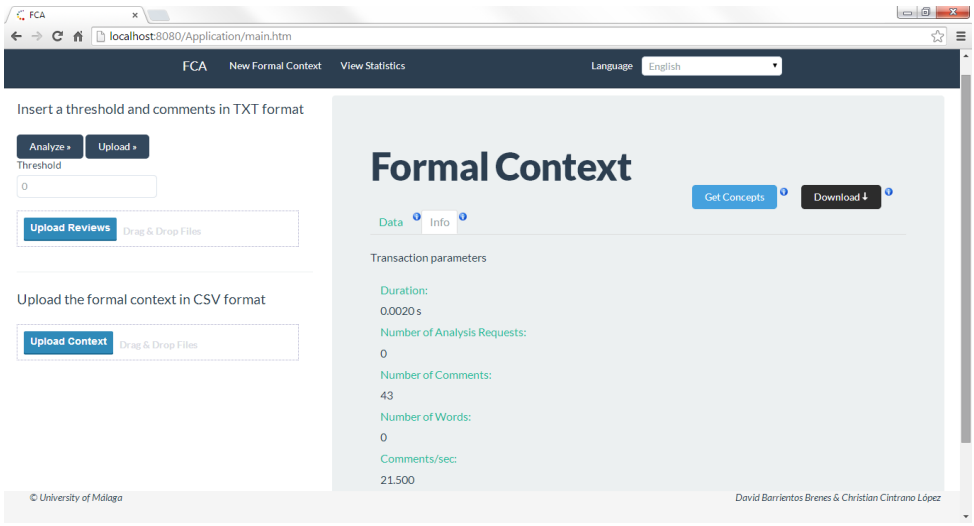

Figura 16: Estadística sobre la transacción de creación del *contexto formal*

10.3. Obtención del contexto formal

Para trabajar con *contextos formales* hay dos opciones.

- Crearlo desde comentarios, con lo que habrá que repetir los dos primeros pasos descritos anteriormente en la sección 10.2.
- Subir un *contexto formal* en formato .csv. Es necesario para el correcto funcionamiento que la primera columna del archivo sea el nombre del usuario.

Una vez mostrada la información (figura 15) podemos ver los datos de la transacción (figura 16), obtener los conceptos de aquellas filas que estén seleccionadas pulsando el botón *Get Concepts* o descargar el *contexto* formado por las filas seleccionadas.

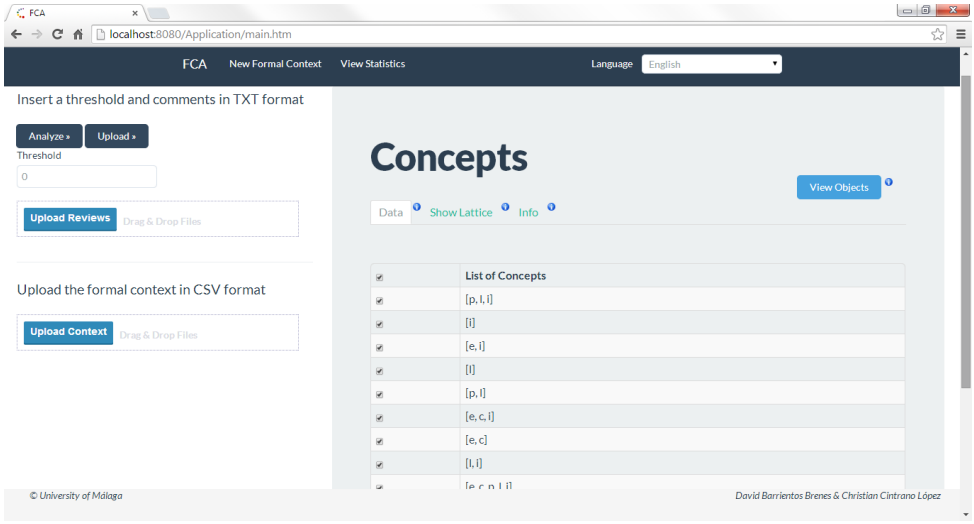

Figura 17: Página que muestra la lista de *conceptos*

10.4. Obtención del retículo de conceptos

En esta página se muestra una lista con los conceptos obtenidos a partir del *contexto formal* (figura 17). Además si pulsamos en la pestaña *Show Lattice* podremos ver el *retículo de conceptos* que representa estos datos (figura 18).

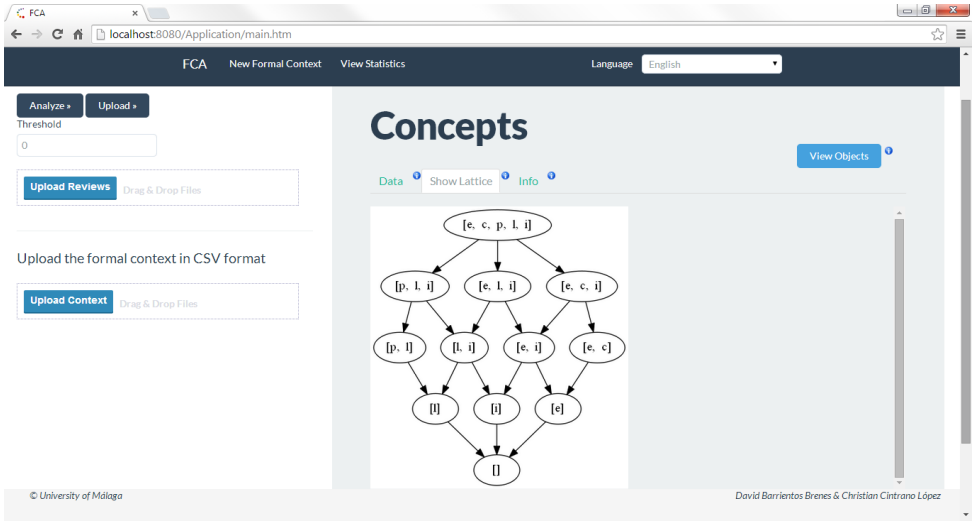

Figura 18: Gráfico que representa el *retículo de conceptos*

10.5. Filtrado de datos

Como se ha nombrado anteriormente es posible trabajar con un conjunto de los datos. A continuación se explicará el proceso de filtrado de datos en tres pasos.

