Plataforma en la nube para el análisis de patrones dentro del campus virtual.

Óscar Buenaposada Cano
Grado en Ingeniería Informática
Herramientas para el trabajo colaborativo

Fatos Xhafa
Atanasi Daradoumis Haralabus

15 de enero de 2017

B) GNU Free Documentation License (GNU FDL)

C) Copyright

FICHA DEL TRABAJO FINAL

Título del trabajo:	*Plataforma en la nube para el análisis de patrones dentro del campus virtual.*
Nombre del autor:	*Óscar Buenaposada Cano*
Nombre del consultor/a:	*Fatos Xhafa*
Nombre del PRA:	*Atanasi Daradoumis Haralabus*
Fecha de entrega (mm/aaaa):	01/2016
Titulación::	*Grado en Ingeniería Informática*
Área del Trabajo Final:	*Herramientas para el trabajo colaborativo*
Idioma del trabajo:	*Español*
Palabras clave	*Nube, minería de datos, software*

Resumen del Trabajo (máximo 250 palabras): *Con la finalidad, contexto de aplicación, metodología, resultados y conclusiones del trabajo.*

El presente **Trabajo Fin de Grado** (en adelante TFG) del **Grado en Ingeniería Informática** de la Universitat Oberta de Catalunya (en adelante UOC) se enmarca en el área de **Herramientas para el trabajo colaborativo** y tiene por objetivo especificar un proceso para analizar los patrones de navegación de los usuarios de la UOC en el Campus Virtual. Es decir, se basa en la minería de datos, explotación y estudio de datos registrados en diferentes formatos.

Concretamente, el estudio realizado en el marco de este TFG se centra en la explotación de datos de navegación de los usuarios del Campus de la UOC, después de una serie de procesos, exportarlos a un formato compatible para poder estudiarlos con la herramienta Weka (plataforma de software para el aprendizaje automático y la minería de datos).

Al ser un volumen de datos es elevado, nuestra línea de trabajo ira sobre el desarrollo y ejecución de software corporativo en un entorno fácil y de bajo coste (a priori), para poder afrontar un ratio de más de trece millones de registros al día. Por este motivo hay que optimizar bien el proceso y valorar su desarrollo en una estructura en la nube o cloud.

Como producto final tendremos una aplicación web que será capaz de generar la información final estructurada adecuada para ser procesada por una aplicación de minería de datos. En el caso del presente TFG, la aplicación seleccionada ha sido Weka que, gracias a los algoritmos que incluye, posibilita la obtención de los patrones de navegación buscados.

Abstract (in English, 250 words or less):

The present Final Degree Work (hereafter FDG) of the Degree in Computer Engineering of the Universitat Oberta de Catalunya (hereinafter UOC) is part of the Tools for Collaborative Work Area and aims to specify a process to analyze the navigation patterns of the users of the UOC in the Virtual Campus. That is, it is based on data mining, exploitation and study of data recorded in different formats.

Specifically, the study carried out within the framework of this TFG focuses on the exploitation of navigation data of users of the UOC Campus, after a series of processes; export them to a compatible format to be able to study them with the Weka tool (Software for automatic learning and data mining).

As a volume of data is high, our line of work will be about developing and executing corporate software in an easy and low-cost environment, to be able to face a ratio of more than thirteen million records per day. For this reason you have to optimize the process well and evaluate its development in a structure in the Cloud.

As final product we will have a web application that will be able to generate structured final information suitable to be processed by a data mining application. In the case of this TFG, the selected application has been Weka which, thanks to the algorithms included, makes it possible to obtain the navigation patterns sought.

Índice

Lista de figuras

1. Introducción.

1.1. Contexto y justificación del Trabajo.

El punto de partida del Trabajo Fin de Grado es la evolución del aplicativo ***Análisis de patrones de navegación de los estudiantes dentro del Campus UOC*** tomando como referencia datos reales de los logs del Campus Virtual de la UOC y posibilitando el análisis de los patrones de navegación de los estudiantes. Dicha evolución ha de poder solventar los problemas actuales de procesamiento, permitir crear mejores modelos predictivos así como un alto rendimiento en el acceso a los datos.

No perdemos de vista, además de la evolución tecnológica y de arquitectura, en especificar un proceso actualizado para analizar los patrones de navegación de los estudiantes de la UOC en el Campus Virtual. Es decir, se centra en la minería de datos, la explotación y el estudio de datos registrados en diferentes formatos.

Uno de los grandes problemas que presenta la minería de datos en su desarrollo es el **procesamiento de una enorme cantidad de datos** para obtener unos resultados lo más precisos y adecuados a la realidad, por lo que es habitual observar ciertas limitaciones de carácter tecnológico en dicho proceso.

Es necesario encontrar una tecnología y arquitectura que permita la resolución de los problemas detectados, sobre todo, económicos (debidos a la **escalabilidad necesaria**) pero también de **infraestructuras tecnológicas**. Es por ello que hemos apostado por el **Cloud Computing, en español Computación en la Nube**, ya que se basa principalmente en el uso de internet como una plataforma tecnológica a través de la cual acceder a un conjunto de servicios y aplicaciones alojados directamente en la red, en lugar de soportes físicos, como ha sido habitual.

Volviendo a la parte de proceso **de minería de datos**, el estudio realizado en el marco de este TFG se centra en la explotación de datos de navegación de los estudiantes del Campus de la UOC, después de una serie de procesos, exportarlos a un formato compatible para poder estudiarlos con la herramienta **Weka** (plataforma de software para el aprendizaje automático y la minería de datos)

En la mayor parte de los Servidores WEB, ciertas acciones realizadas por los usuarios en ellos, son registradas en los logs del servidor en ficheros planos de texto; el estudio del comportamiento de los usuarios a través de lo registrado en estos logs mediante herramientas avanzadas de minería de datos combinadas con la inteligencia artificial,

permite la identificación de **patrones de navegación**.

Tomará como referencia datos reales de los logs del Campus Virtual de la UOC y posibilitará el análisis de los patrones de navegación de los estudiantes.

1.2. Objetivos del Trabajo.

El objetivo principal es realizar una aplicación que procese los datos de navegación de los estudiantes de la UOC contenidos en los Logs del servidor Web Apache y los exporte a un formato compatible con herramientas de análisis (**Weka**) para analizar los patrones de navegación.

En paralelo descubriremos **Cloud Computing** como estrategia para el desarrollo de aplicaciones de tal manera que el usuario pueda acceder a capacidades de computación en la nube de manera automática a medida que las vaya requiriendo, sin necesidad de una interacción humana con su proveedor o sus proveedores de servicios Cloud, con servicios tales como tiempo de servidor y almacenamiento en red. En esa misma línea, veremos su posible uso como **repositorio de documentos**.

Una vez logrado el objetivo principal, en base a los patrones de navegación analizados, se han incorporado mejoras en cuanto a los datos a estudiar. Todos los posibles que permite la información contenida en los Logs de navegación.

1.3. Enfoque y método seguido.

Podemos decir que será en gran parte un producto nuevo ya que construiremos la aplicación de cero, conservando eso sí, las funcionalidades ya establecidas para el análisis de patrones del campus virtual.

Uno de los primeros puntos que nos encontramos es decidir qué tecnologías de desarrollo usaremos para la implementación de la aplicación. En ese sentido evolucionaremos la estructura .NET actual **hacia un código propietario abierto** ya que el consumo de recursos de ASP.NET es importante por lo que se requieren servidores de mayor capacidad.

De entre los modelos de servicio de Cloud Computing, el elegido para la realización de este proyecto, es el modelo de servicio, **Infrastructure as a Service (IaaS)**, en español Infraestructura como Servicio. En este modelo, una organización externaliza el equipamiento utilizado para sus operaciones. El proveedor de servicios Cloud, propietario de la infraestructura, se encarga de mantenerla y administrarla. El cliente obtiene el servicio que precisa en materia de **capacidad de procesamiento, almacenamiento (espacio), capacidad de red (ancho de banda) y sistemas operativos en una infraestructura a la que accede a través de Internet.**

De esta forma se traslada la complejidad computacional desde los ordenadores de los

gestores del servicio a las infraestructuras de un proveedor Cloud, lo que tiene como ventajas entre otras, la ubicuidad de los entornos de realización de prácticas, ahorro de costes, tolerancia a fallos, escalabilidad o la **anulación de cuellos de botella**.

El siguiente punto de cara a la escalabilidad de nuestro trabajo es el estudio de proveedores Cloud Públicos con el fin de identificar los más interesantes para la realización de este proyecto. Destacan Amazon Web Services, Windows Azure de Microsoft y DigitalOcean.

De cara a la integración y usabilidad de nuestro desarrollo, nos decantaremos por **Web Services ya que aportan interoperabilidad** entre aplicaciones de software independientemente de sus propiedades o de las plataformas sobre las que se instalen.

Los servicios Web fomentan los estándares y protocolos basados en texto, que hacen más fácil acceder a su contenido y entender su funcionamiento. Permiten que servicios y software de diferentes compañías ubicadas en diferentes lugares geográficos puedan ser combinados fácilmente para proveer servicios integrados.

1.4. Planificación del Trabajo.

Dentro de este apartado tenemos que describir en primer lugar las premisas que contempla con el objetivo de evitar posibles riesgos. Para ello, debemos tener en cuenta el tiempo efectivo de desarrollo del alumno para compaginar sus obligaciones profesionales con el desarrollo del TFG.

En ese sentido, debemos realizar una planificación adecuada del proyecto y un plan de trabajo que contemple esta situación, de tal forma que se detecten a través de un continuo seguimiento del proyecto, rápidamente desviaciones en los plazos de desarrollo. En ese mismo sentido se intentará **adelantar el tiempo del Plan de Trabajo** sobre el inicialmente descrito.

Por otro lado, el responsable de desarrollo es una única persona, por lo que podría incurrir en retrasos en el mismo debidos a causas externas (enfermedad, trabajo, etc).

Además, el entorno de desarrollo es un único equipo donde se están construyendo y almacenando los entregables del proyecto. Dicho equipo, deberá contar con una política de copias de seguridad en un entorno externo ejecutadas de manera diaria.