1. Desde la pantalla de *conceptos* seleccionamos algunos de éstos y pulsamos el botón *View Objects* (figura 19).

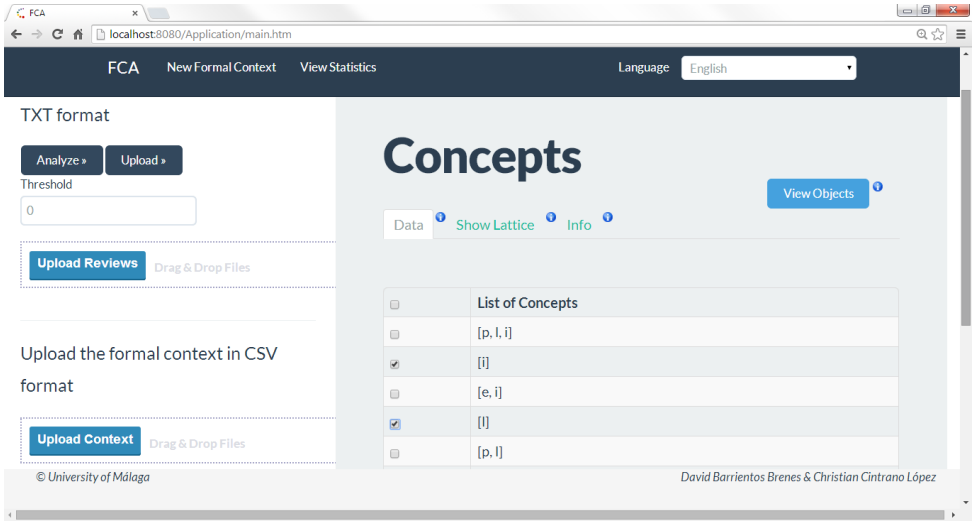

Figura 19: Selección de *conceptos* para ver los objetos que los cumplen

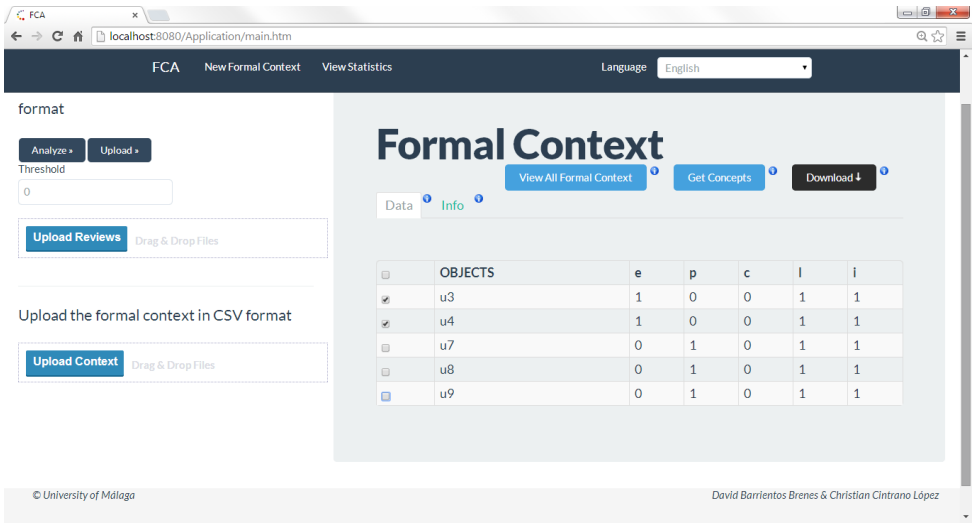

Figura 20: Lista de usuarios

2. Esta acción nos redirigirá a la pantalla del *contexto formal* pero con solo aquellas filas que cumplan los *conceptos* previamente seleccionados (figura 20). Además en este momento aparece un nuevo botón *View All Formal Context* gracias al cual podemos volver a visualizar la tabla completa.

3. Por último, si volvemos a obtener los *conceptos*, estos se vuelven a generar a partir de las filas seleccionadas (figura 21).

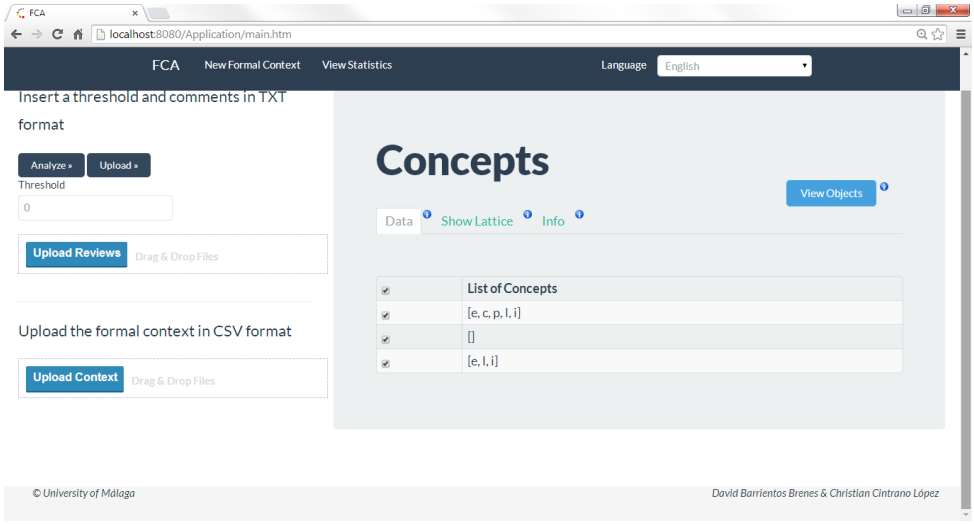

Figura 21: Nuevos *conceptos* a partir de los usuarios seleccionados

11. Experimentos

A continuación se presentarán los experimentos realizados gracias al sistema construido. Se han realizado 3 experimentos. El primero (sección 11.1) es una comparativa sobre hoteles de las provincia de Málaga. El experimento número 2 (sección 11.2) busca obtener aspectos comunes de los hoteles de la Costa del Sol. Y el experimento 3 (sección 11.3) es igual que el anterior pero aplicado al sector de la restauración.

11.1. Experimento: Hoteles de la Costa del Sol

En este experimento se analizarán cuatro hoteles elegidos al azar en la provincia de Málaga. La finalidad del mismo será obtener una comparativa entre hoteles de distinto tipo.

11.1.1. Hotel Molina Lario (Málaga)

Caso de prueba Observar cuáles son las características más relevantes de este hotel según los usuarios.

Datos de entrada y consideraciones iniciales Como entrada del sistema se seleccionaron 10 comentarios de Tripadvisor. Para la selección de los comentarios se tuvieron en cuenta las siguientes consideraciones:

- Los usuarios que publicaron esos comentarios debían de tener al menos, alrededor de 50 reviews. Esta medida fue tomada para minimizar el spam y obtener datos más consolidados.
- Las publicaciones se recogen en inglés para facilitar el análisis y, por lo tanto, el perfil turista es extranjero.
- Se eligió un umbral que nos permitiera obtener alrededor de 10 atributos aproximadamente.
- De cada entrada se recogió solo la siguiente información: titulo, comentario, fecha de realización y modo de viaje (solo, en pareja, familiar, etc.).

Resultados obtenidos Tras la ejecución del sistema se obtuvieron un total de 8 atributos para un umbral de 2:

breakfast, rooftop pool, city, airport, rooms, malaga, cathedral y *hotel*

Para ver el *contexto formal* vease el anexo C.3.

Y se obtuvieron los siguientes conceptos:

- {airport, cathedral, rooms}
- {hotel, city}
- {airport, breakfast, cathedral, rooftop pool, rooms}
- {breakfast, city}
- {airport, rooms}
- {cathedral, rooms, city}
- {breakfast, city, rooms}
- {malaga}
- {hotel, rooms}
- {hotel, city, rooms}
- {rooftop pool, rooms}
- {hotel}
- {cathedral, rooms}
- {rooftop pool, rooms, city}
- {}
- {cathedral}
- {malaga, rooms}
- {city, rooms}
- {city}
- {airport, malaga, breakfast, cathedral, hotel, rooftop pool, city, rooms}
- {airport, malaga, hotel, rooms}
- {breakfast}
- {breakfast, rooms}
- {breakfast, hotel, city}
- {airport, cathedral, rooms, city}
- {breakfast, cathedral, rooftop pool, rooms, city}
- {malaga, breakfast, cathedral, rooftop pool, rooms, city}
- {rooms}
- {hotel, rooftop pool, city, rooms}
- {breakfast, cathedral, rooftop pool, rooms}

La figura 22 muestra el *retículo de conceptos*.

Análisis de los resultados El principal hecho obtenido es el más obvio y es que, para los usuarios, lo más importante es la habitación, ya que si descartamos los *conceptos* de la parte más baja del retículo por ser poco relevantes aparece en todos.

Otro aspecto destacable es que siempre que aparece el aeropuerto o la piscina en el tejado aparece la habitación. Estas son, desde el punto de vista del FCA, unas implicaciones entre los atributos. Además denotan aspectos menos importantes para los usuarios. Estas características son particulares de este hotel y por lo tanto los usuarios las comentan menos. Sin embargo, también se podrían ver como puntos donde enfocarse para una campaña de marketing diferenciador.