Para la consecución del estudio y cumplimiento del proyecto, hemos establecido las siguientes tareas que irán contribuyendo al desarrollo del presente documento:

- **Plan de Trabajo (del 27/09/2016 al 09/10/2016)**
    - **Alcance proyecto**

Definir las situaciones y/o problemas detectados y cuáles se va a abordar dentro del proyecto.

- **Estudio material previo**
  Decidir qué parte del aplicativo anterior tiene continuidad en el nuevo proyecto.
- **Formalización propuesta**
  Establecer programación para las distintas tareas que se abordarán en el proyecto.

- **Planificación y Análisis Requisitos (del 10/10/2016 al 27/10/2016)**

  - **Entrevista usuario/consultor**
    Consideración de nuevos requisitos y/o modificación de antiguos.

  - **Alcance funcional**
    Suma de requisitos o características de las operaciones a realizar. Se hará un análisis detallado de la estructura de los logs del Campus Virtual así como una especificación del proceso para el tratamiento de los logs del Campus Virtual y el análisis de patrones de navegación.

  - **Alcance tecnológico**
    Suma de requisitos o características de la arquitectura que soportará nuestro proyecto.

  - **Consolidación documentación**
    Cierre del análisis. En concreto, se incorporarán al proceso:
    - Diagramas de casos de uso
    - Descripción de mejoras
    - Definición de interfaces
    - Diseño de la capa de datos
    - Diseño de arquitectura tecnológica
    - Plan de pruebas

- **Desarrollo Proyecto (del 28/10/2016 al 11/12/2016)**

  - **Diseño de la Solución**

    - Diseño Funcional
      Traslado del documento de análisis (objetos, servicios,..) al entorno de desarrollo.

    - Diseño Tecnológico
      Traslado del documento de análisis (máquinas necesarias, framework,…) al entorno de desarrollo.

    - Preparación entorno de trabajo.

Establecer IDE desarrollo y creación de máquinas virtuales.

- Implementación del código
  Realización de la aplicación en el entorno montado.

- Ejecución plan de pruebas
  Búsqueda de errores en las distintas operaciones.

- Despliegue
  Finalización del software.

  - Manuales de Usuario
    Documentación para gestión por parte del usuario de la plataforma.

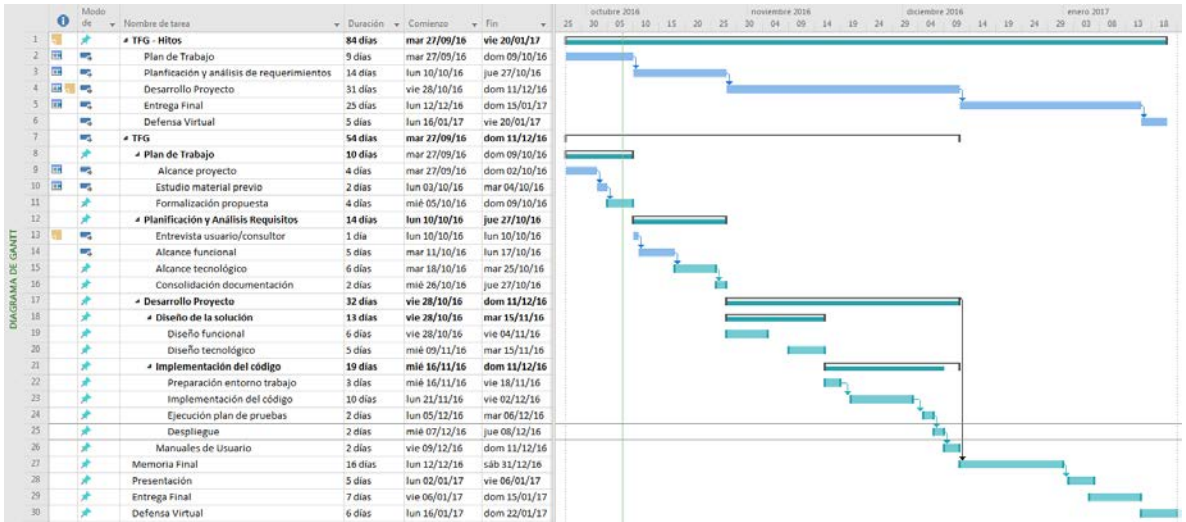
- **Memoria Final (del 12/12/2016 al 31/12/2016)**
  Documento completo.

- **Presentación (del 2/01/2016 al 6/01/2016)**
  Video explicativo del proyecto.

- **Entrega Final (del 6/01/2016 al 15/01/2016)**
  Cierre del proyecto. Esta actividad consistirá en la construcción y entrega de los manuales de usuario y administrador, software completo al 100%, memoria del PFC y presentación.

- **Defensa Virtual (del 16/01/2016 al 22/01/2016)**
  Valoración por parte del tribunal.

De manera gráfica podemos ver el siguiente diagrama temporal del proyecto:



	Modo de	Nombre de tarea	Duración	Comienzo	Fin
1		**TFG - Hitos**	**84 días**	**mar 27/09/16**	**vie 20/01/17**
2		Plan de Trabajo	9 días	mar 27/09/16	dom 09/10/16
3		Planficación y análisis de requerimientos	14 días	lun 10/10/16	jue 27/10/16
4		Desarrollo Proyecto	31 días	vie 28/10/16	dom 11/12/16
5		Entrega Final	25 días	lun 12/12/16	dom 15/01/17
6		Defensa Virtual	5 días	lun 16/01/17	vie 20/01/17
7		**TFG**	**54 días**	**mar 27/09/16**	**dom 11/12/16**
8		**Plan de Trabajo**	**10 días**	mar 27/09/16	dom 09/10/16
9		Alcance proyecto	4 días	mar 27/09/16	dom 02/10/16
10		Estudio material previo	2 días	lun 03/10/16	mar 04/10/16
11		Formalización propuesta	4 días	mié 05/10/16	dom 09/10/16
12		**Planificación y Análisis Requisitos**	**14 días**	**lun 10/10/16**	**jue 27/10/16**
13		Entrevista usuario/consultor	1 día	lun 10/10/16	lun 10/10/16
14		Alcance funcional	5 días	mar 11/10/16	lun 17/10/16
15		Alcance tecnológico	6 días	mar 18/10/16	mar 25/10/16
16		Consolidación documentación	2 días	mié 26/10/16	jue 27/10/16
17		**Desarrollo Proyecto**	**32 días**	**vie 28/10/16**	**dom 11/12/16**
18		**Diseño de la solución**	**13 días**	**vie 28/10/16**	**mar 15/11/16**
19		Diseño funcional	6 días	vie 28/10/16	vie 04/11/16
20		Diseño tecnológico	5 días	mié 09/11/16	mar 15/11/16
21		**Implementación del código**	**19 días**	**mié 16/11/16**	**dom 11/12/16**
22		Preparación entorno trabajo	3 días	mié 16/11/16	vie 18/11/16
23		Implementación del código	10 días	lun 21/11/16	vie 02/12/16
24		Ejecución plan de pruebas	2 días	lun 05/12/16	mar 06/12/16
25		Despliegue	2 días	mié 07/12/16	jue 08/12/16
26		Manuales de Usuario	2 días	vie 09/12/16	dom 11/12/16
27		Memoria Final	16 días	lun 12/12/16	sáb 31/12/16
28		Presentación	5 días	lun 02/01/17	vie 06/01/17
29		Entrega Final	7 días	vie 06/01/17	dom 15/01/17
30		Defensa Virtual	6 días	lun 16/01/17	dom 22/01/17

Figura 1 Diagrama temporal del Proyecto

1.5. Breve sumario de productos obtenidos.

La aplicación **Minería de Datos en la Nube** o Cloud Data Mining ha quedado concluida de la siguiente manera. Una aplicación web que permite:

- Procesamiento de ficheros log de cualquier tamaño obtenidos de los servidores Apache que acogen los servicios del campus virtual de la UOC. La aplicación tiene un único tipo de perfil: administrador/usuario. Dicho perfil accede a una página web en la que se muestra el proceso a obtener secuencialmente los ficheros Arff necesarios para la aplicación Weka.

- El usuario podrá, a través de una sencilla página, realizar el mantenimiento del almacenamiento de los ficheros logs de origen alojados en una estructura en la nube. Dicho almacenamiento acoge cualquier tipo de fichero.

1.6. Breve descripción de los otros capítulos de la memoria.

En los siguientes capítulos podremos ver cómo se ha realizado la etapa de planificación y análisis de requisitos para afrontar los objetivos marcados del proyecto. Además, contaremos con capítulos destinados, lógicamente, al desarrollo del software así como las conclusiones y propuestas funcionales obtenidas.

2. Planificación y Análisis Requisitos.

El punto de partida de nuestra aplicación será un fichero de texto de Logs generados por un servidor Web apache, con datos dispuestos de forma desestructurada que serán tratados para poder ser tratados por herramientas de análisis de datos.

Las conclusiones del análisis de requisitos se han nutrido de las diferentes fuentes de información de las que disponemos, principalmente de unas necesidades aportadas por el cliente, un análisis de información de Logs de navegación de Campus Virtual y la consulta de información y documentación a través de Internet y otras fuentes.

En este apartado diseñaremos las especificaciones o alcance oportunos para la construcción del software y llegar al resultado final.

2.1. Problemas previos a resolver.

Durante la fase de análisis del Proyecto el analista y Jefe de Proyecto se han encontrado con los siguientes problemas previos que habrán de ser resueltos:

1. **Cantidad ingente de datos**.
   El volumen de datos registrado es ingente, llegando a un ratio de más de diez millones de registros al día, con un espacio en disco superior a los diez GB. Por este motivo hay que optimizar bien el proceso para evitar tiempos de espera altos en su ejecución.

   Para ello, se establecerá el mecanismo de poner límites de principio y fin al proceso, pudiendo de esta forma analizar datos de la línea x a la línea y del fichero Log. Además, el propio usuario el que establezca subconjuntos de datos de estos Logs e introduzca ficheros de entrada de datos más pequeños.

2. **Soporte de la información**.
   La información de los Logs de navegación de los usuarios del Campus Virtual es generada por un servidor Web Apache y se almacena en ficheros de texto plano en el sistema de archivos o FileSystem, este soporte, evidentemente, no es óptimo para la explotación de la información ya que no tienen un acceso rápido. Es por ello que se deberá generar operaciones específicas de extracción y estructuración de la información.