También se puede observar que la catedral (significa cercanía), la piscina y el desayuno son aspectos que se tienden a valorar por separado, es decir, que un turista que valora la

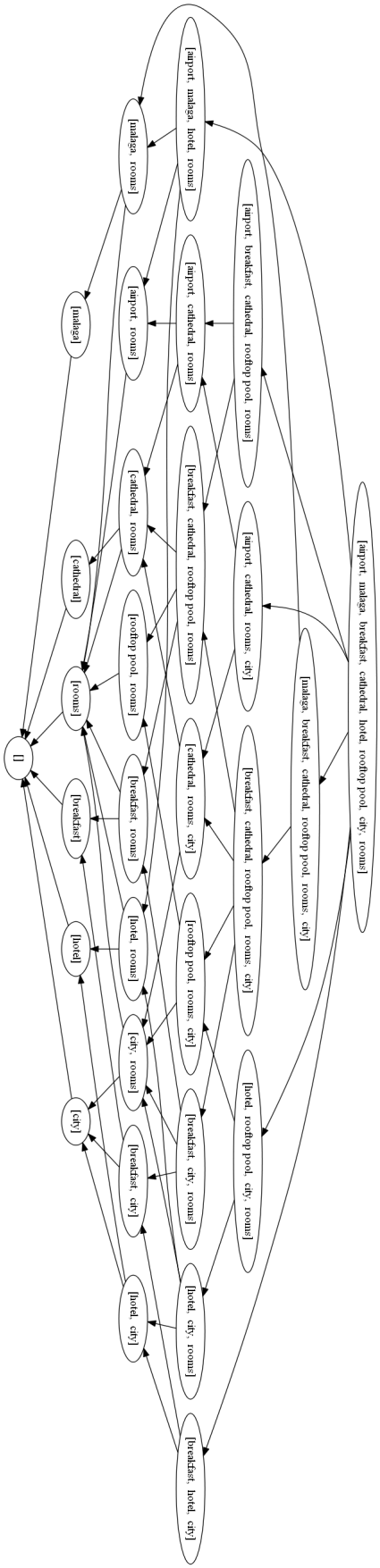

Figura 22: Retículo de Conceptos para el Hotel Molina Lario

piscina le da más igual la catedral. Esto nos permite descubrir grupos de usuarios.

11.1.2. Puente Romano Beach Resort (Marbella)

Caso de prueba Observar cuáles son las características más relevantes de este hotel según los usuarios.

Datos de entrada y consideraciones iniciales Como entrada del sistema se seleccionaron 10 comentarios de Tripadvisor. Para la selección de los comentarios se tuvieron en cuenta las siguientes consideraciones:

- Los usuarios que publicaron esos comentarios debían de tener al menos 20 publicaciones y alrededor de 50 reviews.
- Las publicaciones se recogen en inglés para facilitar el análisis y, por lo tanto, el perfil turista es extranjero.
- Se eligió un umbral que nos permitieran obtener alrededor de 10 atributos aproximadamente.
- De cada entrada se recogió solo la siguiente información: titulo, comentario, fecha de realización y modo de viaje (solo, en pareja, familiar, etc.).

Resultados obtenidos Tras la ejecución del sistema se obtuvieron un total de 8 atributos para un umbral de 2:

rooms, beach, family, staff, couple, restaurants, service y *hotel*

Para ver el *contexto formal* vease el anexo C.4.
Y se obtuvieron los siguientes conceptos:

- {service, restaurants}
- {staff, beach, hotel, restaurants, rooms}
- {couple, restaurants}
- {rooms}
- {staff, restaurants, rooms}
- {beach, restaurants}
- {family, restaurants}
- {hotel, rooms}
- {staff, couple, restaurants, rooms}
- {family, service, couple, beach, restaurants}
- {hotel}
- {service, couple, restaurants}
- {restaurants}

- {staff, family, service, couple, beach, hotel, restaurants, rooms}
- {staff, hotel, restaurants, rooms}
- {staff, service, couple, restaurants, rooms}
- {}
- {family}
- {family, hotel}

La figura 23 muestra el *retículo de conceptos*.

Análisis de los resultados El aspecto que más destaca de este hotel es que el restaurante tiene una mayor importancia que la habitación. Esto es un gran aspecto diferenciador con respecto a los demás hoteles. Además, al contrario de lo que cabía esperar, la playa posee un papel menos importante. Sin embargo se puede observar que unido al restaurante existen *conceptos* más interesantes:

- Los turistas que van acompañados de la pareja o la familia valoran más aspectos relativos al servicio.
- Mientras que para los que van solos o por trabajo tiene más importancia aspectos del hotel en sí como la habitación o el staff.

11.1.3. The Marbella Heights Boutique Hotel (Marbella)

Caso de prueba Observar cuáles son las características más relevantes de este hotel según los usuarios.

Datos de entrada y consideraciones iniciales Como entrada del sistema se seleccionaron 10 comentarios de Tripadvisor. Para la selección de los comentarios se tuvieron en cuenta las siguientes consideraciones:

- Los usuarios que publicaron esos comentarios debían de tener al menos 20 publicaciones y alrededor de 40 reviews.
- Las publicaciones se recogen en inglés para facilitar el análisis y, por lo tanto, el perfil turista es extranjero.
- Se eligió un umbral que nos permitieran obtener alrededor de 10 atributos aproximadamente.
- De cada entrada se recogió solo la siguiente información: titulo, comentario, fecha de realización y modo de viaje (solo, en pareja, familiar, etc.).

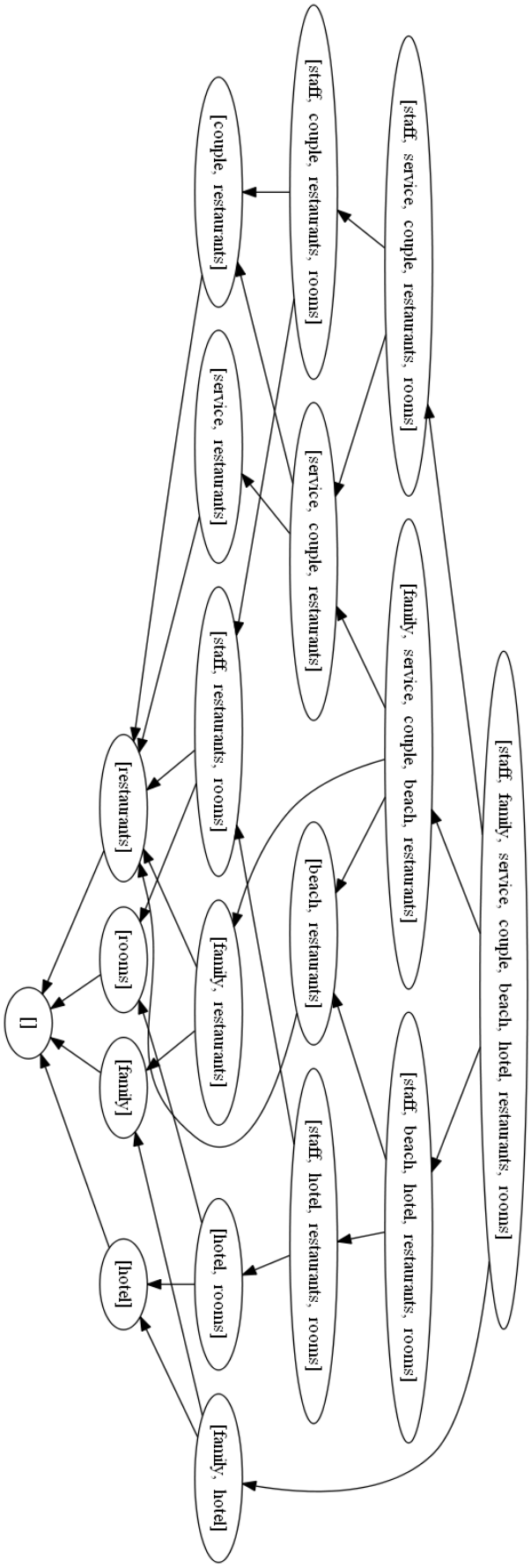

Figura 23: Retículo de Conceptos para el Hotel Puente Romano Beach Resort

Resultados obtenidos Tras la ejecución del sistema se obtuvieron un total de 10 atributos para un umbral de 2:

breakfast, mercedes, car, taxis, marbella heights, nights, marbella, dinner, pool y *couple*

Para ver el *contexto formal* vease el anexo C.5.
Y se obtuvieron los siguientes conceptos:

- {dinner, marbella}
- {pool, breakfast}
- {car, couple}
- {car, dinner, couple, mercedes}
- {couple}
- {marbella}
- {marbella heights}
- {breakfast, couple, mercedes}
- {couple, marbella}
- {mercedes}
- {dinner, car, pool, couple, marbella, taxis, mercedes}
- {breakfast, couple}
- {car, dinner, couple}
- {dinner}
- {car, breakfast, couple, mercedes}
- {dinner, car, breakfast, marbella heights, couple, nights, marbella, taxis}
- {}
- {breakfast, mercedes}
- {car, couple, mercedes}
- {couple, mercedes}
- {car, dinner, pool, breakfast, marbella heights, nights, couple, marbella, taxis, mercedes}
- {breakfast}
- {pool, mercedes}
- {car, dinner, couple, nights, marbella, taxis}
- {car, breakfast, couple, nights, marbella, taxis}
- {car, couple, nights, marbella, taxis, mercedes}
- {car, couple, nights, marbella, taxis}
- {pool}
- {car, dinner, marbella heights, breakfast, couple}
- {marbella heights, couple}
- {car, dinner, breakfast, marbella heights, couple, mercedes}
- {dinner, car, pool, couple, nights, marbella, taxis, mercedes}
- {dinner, car, couple, marbella}
- {car, couple, marbella}
- {car, breakfast, nights, couple, marbella, taxis, mercedes}
- {car, breakfast, couple}

La figura 24 muestra el *retículo de conceptos*.

Análisis de los resultados El primer aspecto destacable de este experimento es que las parejas tienden a valorar más los desayunos que las cenas. Esto contradice lo que se podría esperar, y por eso, este grupo de usuarios es un descubrimiento importante realizado por el sistema. Por otro lado la piscina tiene muy poca importancia mientras que el coche o el taxi aparecen en muchos *conceptos*. De esto se podría deducir que la mayoría de los usuarios prefiere pasar el tiempo fuera del hotel, por lo que necesitan un transporte para desplazarse.

Es curioso que aparece Mercedes, que es la dueña del hotel. Al tratarse de un hotel muy pequeño, se ve que los usuarios valoran la cercanía del trato.

11.1.4. Holiday World (Benalmadena)

Caso de prueba Observar cuáles son las características más relevantes de este complejo hotelero según los usuarios.

Datos de entrada y consideraciones iniciales Como entrada del sistema se seleccionaron 4 comentarios de Tripadvisor por cada uno de los hoteles que componen el complejo: Hotel Polynesia, Hotel Palace, Hotel Holiday Hydros y Hotel Village. Para la selección de los comentarios se tuvieron en cuenta las siguientes consideraciones:

- Los usuarios que publicaron esos comentarios debían de tener al menos 20 publicaciones y alrededor de 40 reviews.
- Las publicaciones se recogen en inglés para facilitar el análisis y, por lo tanto, el perfil turista es extranjero.
- Se eligió un umbral que nos permitieran obtener alrededor de 10 atributos aproximadamente.
- De cada entrada se recogió solo la siguiente información: titulo, comentario, fecha de realización y modo de viaje (solo, en pareja, familiar, etc.).

Resultados obtenidos Tras la ejecución del sistema se obtuvieron un total de 8 atributos para un umbral de 3:

family, staff, hotel, floor, kids, rooms, adults y *food*

Sin embargo el número de *conceptos* generado era demasiado elevado para su análisis por lo que se tomó la decisión de aumentar el umbral a 4 con los que se obtuvieron los siguientes atributos:

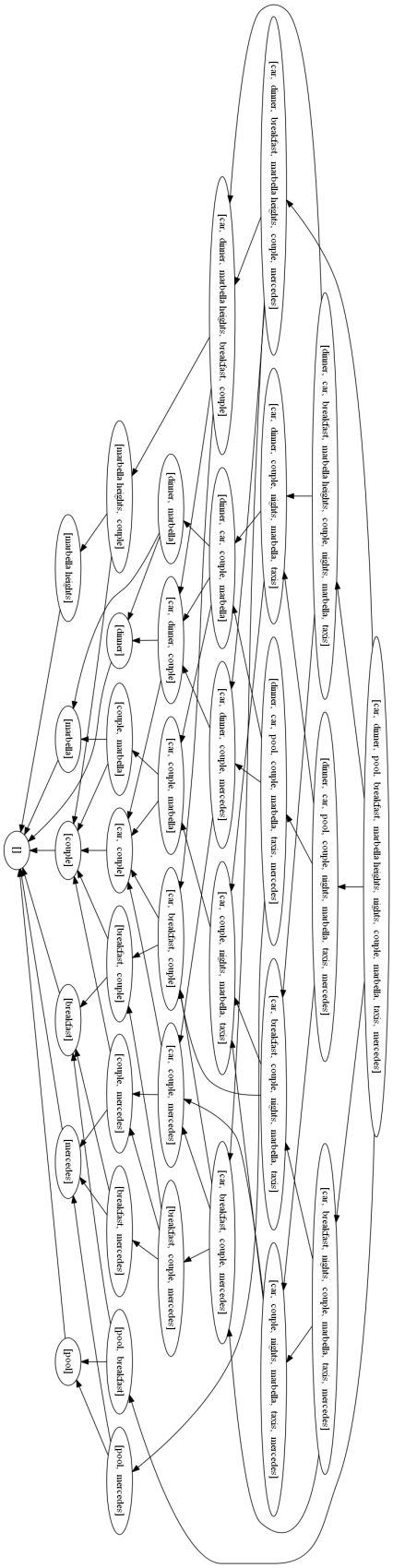

Figura 24: Retículo de Conceptos para el Hotel Marbella Heights Boutique

family, staff, hotel, kids, rooms y *food*

Para ver el *contexto formal* de ambos casos vease los anexos C.6 y C.7.
Se obtuvieron los siguientes conceptos para el umbral de 4:

- {family, food, rooms}
- {family, rooms}
- {staff, hotel}
- {food, kids}
- {kids}
- {staff, food, hotel, rooms}
- {family, kids}
- {staff}
- {staff, hotel, rooms}
- {family, hotel, rooms}
- {staff, food, hotel, kids, rooms}
- {staff, food, hotel}
- {staff, family, hotel, rooms}
- {hotel, rooms}
- {food}
- {staff, food, family, hotel, kids, rooms}
- {hotel}
- {staff, food}
- {food, family, kids}
- {food, kids, rooms}
- {}
- {food, hotel, rooms}
- {food, hotel}
- {food, family, hotel, rooms}
- {rooms}
- {staff, family, food, rooms}
- {food, rooms}
- {staff, family, rooms}
- {staff, food, rooms}
- {family}
- {food, family}
- {staff, rooms}

La figura 25 muestra el *retículo de conceptos*.

Análisis de los resultados Al ser un complejo turístico para toda la familia destacan los atributos *family* y *kids*. Los niños parece que es el atributo con mayor importancia, con lo que se puede interpretar que, para las familias, el hecho más importante es que sus hijos disfruten.

Además se puede observar que los atributos están repartidos uniformemente entre los *conceptos*. De esto se deduce que los valores más importantes para las familias son la comida, la habitación y el servicio.

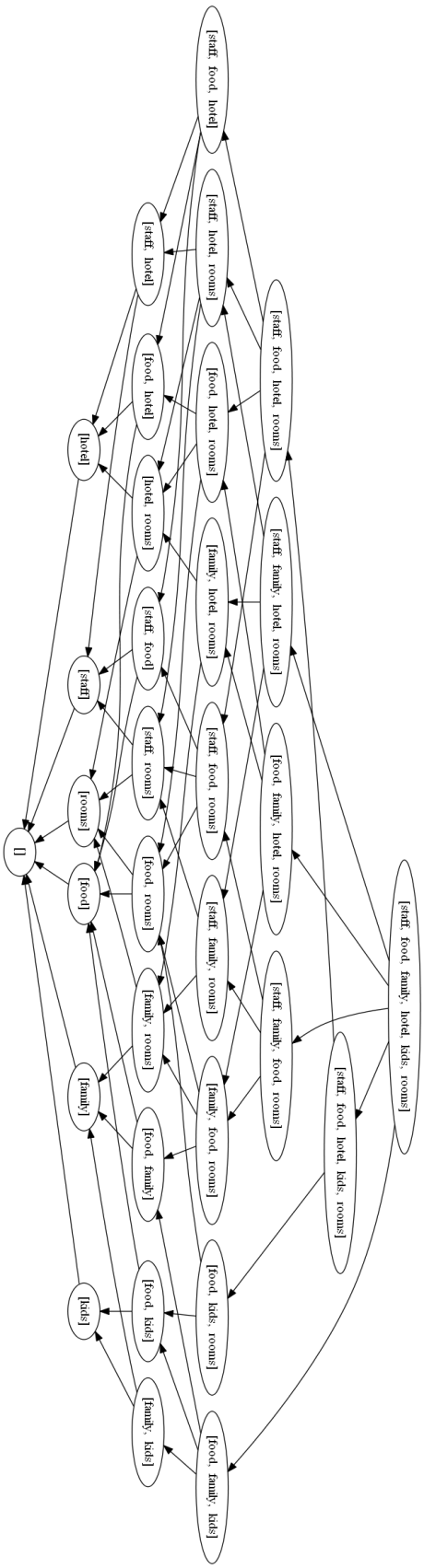

Figura 25: Retículo de Conceptos para el complejo hotelero Holiday World con un umbral de 4