   En el caso del **contenido**, si bien los ficheros de log siguen una estructura definida, esta estructura no permite extraer de forma directa los patrones de navegación cuyo análisis es el objeto del presente trabajo; por ello será necesario realizar un **preprocesamiento** de dichos ficheros de log. Se ha considerado dentro del alcance del presente TFG, la construcción de una aplicación que posibilite la realización de este procesamiento previo.

3. **Estructuración de la información**.
   Aunque cada línea de apunte de registro del fichero log tiene una serie de campos bajo una estructura, no se puede considerar de ningún modo como datos estructurados que habrá que procesar y estructurar.

4. **Arquitectura insuficiente para el proceso.**
   Este trabajo debe contemplar un entorno de producción donde la aplicación pueda en un momento dado permitir prescindir del hardware dedicado que tengamos desplegado en las dependencias de la universidad. Se trata de eliminar la inversión en hardware y el coste de las licencias de software, en el sentido de redimensionar los recursos con el mínimo coste.

2.2. Estructura logs del Campus Virtual.

Los ficheros de log del Campus Virtual de la UOC, generados por el servidor web Apache, están formados por millones de líneas soportados en ficheros de texto, uno por día, donde cada una de ellas representa la operación efectuada por un determinado estudiante dentro del campus virtual. Entre otra información, estas líneas registran la dirección IP del dispositivo desde el que se ha accedido y no el nombre de usuario del estudiante, lo que asegura una mayor privacidad.

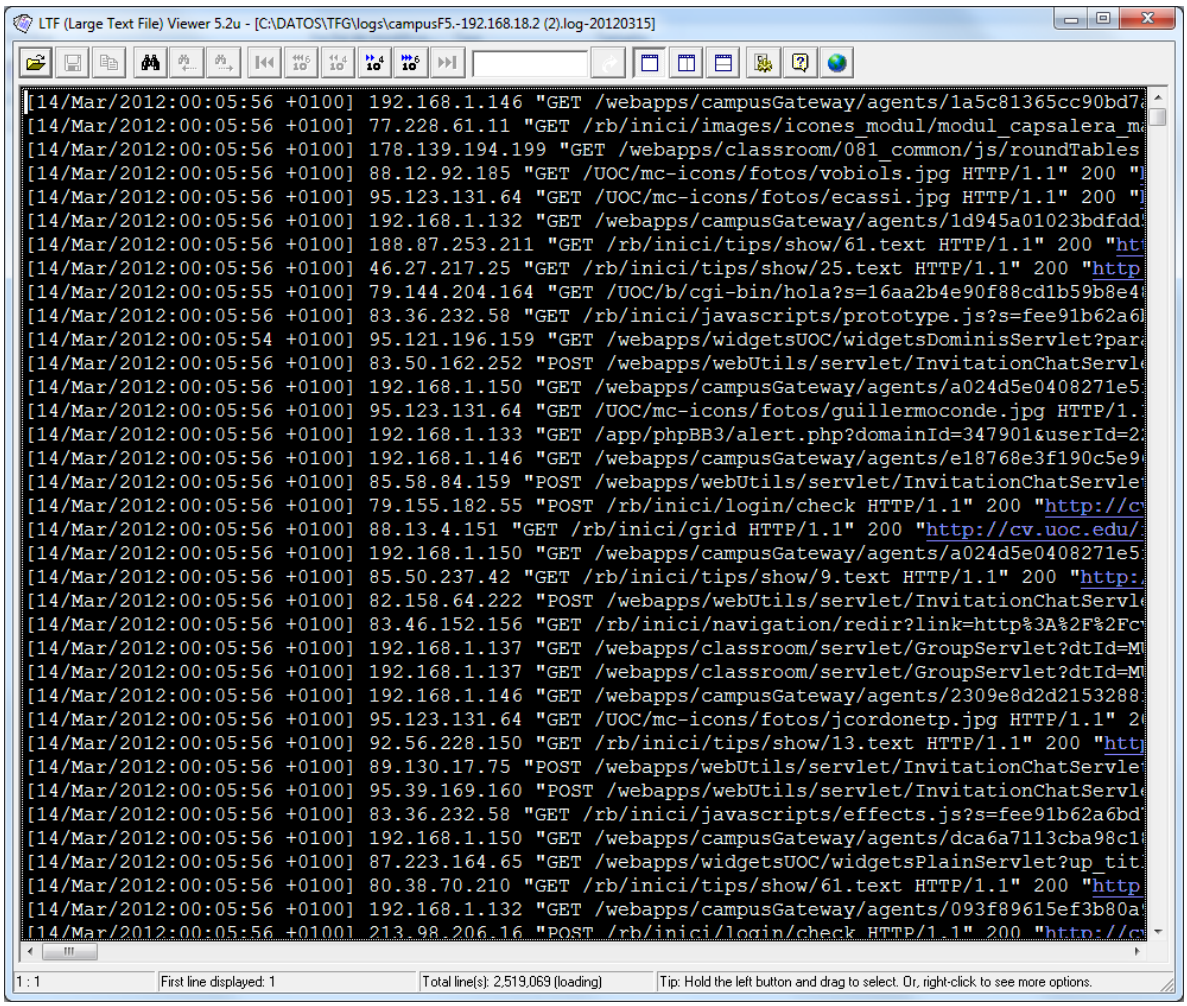


Figura 2 Ejemplo de registros del log Apache

Debemos ve qué posibilidades nos ofrece la información registrada en los logs del servidor en cuanto a posibilidades de explotación y análisis de datos. En ese sentido, analizaremos los

datos de entrada de los que se nutrirá nuestra aplicación, debiendo responder a las siguientes preguntas:

1. Formato de información.
2. Que información se registra.
3. Tipos de información. Existen códigos que hay que determinar que significan.

Los diferentes apuntes o registros (acciones de cada usuario) almacenados en los Logs, son generados por el servidor web perteneciente a la **Universidad Oberta de Cantalunya** desplegado en Apache, que almacena las acciones de los usuarios en filas (una por acción) en un fichero de texto correspondiente al día en que se realiza el apunte (al día siguiente se genera otro fichero).

Poder ver en un sistema operativo común como puede ser Windows (7, 10,…) un fichero de semejante tamaño necesita herramientas que no carguen el fichero completo en memoria. En por eso que los conocidos visores como Notepad, Notepad++, etc. no son válidos para esta tarea. Para realización de este proyecto hemos usado la herramienta **LTFViewer** que nos permite navegar por las filas sin problemas. Está diseñado para ver archivos de texto usando muy poca memoria y es capaz de abrir archivos muy grandes (> 1 GB) al instante. La **indexación de archivos en segundo plano hace que la navegación sea aún más rápida**. También nos permite realizar una búsqueda de texto compleja de alta velocidad por medio de texto plano o expresión regular. Otro punto a su favor es su licencia de tipo gratuito. A raíz de su uso hemos descubierto el patrón que deberíamos seguir para el procesamiento de grandes ficheros en nuestra aplicación, esto es, evitar la carga de los ficheros de origen en memoria. El resultado no poder ser más satisfactorio: **es capaz de abrir y leer un fichero de doce GB en menos de un segundo.**

Los datos generados en cada línea o acción, si bien no están estructurados, siguen una pauta que será necesario estudiar y analizar y que definiremos en el siguiente punto.

Una vez hecho esto, habrá que aplicar técnicas de minería de datos que permitan estructurar primero y posteriormente analizar dichos datos. Pasaremos ahora a definir la pauta que siguen los registros de los ficheros Logs. Como referencia se ha usado este enlace:

http://support.moonpoint.com/network/web/server/apache/log_format.php

Se trata de un fichero en formato texto con n líneas (dependiendo del tráfico de servidor web, normalmente puede generar millones de líneas) y cada vez que un usuario realiza una acción en la página web (en este caso en la UOC) se grabará una línea con los datos de acceso a un recurso realizada desde una determinada dirección IP y está formada por los siguientes elementos (16) separados por espacios:

- **Fecha y hora registro**: Momento temporal en el que se ha registrado la operación.
  Su formato es el siguiente: <día>/<mes>/<año>:<hora>:<minuto>:<segundo> <huso horario>

- **Dirección IP**: Identificador IP del dispositivo desde el que se ha realizado la acción registrada en el log.

- **Método petición**: Cada una de las líneas de los logs del Campus Virtual lleva asociado un tipo de operación identificado por el método de petición utilizado (POST, GET,

HEAD...).

- **Ruta recurso solicitado**: Ruta del Campus Virtual a la que se ha accedido, acompañada de los parámetros necesarios para realizar la operación demandada.

- **Protocolo**: Protocolo utilizado para acceder al recurso. Por ejemplo: HTTP/1.1.

- **Código respuesta**: Código de respuesta obtenido tras el acceso al recurso.

- **Ruta origen**: Dirección desde la que se ha realizado la solicitud del recurso.

- **Navegador**: Información identificativa del navegador desde el que se ha accedido al recurso.

- **N. º bytes obtenido**: Total de bytes recibidos al acceder al recurso. Por ejemplo, en caso de haber accedido correctamente a un fichero, el valor que aquí figuraría sería el tamaño del mismo.

- **Otros**: El log del campus virtual de la UOC dispone de un último parámetro que no ha conseguido identificarse. No obstante, dado que su relevancia es nula para el presente estudio, únicamente vamos a limitarnos a señalar su existencia.

2.2.1. Códigos de respuesta.

Es importante señalar que el protocolo considerado a efectos del presente estudio será únicamente HTTP; las líneas de los ficheros log que refieran a otros protocolos serán descartadas y no se considerarán a efectos de la determinación de patrones de navegación.

En el caso del protocolo HTTP, estos códigos de respuesta están normalizados, por lo que es posible determinar cuándo se ha producido un error en el acceso al recurso o cuando este se ha realizado con éxito. Dichos códigos están formados por tres dígitos siendo el primero de ellos el que identifica el tipo de respuesta según los siguientes criterios:

- **1xx**: Informativo. La petición se recibe y sigue el proceso. Esta familia de respuestas indican una respuesta provisional.

- **2xx**: Éxito. La acción requerida por la petición ha sido recibida, entendida y aceptada.

- **3xx**: Redirección. Para completar la petición se han de tomar más acciones.

- **4xx**: Error del cliente. La petición no es sintácticamente correcta y no se puede llevar a cabo. Por ser visibles desde el navegador, son especialmente conocidos los siguientes códigos:

    - 400 Petición errónea.
    - 401 Acceso no autorizado.
    - 403 Acceso prohibido.
    - 404 Recurso no encontrado

- **5xx**: Error del servidor. El servidor falla al atender la petición que aparentemente es correcta.

2.2.2. Métodos de petición

En cada línea de LOG se indica el método bajo el que se ha realizado la solicitud de una determinada operación. Estos son los siguientes:

- **GET**: Se utiliza para recuperar información, generalmente especificando parámetros en la URL, por ejemplo:
  http://midominio.es/miweb.htm?parametro1=valor1 & parametro2=valor2

- **POST**: Cuando una petición se realiza usando el método POST, los datos se adjuntan a la petición a modo de objeto. Una de las ventajas de POST sobre GET es que facilita el envío de un mayor volumen de datos al servidor y que los parámetros están ocultos (no se visualizan en la URL).

- **HEAD**: Es equivalente al método GET excepto que el servidor no devolverá contenido, sólo las cabeceras HTTP. Generalmente se utiliza para comprobar si un enlace es válido.

- **OPTIONS**: Permite al cliente conocer las opciones y requisitos asociados con un recurso o las capacidades del servidor.

El protocolo HTTP considera además los siguientes métodos:

- **PUT**: Permite guardar el contenido de la petición en el servidor bajo la URL de la petición.

- **DELETE**: Método utilizado para que el servidor borre el recurso indicado por la URL de la petición.

- **TRACE**: Se utiliza para determinar si existe el receptor del mensaje enviado y usar la información para hacer un diagnóstico.

2.3. Internacionalización de la aplicación.

Es necesario que los **usuarios de diferentes países y con diferentes culturas puedan** usar servicios adaptados correctamente para procesar información usando su idioma de origen, su sistema de escritura, su sistema de medida, sus calendarios y otras reglas y convenciones culturales.

La especificación de un conjunto particular de convenciones culturales es importante para que nuestra aplicación procese la información que intercambia con el usuario correctamente. Es aconsejable que pueda acceder a los diferentes entornos idiomáticos a través de sencillos desplegables o iconos con los distintos idiomas.

Figura 3 Desplegable idiomático UOC.

2.4. Patrones de navegación a analizar.

Cada acción que realiza el usuario en el Campus Virtual queda registrada en los Logs a través del registro de una serie de información determinada como hemos visto anteriormente. Por otra parte, el acceso al Campus Virtual implica el inicio de una sesión de trabajo por parte del usuario que ha accedido, dicha sesión de trabajo es identificada por una secuencia alfanumérica, que se realiza sobre un equipo informático (Tablet, portátil, PC, Smartphone,..) identificado con una dirección IP.

Bajo este razonamiento anterior es posible **vincular la sesión de navegación que realiza un usuario y dispositivo desde donde lo realiza con todos los datos registrados en una línea de log.**

Así pues el presente trabajo hará las trasformaciones necesarias en los ficheros Logs de navegación a través del Campus Virtual para que se puedan analizar, a modos de ejemplo, todos los patrones de navegación que sean posibles con los datos de los que disponemos.

Estos son algunos **ejemplos**:

1. **Recursos accedidos**.
   Este patrón permitirá identificar a aquellos **recursos solicitados de manera más usual** por los usuarios del Campus Virtual de la UOC cuyas acciones se han registrado en los logs.

   Dada la pareja dirección IP e identificador de sesión, denominaremos **secuencia de navegación** al conjunto de recursos visitados por una determinada dirección IP durante una sesión de trabajo; es decir, que una secuencia de navegación

estará formada por las URL a las que ha accedido un usuario desde que inicia sesión en el Campus Virtual hasta que finaliza su actividad. De esa forma, si un mismo usuario (identificado por la dirección IP del dispositivo desde el que ha accedido) iniciara una nueva sesión, el conjunto de recursos accedidos se consideraría parte de otra secuencia.

2. **Periodos de tiempo de navegación**.
   Este patrón de navegación permitirá identificar los tiempos en los que un usuario accede a los recursos, así como los periodos temporales donde se concentra la mayor cantidad de tráfico.

   Dada una determinada dirección IP e identificador de sesión, describiremos las horas a las que se conecta el usuario y con qué frecuencia solicita los diferentes recursos en una o varias sesiones de trabajo.

   Así pues se mostrará las horas a las que determinado usuario, en determinada sesión realiza peticiones de recursos del Campus Virtual de la UOC.

3. **Respuestas del servidor**.
   Este patrón de navegación permitirá identificar las respuestas del servidor apache del sitio web del Campus de la UOC que obtiene el usuario en una o varias sesiones. Esto tendrá especial interés en caso de los accesos erróneos y poder determinar comportamientos particulares.

   Dada una dirección IP que indique el dispositivo de conexión, y una sesión en la que se conecte, vinculará las respuestas que se ha obtenido del servidor web. De esta forma se mostrarán los códigos de respuesta que determinada IP a obtenido en determinada sesión.

4. **Tipos de recursos accedidos**.
   Este patrón de navegación permitirá determinar los tipos de recursos a los que se accede mayoritariamente en cada dispositivo y en cada sesión.

   Dada una determinada IP en una cierta sesión se detallará los tipos de recursos accedidos (jpg, htm, cgi, php,….) y establecer patrones de uso de los usuarios.

5. **Exploradores web usados**.
   Este patrón de navegación establecerá los exploradores web más usados y vinculará las diferentes sesiones con diferentes dispositivos con el explorador usado. Esto tiene especial interés para definir patrones de uso de navegadores web en las diferentes sesiones del usuario.

   Si bien, evidentemente cada sesión (secuencia de navegación) corresponderá únicamente a un tipo de explorador, será posible sacar conclusiones con respecto a tipos de exploradores más utilizados o con estudios más complejos, con exploradores por zonas geográficas o exploradores relacionados con

códigos de error.

6. **Ubicación desde donde se navega**.
   Este patrón de navegación establecerá las ubicaciones geográficas correspondientes a una determinada IP para una sesión en concreto. Al igual que en el punto anterior, evidentemente, cada sesión de navegación corresponderá a una única ubicación, pero será posible sacar conclusiones de las ubicaciones desde donde navegan los diferentes usuarios en cada sesión, mayor concentración de conexiones, o estudios más complejos como son que combinen por ejemplo las horas en las que se conecta cada zona geográfica.

   Es importante tener en cuenta que el dato de la ubicación geográfica no viene definido en los Logs del servidor Apache, por ello tendremos que diseñar mecanismos que permitan obtener de manera externa este dato a través de la dirección IP.

2.5. Métodos aplicados.

El punto de partida será un estudio del formato de datos necesarios para el software destino (Weka). Posteriormente, después de haber aplicado el proceso de minería de datos, estructurándolos y realizando su procesado, será necesario utilizar métodos de análisis de dichos datos.

Usaremos KMeans (o método de los centroides) Ya que es un **método de agregación** (del inglés **clustering**) que, como tal, propone a partir de un conjunto de datos, la obtención de una **enumeración de grupos** (clusters) **de objetos con características similares**. Concretamente, el método KMeans, se basa en la obtención de un número *k* de grupos que es **fijado al principio del proceso**.

El proceso comienza fijando un punto inicial del espacio (denominado **semilla** o, en inglés, seed) como centro del grupo potencial que se va a formar. Esta semilla puede ser bien uno de los objetos que forman parte del conjunto de datos inicial, bien una combinación de valores creada de forma artificial representando un resumen de las características de varios objetos. Con dicha premisa, podemos extraer los pasos que sigue el método KMeans:

- Seleccionar las semillas iniciales
- Calcular los centros
- Asignar objetos al grupo con centro más próximo
- Recalcular los centros
- Continuar hasta que no haya variación en los grupos

Una vez formados los grupos, se determina, para cada objeto a estudiar, qué centroide tiene más cerca asignando dicho objeto al grupo representado por el centroide más próximo. A este proceso se le denomina **agregación**.

Como resultado de la agregación de objetos, es necesario volver a calcular el centro de cada grupo. Esto se hace obteniendo para cada dimensión, el valor medio de todos los objetos que forman parte del grupo tratado.

Una vez calculados los nuevos centros, el proceso se inicia otra vez y se repite hasta que, en dos iteraciones consecutivas no se produzcan cambios en los centros (o se produzcan pocos).

2.6. Aplicación externa para el procesado de los logs.

Como salida de la aplicación, obtendremos un fichero intermedio y uno final que se podrán estudiar con una herramienta de análisis de datos.

En nuestro caso la aplicación seleccionada para el estudio será Weka, que contiene una colección de herramientas de visualización y algoritmos para análisis de datos y modelado predictivo, que además cuenta con una interfaz gráfica de usuario muy atractiva que permite acceder fácilmente a todas sus funcionalidades.

Es una herramienta de código libre que se puede descargar fácilmente en:
http://www.cs.waikato.ac.nz/ml/weka/index_downloading.html

Figura 4 Pantalla bienvenida aplicación Weka

Las principales características de esta herramienta son las siguientes:

- Está disponible libremente bajo la licencia pública general de GNU.
- Es muy portable porque está completamente implementado en Java y puede correr en casi cualquier plataforma.
- Contiene una extensa colección de técnicas para preprocesamiento de datos y modelado.
- Es fácil de utilizar por un principiante gracias a su interfaz gráfica de usuario.

Weka soporta varias tareas estándar de minería de datos, especialmente,

preprocesamiento de datos, clustering, clasificación, regresión, visualización, y selección. Todas las técnicas de Weka se fundamentan en la asunción de que los datos están disponibles en un fichero plano (*flat file*) o una relación, en la que cada registro de datos está descrito por un número fijo de atributos (normalmente numéricos o nominales, aunque también se soportan otros tipos).

También proporciona acceso a bases de datos vía SQL gracias a la conexión JDBC (*Java Database Connectivity*) y puede procesar el resultado devuelto por una consulta hecha a la base de datos. No puede realizar minería de datos multi-relacional, pero existen aplicaciones que pueden convertir una colección de tablas relacionadas de una base de datos en una única tabla (desnormalización) que para ser procesada con Weka.