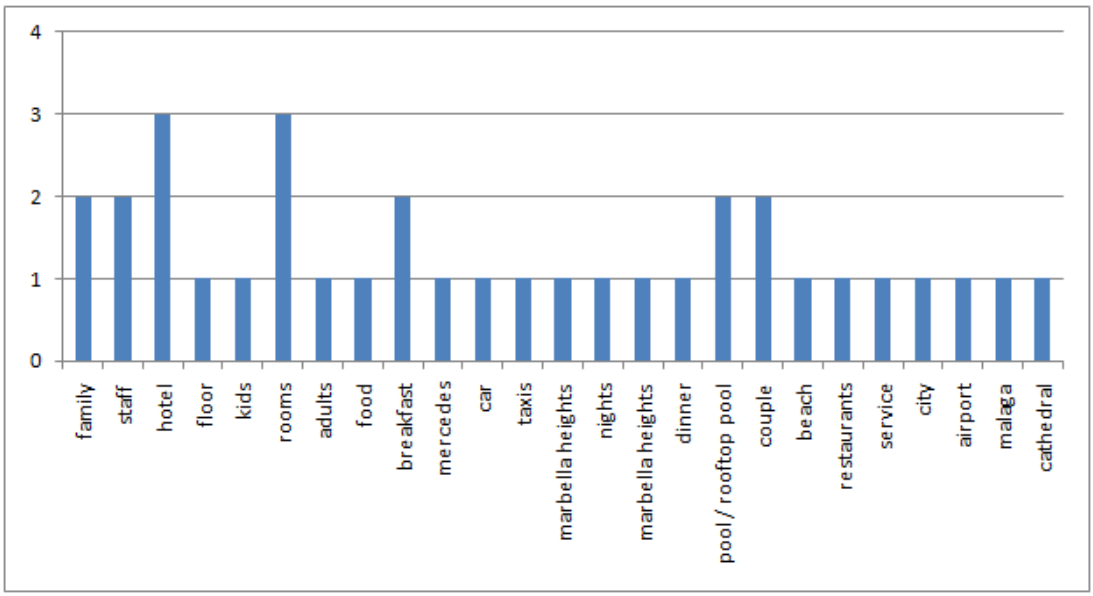

Figura 26: Comparativa de ocurrencias de los atributos

11.1.5. Comparativa de los resultados

En estos experimentos se ha podido comprobar cómo los atributos de cada hotel pueden ser diferentes entre sí dependiendo del tipo de público al que se oriente. Sin embargo, existe un conjunto de características que, en general, son muy importantes como son las habitaciones, el staff, los desayunos y el hotel. Estos son aspectos imprescindibles que todo sistema de recomendación de hoteles debería considerar. Además de los datos también se deduce que los turistas que visitan la Costa del Sol suelen hacerlo con su familia o pareja. Esto provoca una división demográfica muy importante de cara a la realización de campañas de marketing.

Finalmente, en la figura 26 se muestra el numero de apariciones de los atributos presentados en estos casos de estudio.

11.2. Experimento: Características de los hoteles de la Costa del Sol

Caso de prueba Este experimento busca obtener las características comunes de los hoteles de la Costa del Sol. De este modo se podrá obtener una categorización inicial de los intereses comunes en este tipo de establecimientos.

Datos de entrada y consideraciones iniciales Como entrada del sistema se seleccionaron 12 comentarios de Tripadvisor sobre 12 hoteles de toda la Costa del Sol. Para la selección de cada uno de los comentarios se tuvieron en cuenta las siguientes consideraciones:

- Los usuarios que publicaron esos comentarios debían de tener al menos, alrededor de 100 reviews. Esta medida fue tomada para minimizar el spam y obtener datos más consolidados.
- Las publicaciones se recogen en inglés para facilitar el análisis y, por lo tanto, el perfil turista es extranjero.
- Se eligió un umbral que nos permitieran obtener unos 10 atributos aproximadamente.
- De cada entrada se recogió solo la siguiente información: titulo, comentario, fecha de realización y modo de viaje (solo, en pareja, familiar, etc.).

Resultados obtenidos Tras la ejecución del sistema se obtuvieron un total de 9 atributos:

room, parking, rooms, beach, staff, couple, night, hotel y *things*

Para un umbral de 2.

Tras realizar una unión de los sinónimos *room* y *rooms*. La lista resultante de atributos fue la siguiente:

room, parking, beach, staff, couple, night, hotel y *things*

Para ver el *contexto formal* vease el anexo C.8

Y se obtuvieron los siguientes conceptos:

- {ROOM, staff, couple, beach, hotel}
- {ROOM}
- {couple}
- {ROOM, parking, things}
- {ROOM, couple, beach, hotel}
- {ROOM, staff}
- {ROOM, couple, beach}
- {ROOM, couple}
- {ROOM, parking}
- {ROOM, couple, night, hotel, beach}
- {ROOM, couple, hotel}
- {couple, hotel}
- {things}
- {ROOM, parking, couple, night}
- {ROOM, couple, night}
- {}
- {ROOM, parking, beach, things}
- {beach}
- {ROOM, parking, staff, couple, things}
- {ROOM, parking, couple}

- {ROOM, parking, staff, couple, night, beach, hotel, things}
- {ROOM, night}
- {ROOM, beach}
- {ROOM, staff, couple}
- {couple, beach}

La figura 27 muestra el *retículo de conceptos*.

Análisis de los resultados Primero se debe aclarar el significado del atributo *things*. Este se refiere al conjunto de actividades o entretenimiento extra que se ofrece en el hotel o en sus alrededores.

Se puede observar que, como en anteriores experimentos, la habitación es el atributo más importante para los turistas.

Un *concepto* especialmente destacable es el de {room, couple, night}. Representa a las parejas que salen por la noche. Son un grupo muy importante en la Costa del Sol ya que suelen consumir bastante, por ejemplo al salir de fiesta. Además para las parejas también tiene mucha importancia la playa. Sin embargo parece que existe una separación entre las parejas nocturnas con las playeras. Esto se podría interpretar que al ser actividades de horarios opuestos los turistas se deben de decantar por solo una de las dos actividades.

Por otro lado, la playa y el staff tienden a estar por separado. Esto indica que las personas que vienen por las playas de la Costa del Sol tienden a pasar poco tiempo en el hotel, por lo que no les importa la calidad del personal.

El atributo *things* aparece en varios *conceptos* junto a el parking o la playa. Esto nos muestra que los usuarios a los que les gusta hacer turismo salen mucho del hotel, por lo que el parking es importante. Sin embargo para las parejas es más relevante el staff por lo que parece ser que pasan más tiempo en el hotel.

Realizando una comparación entre los atributos obtenidos en este experimento con los presentados en la sección 11.1.5 se puede observar que, como cabía esperar la mayoría de los atributos son similares, demostrando el correcto funcionamiento del sistema.

Como conclusión final se ha podido ilustrar como los principales atractivos de la Costa del Sol, sus playas y actividades/recursos turísticos, son de los aspectos más valorados por los usuarios.

11.3. Experimento: Restaurantes de la Costa del Sol

Este experimento busca obtener las características comunes del sector de la restauración en la Costa del Sol. De este modo se podrá obtener una categorización inicial de los intereses comunes en este tipo de establecimientos.

Datos de entrada y consideraciones iniciales Como entrada del sistema se seleccionaron 10 comentarios de Tripadvisor sobre 10 restaurantes de toda la Costa del Sol.