Para nuestro caso, implementa el método K-Means a través de un algoritmo denominado SimpleKMeans implementado en la clase *weka.clusterers.SimpleKMeans*.

Existen infinidad de manuales y tutoriales en la red que nos indican cómo manejar la aplicación y sacarle el máximo rendimiento. A continuación recogemos los principales pasos que tendremos que realizar para aplicar el algoritmo escogido al conjunto de datos de salida

- **Pantalla principal de la aplicación.**
  Desde el primero momento podremos acceder a todas las funcionalidades de la aplicación. Exploraremos las imprescindibles para el ámbito del actual TFG.

Figura 5 Pantalla principal de la aplicación.

- **Carga de datos.**
  Para poder comenzar a usar Weka, necesitaremos cargar los datos que hemos obtenido de nuestra aplicación. Para ello bastará con clica en el botón "Explorer" y aparecerá la siguiente pantalla, que será donde cargaremos y aplicaremos el algoritmo.

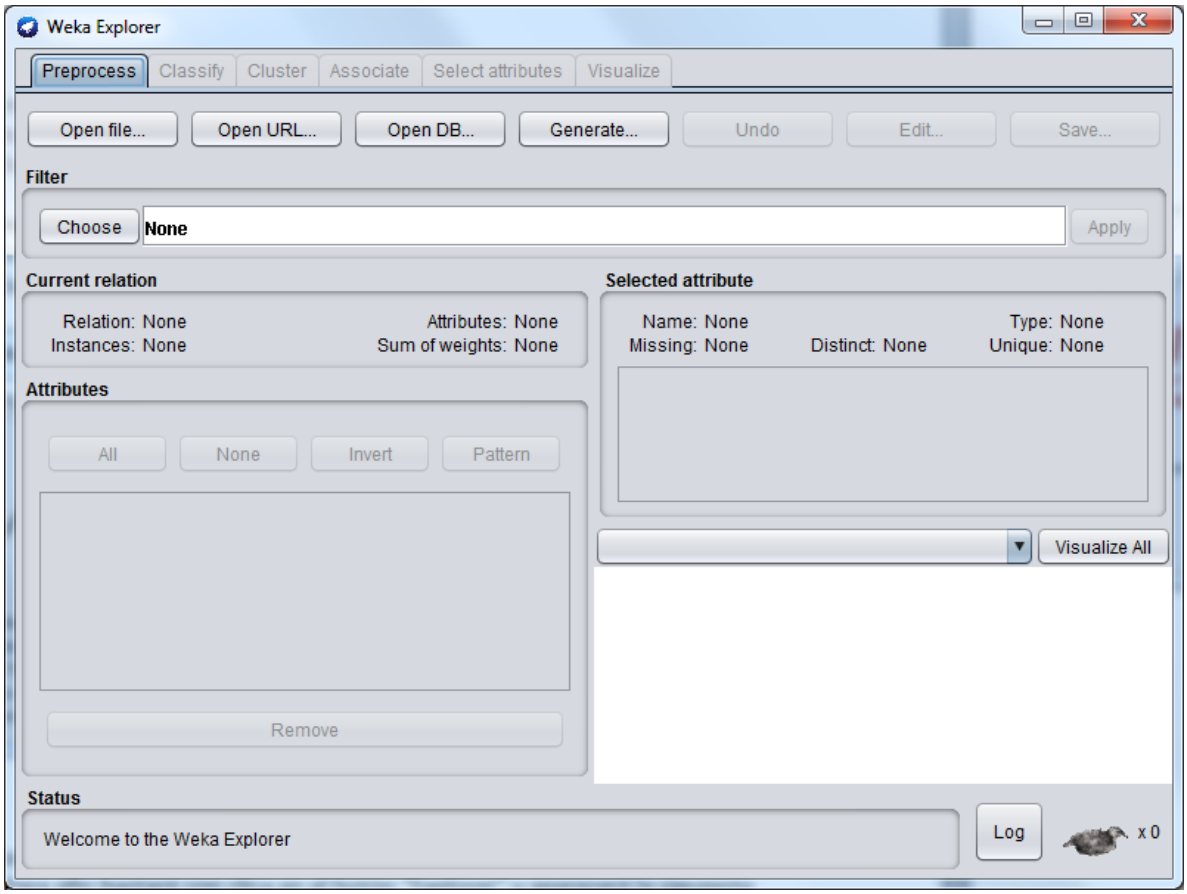


Figura 6 Opción "Explorer"

Abrimos el fichero clicando en “Open file…”.

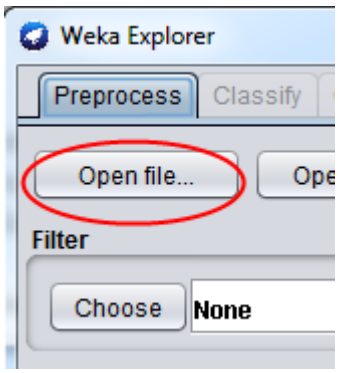


Figura 7 Open file

Y seleccionamos el fichero origen que deseamos estudiar.

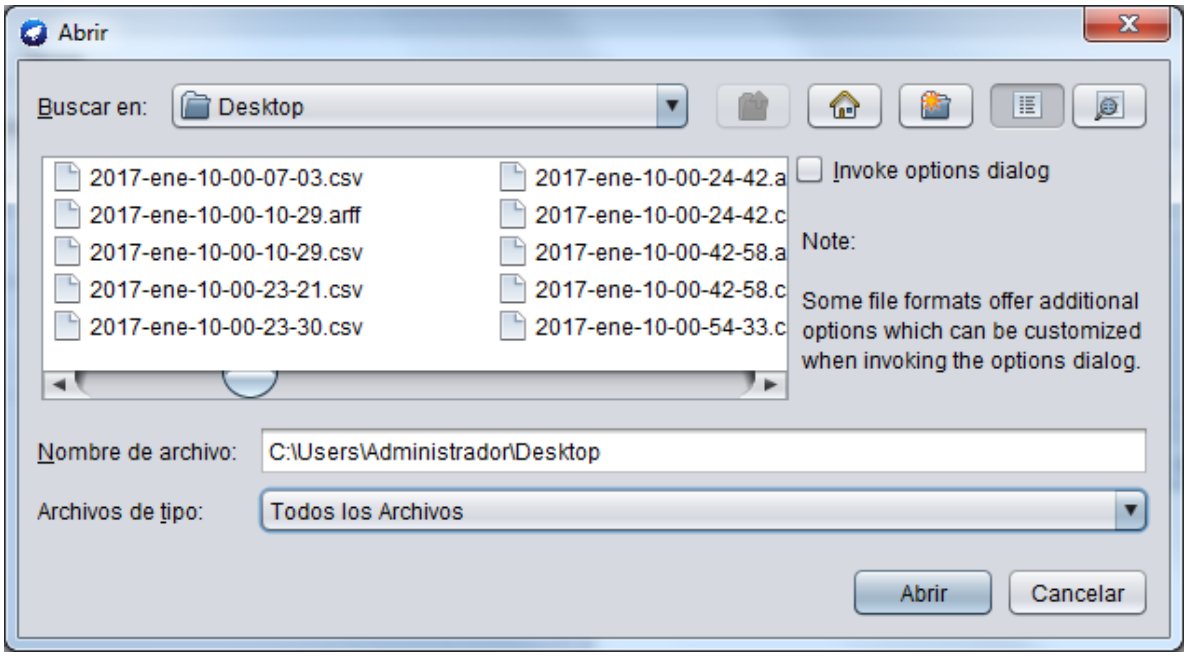


Figura 8 Selección de archivo a estudiar

Nuestra aplicación devolverá archivos de dos tipos. Por un lado tendremos los ficheros CSV (**fichero intermedio**) que nos informarán de los datos que vamos a estudiar.

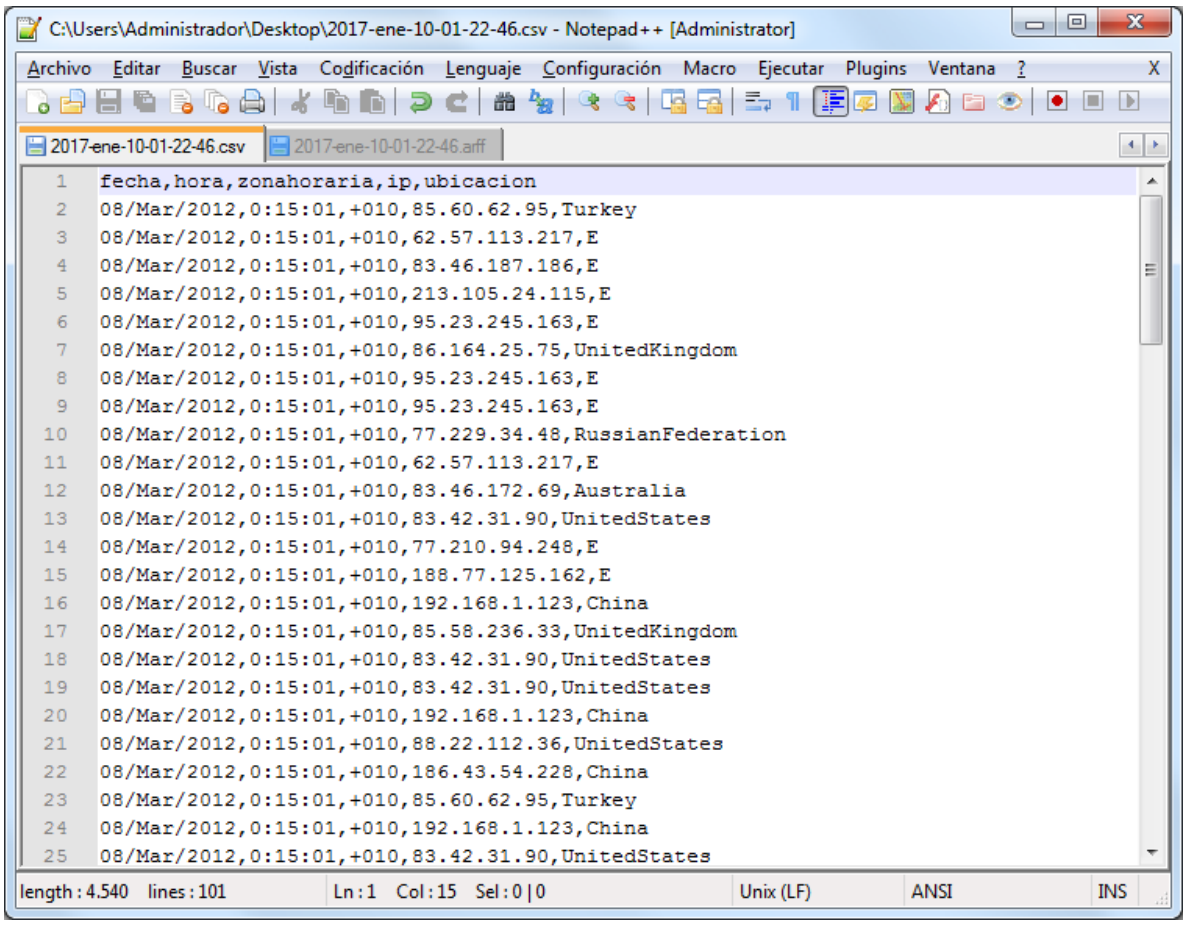


Figura 9 Fichero intermedio CSV

Y por otro lado obtendremos también el fichero ARFF (**fichero final**) generado a partir del fichero intermedio CSV con el que podremos trabajar directamente en Weka.