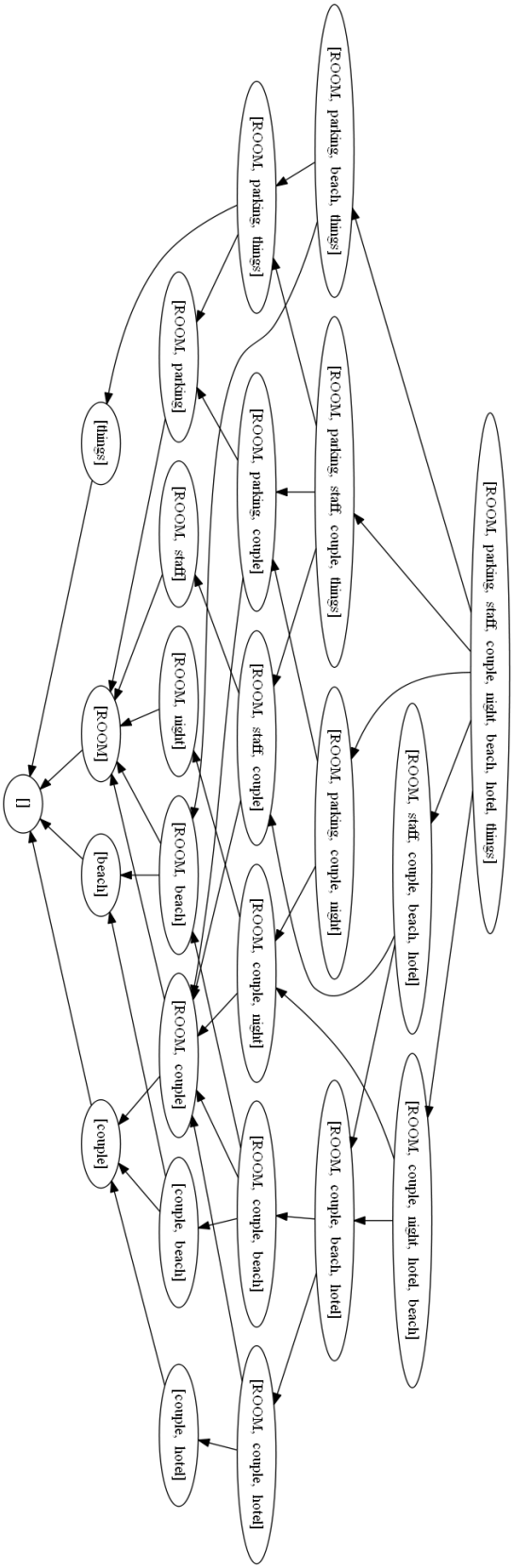

Figura 27: Retículo de Conceptos

Para la selección de cada uno de los comentarios se tuvieron en cuenta las siguientes consideraciones:

- Los usuarios que publicaron esos comentarios debían de tener al menos 55 publicaciones, y la media de todas las reviews debía de ser unas 90. Esta medida fue tomada para minimizar el spam y obtener datos más consolidados.
- Las publicaciones se recogen en inglés para facilitar el análisis y, por lo tanto, el perfil turista es extranjero.
- Se eligió un umbral que nos permitieran obtener unos 10 atributos aproximadamente.
- De cada entrada se recogió solo la siguiente información: titulo, comentario, fecha de realización y modo de viaje (solo, en pareja, familiar, etc.).

Resultados obtenidos Tras la ejecución del sistema se obtuvieron un total de 7 atributos para un umbral de 1:

visit, time, wine, food, restaurant, notch y *service*

Para ver el *contexto formal* vease el anexo C.9
Y se obtuvieron los siguientes conceptos:

- {time}
- {wine}
- {food, service, visit, restaurant}
- {food, service, visit}
- {wine, time, food, service, visit, restaurant, notch}
- {wine, time, food, service, visit}
- {service, restaurant}
- {}
- {service}
- {notch}

La figura 28 muestra el *retículo de conceptos*.

Análisis de los resultados Hay varios atributos que son necesarios definir para poder realizar una correcta interpretación.

- *notch* se refiere a la categoría del restaurante (top notch = primera categoría)
- *time* es un atributo muy general que puede expresar el tiempo que el camarero pasa en la mesa.
- *visit* simplemente hace referencia a la visita por lo que no es relevante para este análisis.

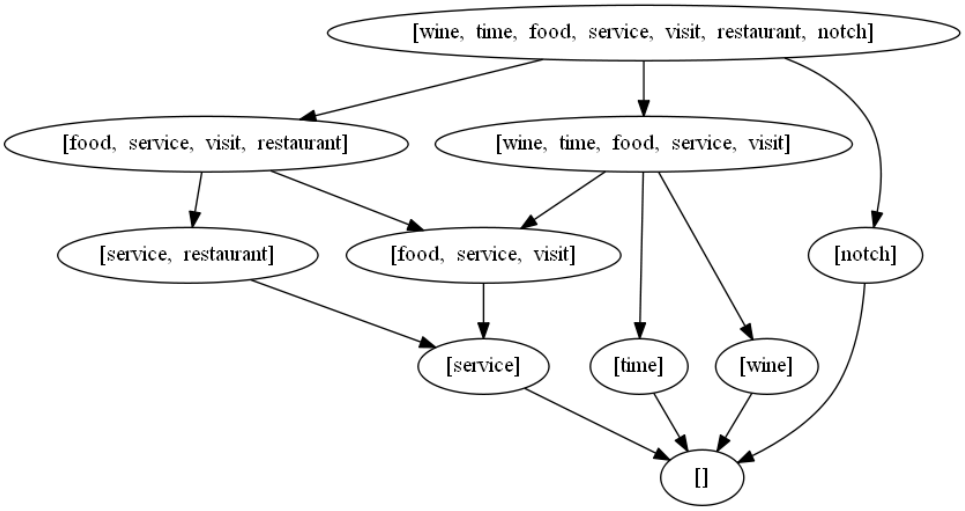

Figura 28: Retículo de Conceptos de los restaurantes de la Costa del Sol

Tras estas consideraciones pasamos al análisis de los *conceptos*.

El primer hecho que llama la atención es el reducido número de estos con respecto a otros experimentos. Esto puede ser por dos motivos:

- La amplia variedad de tipos de restaurantes, lo cual dificulta la obtención de características comunes entre todos ellos.
- Los comentarios sobre restaurantes suelen ser más cortos. Esto provoca que se obtengan menos *keywords* de los textos y, por lo tanto, un menor número de atributos.

Por otro lado está el *concepto {food, service, visit}* y *{wine, time, food, service, visit}*. Esto nos establece una relación entre los usuarios que valoran *{food, service, visit}* y los de *wine*. Con esto se indica que a muchos usuarios a los que les importa la comida y el servicio, también les importa el vino (alcohol en general). Sin embargo existe una separación entre los que valoran las bebidas alcohólicas y los que dan un mayor peso al establecimiento (*restaurant*).

Por último, el atributo *notch* está separado del resto lo que nos indica que existe un grupo de usuarios a los que solamente les interesa la categoría del restaurante, sin importar, lo buena o mala que sea la comida, entre otros aspectos.

12. Conclusiones y trabajos futuros

En este TFG se ha desarrollado un sistema que implementa algunas técnicas innovadoras que pueden ser usadas en sistemas de recomendación. El campo de estudio de los sistemas de recomendación es muy amplio y abarca técnicas muy diferentes entre sí. En esta memoria se ha tratado de exponer de manera resumida la necesidad de estos sistemas y, en lugar de exponer un gran número de teorías, se ha profundizado en la teoría del *análisis formal de conceptos* y su utilidad para este tipo de sistemas.

En lo que respecta al estudio del FCA, fue una grata sorpresa al descubrir la simplicidad de la teoría. Otros aspectos destacables son: su aplicabilidad en múltiples campos de estudio y la cantidad de algoritmos que trabajan sobre él.

Esta simplicidad se refleja en la facilidad con la que se puede realizar una implementación en prácticamente cualquier lenguaje de programación, de una manera muy directa e intuitiva.

Además de presentar el estado del arte de los sistemas de recomendación se realizó una aplicación que es capaz de realizar las operaciones básicas que propone la teoría de FCA.

La implementación de este sistema se realizó en varias etapas. Se debe hacer una especial mención a la primera etapa que fue la de análisis de los textos. Este es un problema muy complejo en el que entran aspectos de análisis de sentimientos y minería de datos y, por tanto, ha sido la parte de mayor dificultad del TFG. El resto del desarrollo siguió una metodología incremental sin grandes contratiempos.