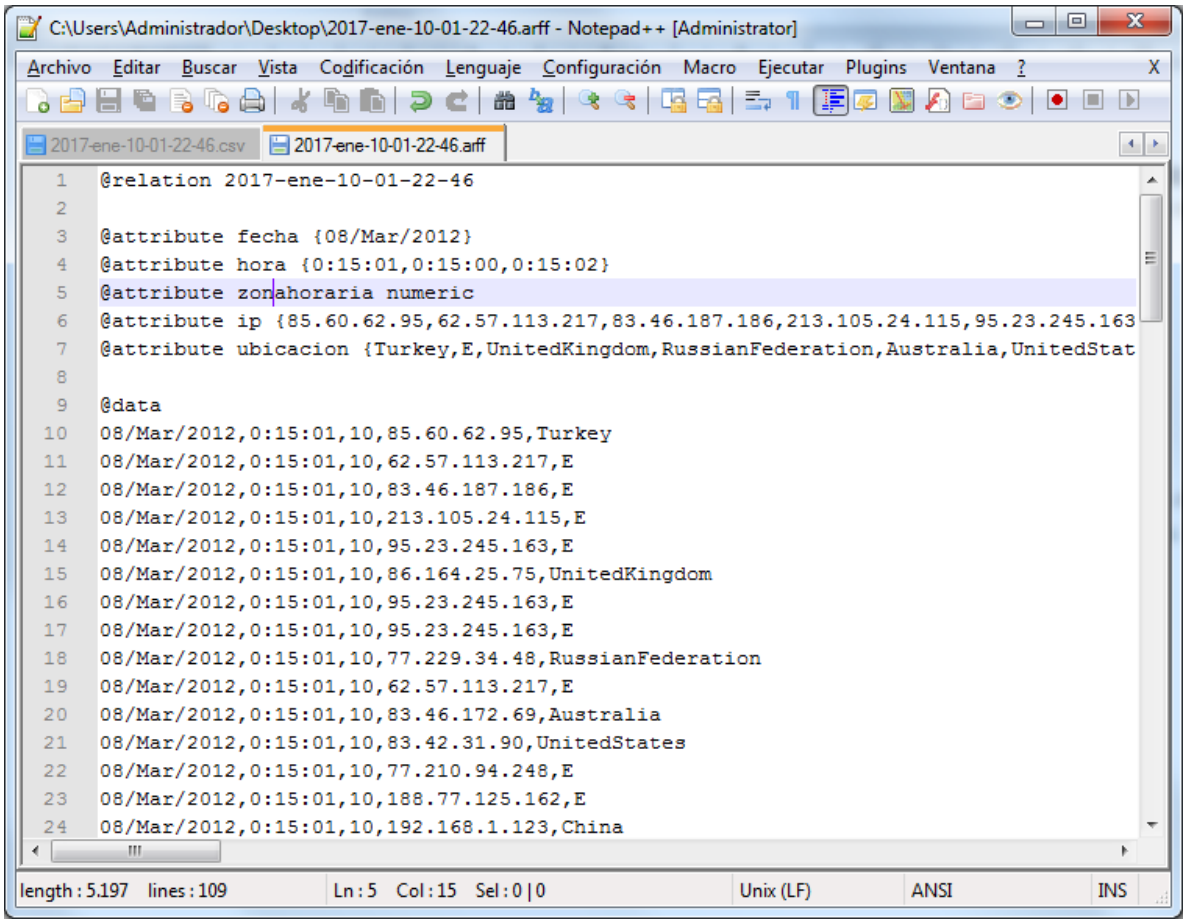


Figura 10 Fichero final ARFF

Nativamente Weka trabaja con un formato denominado arff , acrónimo de Attribute-Relation File Format. Este formato está compuesto por una estructura claramente diferenciada en tres partes:

- **Cabecera**.
  Se define el nombre de la relación. Su formato es el siguiente:

  @relation <nombre-de-la-relación>

  Donde <nombre-de-la-relación> es de tipo String*. Si dicho nombre contiene algún espacio será necesario expresarlo entrecomillado.

- **Declaraciones de atributos.**
  En esta sección se declaran los atributos que compondrán nuestro archivo junto a su tipo. La sintaxis es la siguiente:

@attribute <nombre-del-atributo> <tipo>

Donde <nombre-del-atributo> es de tipo String teniendo las mismas restricciones que el caso anterior. Weka acepta diversos tipos, estos son:

a) NUMERIC Expresa números reales.
b) INTEGER Expresa números enteros.
c) DATE Expresa fechas, para ello este tipo debe ir precedido de una etiqueta de formato entrecomillada. La etiqueta de formato está compuesta por caracteres separadores (guiones y/o espacios) y unidades de tiempo:

dd Día.
MM Mes.
yyyy Año.
HH Horas.
mm Minutos.
ss Segundos.

d) STRING Expresa cadenas de texto, con las restricciones del tipo String comentadas anteriormente.

e) ENUMERADO El identificador de este tipo consiste en expresar entre llaves y separados por comas los posibles valores (caracteres o cadenas de caracteres) que puede tomar el atributo.

- **Sección de datos.**
  Declaramos los datos que componen la relación separando entre comas los atributos y con saltos de línea las relaciones. Ejemplo para atributos fecha, hora, ip y localización:

@data
08/Mar/2012,0:15:01,10,85.60.62.95,Turkey
08/Mar/2012,0:15:01,10,62.57.113.217,E
08/Mar/2012,0:15:01,10,83.46.187.186,E
08/Mar/2012,0:15:01,10,213.105.24.115,E
08/Mar/2012,0:15:01,10,95.23.245.163,E
08/Mar/2012,0:15:01,10,86.164.25.75,UnitedKingdom
08/Mar/2012,0:15:01,10,95.23.245.163,E
08/Mar/2012,0:15:01,10,95.23.245.163,E
08/Mar/2012,0:15:01,10,77.229.34.48,RussianFederation

Nótese que hemos codificado el campo nulo o vacío con el **valor "E"** para poder interpretarlo mejor en los proceso de minería de datos.

Tanto en formato CSV como en formato ARFF, Weka está preparado para incorporar los dos tipos. Como veremos con más detalle en la etapa de desarrollo, hemos

decidido mantener los dos tipos por un mejor control de los procesos a la hora de evitar los posibles cuellos de botella del proceso. No obstante disponemos de infinidad de herramientas gratuitas para pasar de un formato a otro además del propio Weka en su opción Tools → ArffViewer.

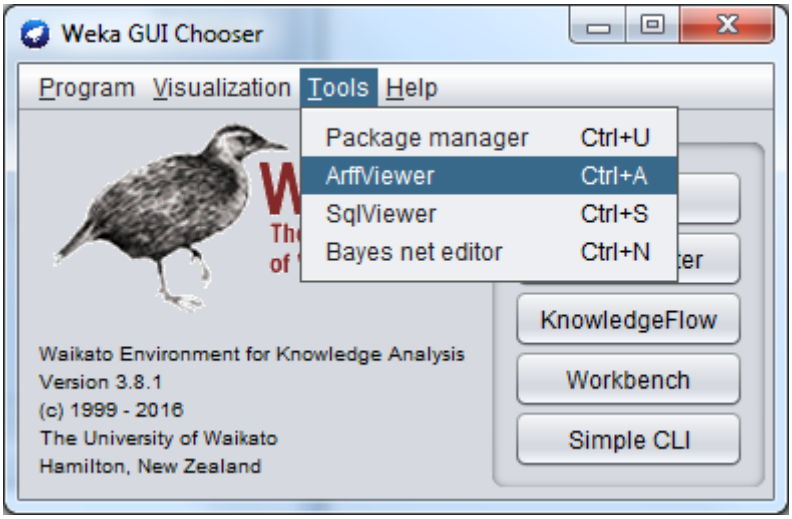


Figura 11 Conversor CSV ---> ARFF

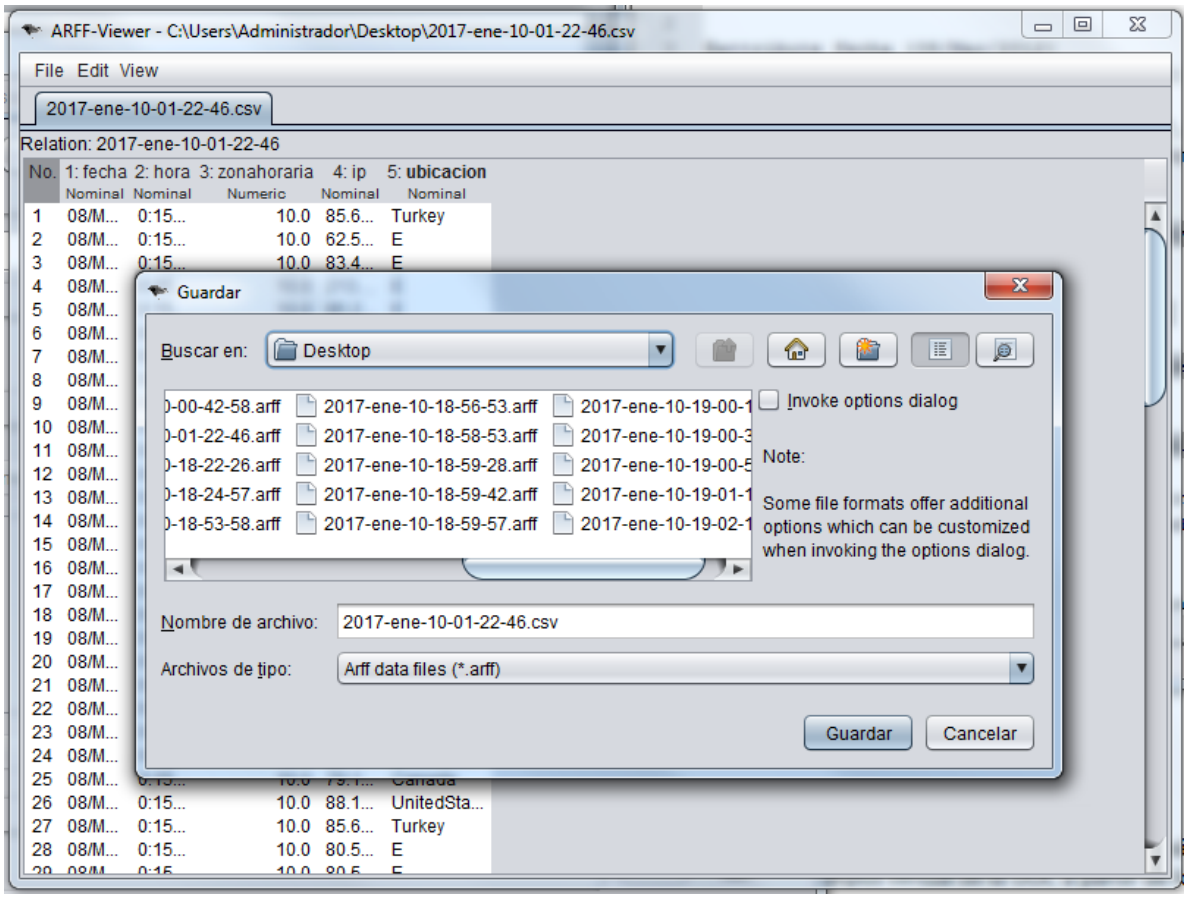


Figura 12 Guardar como tipo ARFF

Ya sea en un formato o en otro, una vez cargado el fichero podemos ver como Weka nos muestra las cabeceras, instancias y atributos de partida.

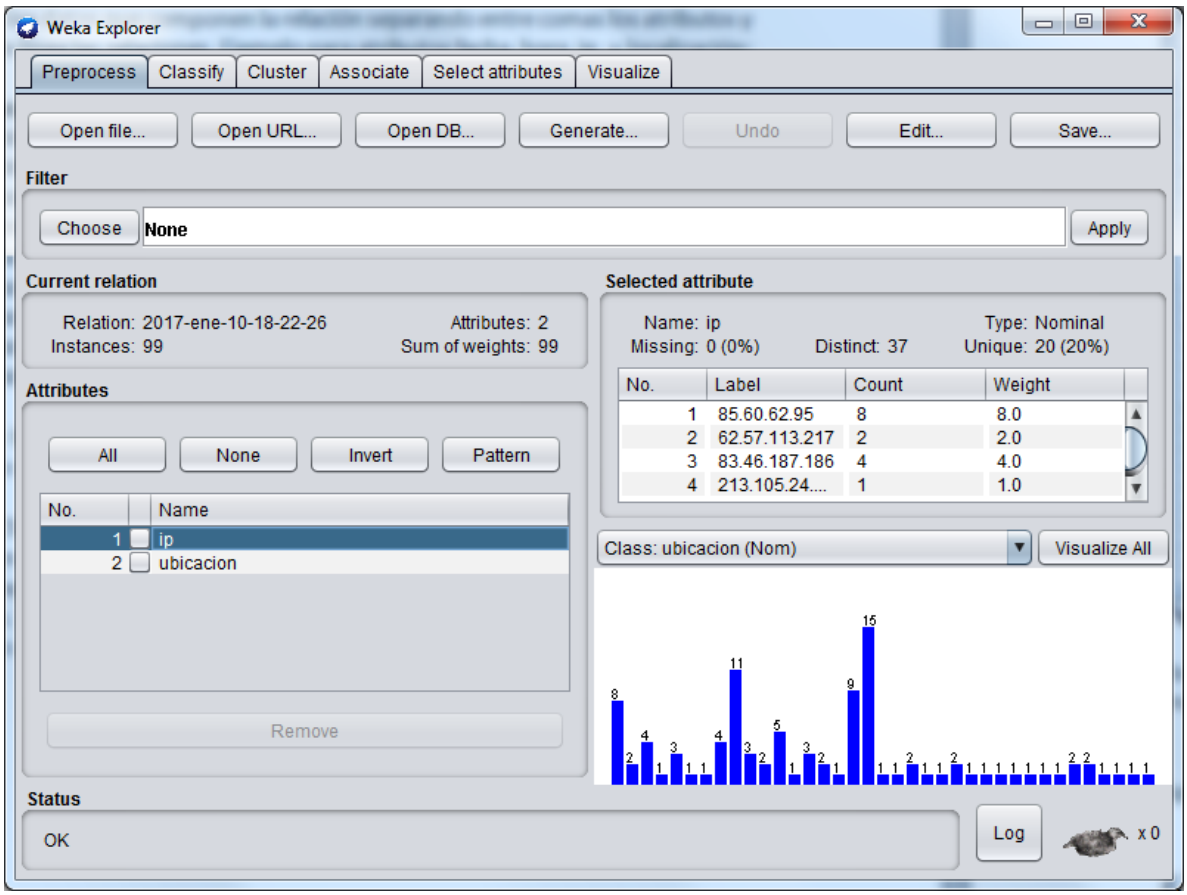


Figura 13 Fichero ARFF cargado correctamente.

- **Aplicación del algoritmo Kmeans.**

  En este momento ya estaremos en disposición de aplicar el algoritmo Kmeans, por lo que simplemente tendremos que usar la opción “cluster” sobre los datos cargados.

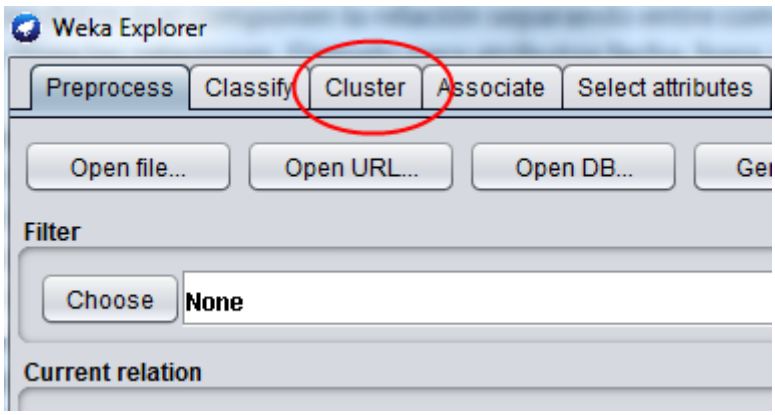


Figura 14 Opción Clúster.

Y seleccionaremos el botón “Choose” para elegir el método clúster que queremos aplicar.

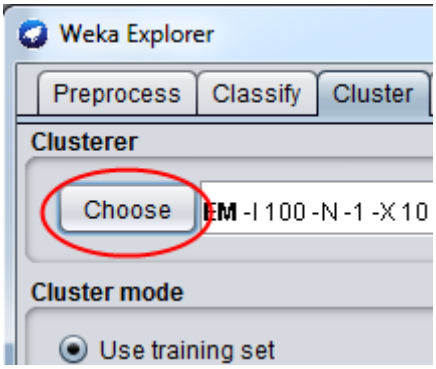


Figura 15 Opción "Choose" del Cluster.

Se nos desplegará un árbol donde aparecen todos los algoritmos disponibles donde seleccionaremos “SimpleKMeans”.

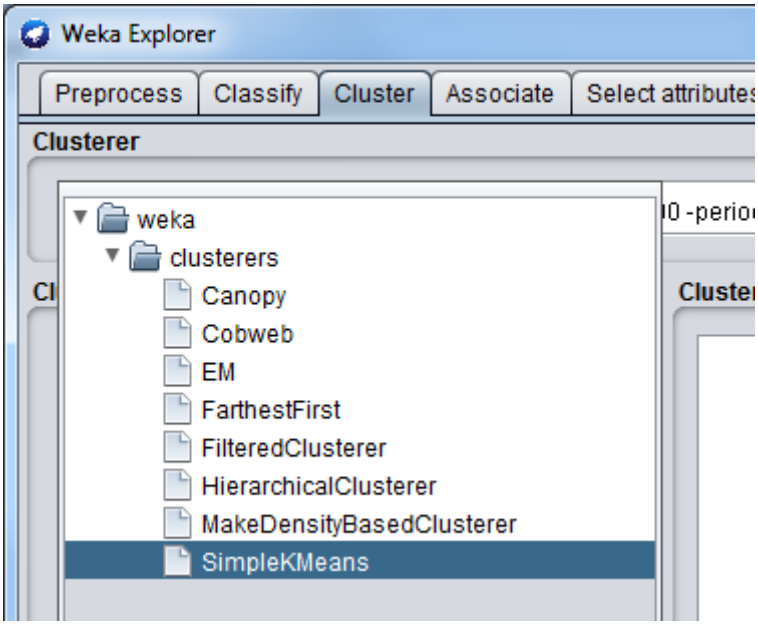


Figura 16 Algoritmos disponibles.

En este punto ya tendríamos cargados los datos y seleccionado el algoritmo. Weka implementa el método K-Means a través de un algoritmo denominado SimpleKMeans implementado en la **clase Weka.clusteres.SimpleKMeans**.