El sistema final obtenido ha sido bastante satisfactorio, cumpliéndose todos los requisitos. Sin embargo, se ha llegado a la conclusión de que, en el estado actual del desarrollo, para poder obtener un verdadero conocimiento sobre los resultados que el sistema presenta se necesita a un experto que sea capaz de interpretar correctamente estos datos. De esta manera el sistema se presenta como un sistema de ayuda a la toma de decisiones (DSS) con técnicas de social CRM.

Finalmente se presentan una serie de propuestas para futuras líneas de TFGs relacionadas con el trabajo descrito en esta memoria:

- Aplicar técnicas de sinónimos y análisis de sentimientos a los comentarios.
- Emplear Twitter como fuente de comentarios, empleando para ello algún desarrollo o herramienta que automatice la obtención de tweets.
- Ampliar este desarrollo con técnicas de big data. Se puede realizar conjuntamente a la propuesta anterior.
- Desarrollar un sistema completo de recomendaciones empleando este desarrollo como un subsistema.

Anexos

A. Temporización del trabajo

A continuación se expone un desglose del tiempo empleado en cada tarea del TFG:

Tarea	**Duración (en horas)**
Planificación y seguimiento	10
Diseño	5
Estudio y Aprendizaje	142
Desarrollo Beta de la aplicación	30
Desarrollo Final del sistema	57
Problemas y mejoras	35
Testing	12
Desarrollo de la memoria	75
Total	366

B. Herramientas para FCA

Lista de otras herramientas para trabajar con *contextos formales* y *conceptos* o basadas en FCA.

Tockit Software para trabajar con FCA [48].

ToscanaJ Plugin para FCA [49].

Docco Es un sistema de gestión de documentos que emplea el motor de búsqueda de Lucene, el cual es mejorado mediante técnicas de FCA [12]

Concept Explorer Permite dibujar el *retículo de conceptos* pero no da los *conceptos* por separado [9].

Galicia Pretende ser una plataforma de software integrado que incluye componentes para las operaciones con retículos que podrían ser necesarias en aplicaciones prácticas o en estudios teóricos [22].

FcaStone Software de conversión de formato y la generación de retículos en línea de comandos [19]

Camelis Sistema de información lógica basado en FCA [7].

Colibri Concepts Permite descargar el código desde subversión [8].

Contextual Role Editor [10]

Coron Software sobre data mining para Unix [11].

Erca Framework para FCA [14].

FCA algorithms (FCALGS) [15]

FCA Bedrock Herramienta para la creación de *contextos formales* a partir de archivos .csv [16].

Griff Software de FCA escrito en Ruby [51].

LaTeX for FCA Estilos para ayudar a crear ficheros LaTeXsobre FCA [36].

Lattice Miner Panel gráfico muy sencillo para dibujar *retículos de conceptos*. Emplea ficheros de Galicia como *contexto formal* [37].

OpenFCA Programa escrito en C#/.NET para visualizar retículos. Posee videos tutoriales [42].

FCA Extension for Excel Extensión para Microsoft Excel que nos permite trabajar con *contextos formales* y visualizar *retículos de conceptos*. Muy sencillo de usar y con vídeos explicativos [17].

FcaView Tab Plugin Extensión para Protégé [20].

OntoComP Otra extensión para Protégé [41].

FcaLib No es exactamente un software para trabajar con FCA, sino un plugin para Eclipse que ayuda a la refactorización de código basándose en la teoría del *retículo de conceptos* [18].

Sage Software open-souce para matemáticas. Existen paquetes específicos para trabajar con FCA [44].

NetworkX Paquete escrito en Python para la creación, manipulación y estudio de la estructura, la dinámica y las funciones de redes complejas. Es integrable con Sage y ofrece paquetes para FCA [40].

JaLaBa Web para la obtención del *retículo de conceptos*. Sencillo e intuitivo, pero poco eficiente para *contextos* grandes [27].

GaloisExplorer Aplicación de escritorio para visualizar *conceptos*. Muestra el retículo como esferas en 3D. Requiere el *contexto formal* como un archivo .tlb [23].

C. Contextos Formales

C.1. MUSHROOM.csv

e	p	c	l	i
1	0	1	0	0
1	0	1	0	1
1	0	0	1	1
1	0	0	1	1
1	0	1	0	1
1	0	1	0	0
0	1	0	1	1
0	1	0	1	1
0	1	0	1	1
0	1	0	1	0

C.2. MUSHROOMusers.csv

USERS	e	p	c	l	i
u1	1	0	1	0	0
u2	1	0	1	0	1
u3	1	0	0	1	1
u4	1	0	0	1	1
u5	1	0	1	0	1
u6	1	0	1	0	0
u7	0	1	0	1	1
u8	0	1	0	1	1
u9	0	1	0	1	1
u10	0	1	0	1	0

C.3. hotel-molina-lario.csv

USERS	breakfast	rooftop pool	city	airport	rooms	malaga	cathedral	hotel
cbouknight	0	0	0	1	1	1	0	1
gaserati	1	0	1	0	0	0	0	1
rainbowfan	0	0	0	0	0	1	0	0
TravelerMike Plano	1	1	1	0	1	1	1	0
Troy B	0	0	0	0	0	0	0	0
Cestrian56	1	1	0	1	1	0	1	0
ViviDublin	0	1	1	0	1	0	0	1
Ztraveladdict	0	0	0	0	0	0	1	0
IntrepidHector	1	0	0	0	1	0	0	0
Mike L	0	0	1	1	1	0	1	0

C.4. puente-romano-beach-resort.csv

USERS	rooms	beach	family	staff	couple	restaurants	service	hotel
Marcus K	1	0	0	1	1	1	0	0
Maskita	0	0	1	0	0	0	0	0
mini_moi75	0	1	1	0	1	1	1	0
Agkarajit	0	1	0	0	0	1	0	0
Ctmom90	0	0	1	0	0	0	0	1
john-hana	1	0	0	1	1	1	1	0
Alex-124	0	0	0	0	0	1	1	0
Sagittarius_7	1	0	0	0	0	0	0	1
Sara A	0	0	0	0	0	0	0	1
Oliver_Zeitoun	1	1	0	1	0	1	0	1

C.5. marbella-heights-boutique.csv

USERS	breakfast	mercedes	car	taxis	marbella heights	nights	marbella	dinner	pool	couple
JJReid1030	0	0	0	0	1	0	0	0	0	0
Lynnemul	1	1	0	0	0	0	0	0	0	0
Caroline397	1	1	1	1	0	1	1	0	0	1
DD17DD	0	1	0	0	0	0	0	0	1	0
Stephersan	1	0	0	0	0	0	0	0	1	0
travelsar0und	1	0	1	1	1	1	1	1	0	1
mill501	0	1	1	1	0	1	1	1	1	1
karl45	0	0	0	0	0	0	1	1	0	0
MarinaCrijns	0	0	0	0	1	0	0	0	0	1
LizzyMEB	1	1	1	0	1	0	0	1	0	1

C.6. holiday-world-th3.csv

USERS	family	staff	hotel	floor	kids	rooms	adults	food
weeclairey	1	0	0	0	1	0	0	0
Maureen M	0	0	0	0	0	0	0	0
carol d	0	0	1	0	0	1	0	1
Gemma H	0	1	0	0	0	1	1	1
darncar	0	1	1	1	1	1	1	1
storm082	1	0	0	1	0	0	0	1
Dominic M	1	1	1	0	0	1	0	0
RussCityboy	1	1	0	0	0	1	1	1
debbie_clarke61	1	0	1	0	0	1	0	1
eaglecreek	0	0	0	0	0	0	0	0
ali24west	0	1	1	0	0	0	0	1
MartinDNorwich	1	0	0	1	0	1	0	0
axel07	0	0	1	1	0	0	0	0
EnglandNorfolk	1	0	0	0	1	0	1	1
KennybOrmiston	1	0	0	0	1	0	0	0
Yorkshirelass024	0	0	0	0	1	1	0	1

C.7. holiday-world-th4.csv

USERS	family	staff	hotel	kids	rooms	food
carol d	0	0	1	0	1	1
Gemma H	0	1	0	0	1	1
darncar	0	1	1	1	1	1
storm082	1	0	0	0	0	1
Dominic M	1	1	1	0	1	0
debbie_clarke61	1	0	1	0	1	1
RussCityboy	1	1	0	0	1	1
ali24west	0	1	1	0	0	1
eaglecreek	0	0	0	0	0	0
MartinDNorwich	1	0	0	0	1	0
axel07	0	0	1	0	0	0
EnglandNorfolk	1	0	0	1	0	1
KennybOrmiston	1	0	0	1	0	0
Yorkshirelass024	0	0	0	1	1	1
Maureen M	0	0	0	0	0	0
weeclairey	1	0	0	1	0	0

C.8. hoteles-costa-del-sol.csv

HOTELS	room	parking	rooms	beach	staff	couple	night	hotel	things
El Oceano Beach Hotel	0	0	1	1	1	1	0	1	0
Hotel Puerta del Mar	1	0	0	0	0	0	1	0	0
Hotel La Fonda	1	0	0	0	0	1	0	1	0
Hotel La Luna Blanca	0	1	1	1	0	0	0	0	1
Marriott's Marbella Beach Resort	0	1	1	0	0	1	1	0	0
Marbella Club Hotel	0	0	0	1	0	1	0	0	0
NH Malaga	0	0	1	0	1	0	0	0	0
Malaga Hotel Picasso	0	0	1	1	0	1	1	1	0
Marriott's Playa Andaluza	0	0	0	0	0	0	0	0	1
Gran Hotel Elba Estepona and Thalasso Spa	0	0	0	0	0	1	0	1	0
The Marbella Heights Boutique Hotel	0	1	1	0	1	1	0	0	1
Hotel Casa Rosa	1	0	0	0	0	1	0	0	0

C.9. restaurantes.csv

RESTAURANTS	visit	time	wine	food	restaurant	notch	service
Lotus-La Malaguena	0	1	0	0	0	0	0
Meson de Calahonda	1	1	1	1	0	0	1
Los Barriles	0	0	0	0	0	0	0
Gelateria Di Porto Marina	0	0	0	0	0	0	0
Serafina	0	0	0	0	0	1	0
El Gato Lounge	0	0	0	0	0	0	0
Restaurante Palangreros	1	0	0	1	1	0	1
Pancho Villa	0	0	0	0	1	0	1
El Muelle de Arriate	0	0	1	0	0	0	0
El Meson de Cervantes	0	0	0	0	0	1	0

Bibliografía

[1] AJAX. http://es.wikipedia.org/wiki/AJAX.

[2] AlchemyAPI. http://www.alchemyapi.com/.

[3] Analyzing Twitter Data with Apache Hadoop, part 2: Gathering Data with Flume. http://blog.cloudera.com/blog/2012/10/analyzing-twitter-data-with-hadoop-part-2-gathering-data-with-flume/.

[4] Apache Ant. http://ant.apache.org/.

[5] Apache Flume. http://flume.apache.org/FlumeDeveloperGuide.html.

[6] Bootstrap. getbootstrap.com.

[7] Camelis. http://www.irisa.fr/LIS/ferre/camelis/.

[8] Colibri Concepts. https://code.google.com/p/colibri-concepts/.

[9] Concept Explorer. http://sourceforge.net/projects/conexp/.

[10] Contextual Role Editor. http://homepage.univie.ac.at/henri.muehle/misc.php#core.

[11] Coron system. http://coron.loria.fr/site/index.php.

[12] Docco. http://tockit.sourceforge.net/docco/index.html.

[13] Eclipse. https://www.eclipse.org/.

[14] Erca. https://code.google.com/p/erca/.

[15] FCA algorithms (FCALGS). https://code.google.com/p/erca/.

[16] Fca bedrock. http://sourceforge.net/projects/fcabedrock/.

[17] Fca Extension for Excel. http://www.fca.radvansky.net/news.php.

[18] FcaLib. https://code.google.com/p/fcalib/.

[19] FcaStone. http://fcastone.sourceforge.net/.

[20] FcaView Tab Plugin. http://info.med.hokudai.ac.jp/fca/fcaviewtab/fcaviewtab.html.

[21] Flatly (plantilla de Bootstrap). http://bootswatch.com/2/.

[22] Galicia. http://www.iro.umontreal.ca/~galicia/.

[23] GaloisExplorer. http://sourceforge.net/projects/galoisexplorer/.

[24] Gate. https://gate.ac.uk/.

[25] Graphviz. http://www.graphviz.org/.

[26] Hadoop. http://hadoop.apache.org/.

[27] JaLaBa. http://maarten.janssenweb.net/jalaba/JaLaBA.pl.

[28] JBoss. http://www.jboss.org/.

[29] JGraphT. http://jgrapht.org/.

[30] jQuery. http://jquery.com/.

[31] jQuery BlockUI. http://malsup.com/jquery/block/.

[32] jQuery File Download. http://jqueryfiledownload.apphb.com/.

[33] jQuery jGrowl. http://plugins.jquery.com/jgrowl/.

[34] jQuery Upload File. http://plugins.jquery.com/uploadfile/.

[35] jQuery Validate. http://plugins.jquery.com/validate/.

[36] LaTeX for FCA. http://www.math.tu-dresden.de/~ganter/fca/.

[37] Lattice Miner. http://sourceforge.net/projects/lattice-miner/.

[38] Maven. maven.apache.org/.

[39] Natural Language Toolkit. http://www.nltk.org/.

[40] NetworkX. http://networkx.github.io/.

[41] OntoComP. https://code.google.com/p/ontocomp/.

[42] OpenFCA. https://code.google.com/p/openfca/.

[43] R. http://www.r-project.org/.

[44] Sage. http://www.sagemath.org/.

[45] Scrum. https://www.scrum.org/.

[46] SentiStrength. http://sentistrength.wlv.ac.uk/.

[47] Spring Framework. http://projects.spring.io/spring-framework/.

[48] Tockit. http://tockit.sourceforge.net/tockit/index.html.

[49] ToscanaJ. http://tockit.sourceforge.net/toscanaj/index.html.

[50] Web del grupo SICUMA. http://www.sicuma.uma.es/es/.

[51] R. J. Cole. Griff. http://sourceforge.net/projects/griff/.

[52] Pablo Cordero, Manuel Enciso, Angel Mora, and Manuel Ojeda-Aciego. Computing minimal generators from implications: a logic-guided approach. In Laszlo Szathmary and Uta Priss, editors, *CLA*, volume 972 of *CEUR Workshop Proceedings*, pages 187–198. CEUR-WS.org, 2012.

[53] Huaiguo Fu and Engelbert Mephu Nguifo. A lattice algorithm for data mining.

[54] Bernhard Ganter. Two basic algorithms in concept analysis. In Léonard Kwuida and Baris Sertkaya, editors, *ICFCA*, volume 5986 of *Lecture Notes in Computer Science*, pages 312–340. Springer, 2010.

[55] Bernhard Ganter and Gerd Stumme. Methods and applications in computer science. *Formal Concept Analysis*, 46:1–117, 2002-2003.

[56] Saeid Hosseini, Sayan Unankard, Xiaofang Zhou, and Shazia Wasim Sadiq. Location oriented phrase detection in microblogs. In Sourav S. Bhowmick, Curtis E. Dyreson, Christian S. Jensen, Mong-Li Lee, Agus Muliantara, and Bernhard Thalheim, editors, *DASFAA (1)*, volume 8421 of *Lecture Notes in Computer Science*, pages 495–509. Springer, 2014.

[57] José L. Leiva, Antonio Guevara, Carlos Rossi, and Andrés Aguayo. Realidad aumentada y sistemas de recomendación grupales. *Estudios y Perspectivas de Turismo*, 23:40–59, 2014.

[58] Bing Liu. Opinion mining. pages 1–7.

[59] Bing Liu. Sentiment analysis: A multi-faceted problem. *IEEE Intelligent Systems*, 2010.

[60] Bing Liu. Sentiment analysis and opinion mining. In *AAAI*, pages 1–198, 2011.

[61] Bing Liu and Lei Zhang. A survey of opinion mining and sentiment analysis. In Charu C. Aggarwal and ChengXiang Zhai, editors, *Mining Text Data*, pages 415–463. Springer, 2012.

[62] Uta Priss. Uta Priss. http://www.upriss.org.uk/fca/fcasoftware.html.

[63] G. Stumme, R. Taouil, Y. Bastide, N. Pasqier, and L. Lakhal. Computing iceberg concept lattices with Titanic. *J. Data and KnowledgeEngineering (DKE)*, 42(2):189–222, 2002.