La ejecución es parametrizable a través de una serie de opciones donde las más relevantes son las siguientes:

Parámetro	Descripción
numClusteres	Total de agrupaciones (clústeres)

Seed	Valor de la semilla a partir de la cual se generará el número aleatorio que inicializará los centros de los clústeres.

Para acceder la pantalla de parametrización, debemos clicar en el nombre del algoritmo.

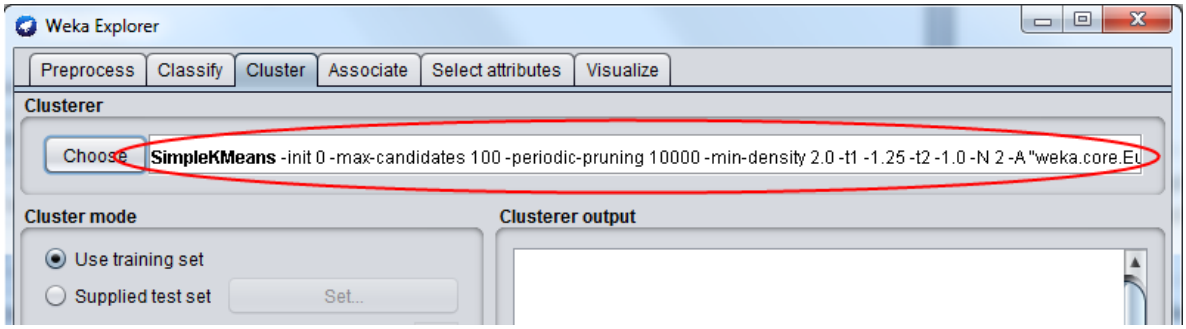


Figura 17 Apertura parametrización algoritmo.

Una vez dentro del formulario de parametrización podremos cambiar los dos parámetros descritos y otros más específicos del algoritmo.

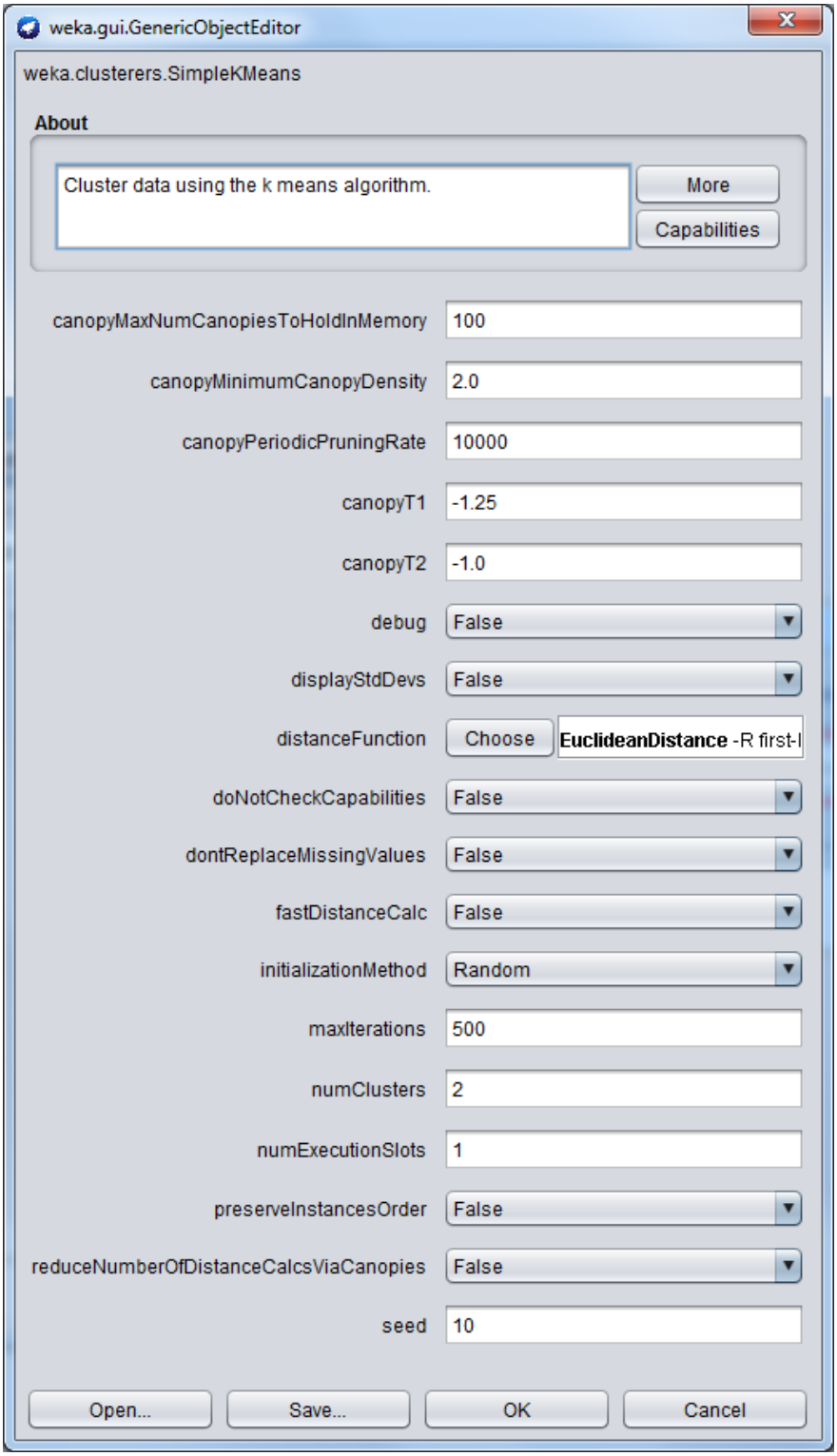


Figura 18 Parametrización algoritmo.

2.7. Alcance funcional.

El objetivo de este trabajo es el Análisis de patrones de navegación de los estudiantes del campus virtual de la UOC a partir de los ficheros Logs generados por el sitio www.uoc.edu. En este punto se especificarán las actividades para cumplir el citado objetivo.

Sirva como visión general del proceso a realizar el siguiente flujograma:

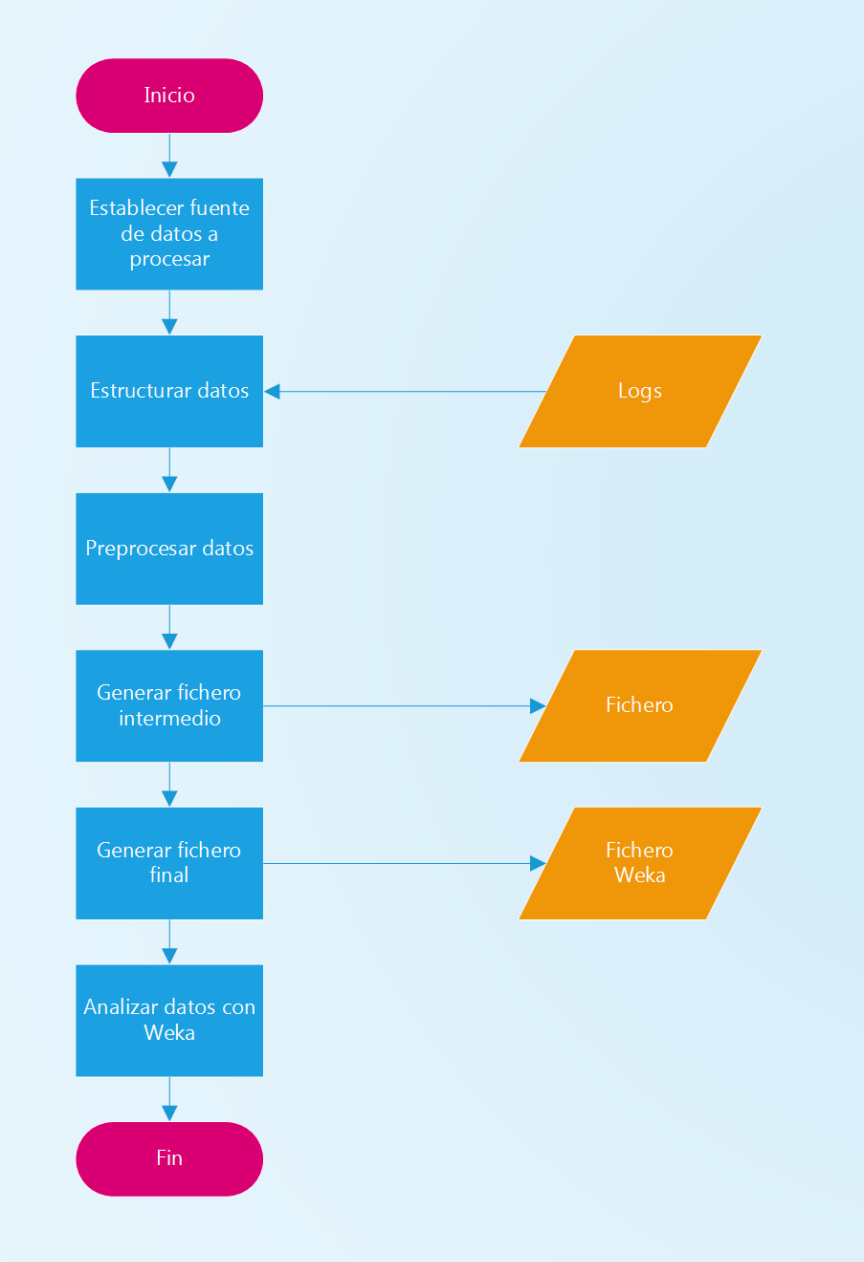


Figura 19 Flujograma de la Aplicación

2.7.1. Fuente de datos a procesar.

Esta primera actividad va a consistir únicamente en seleccionar la fuente de datos que se va a importar y que va a ser fuente de estudio. Es importante hacer notar como requerimiento que estos datos de entrada deben tener un formato muy claro y concreto, y es el que genera el servidor web Apache en sus Logs de sistema como ya hemos descrito anteriormente.

Necesitaremos poder disponer de datos reales (proporcionado por el consultor) y poder estudiar datos que ofrecen los logs de del Campus Virtual de la UOC, estudiar la estructura de dichos datos y poder realizar baterías de pruebas preliminares en la depuración del software.

2.7.2. Estructuración de los datos.

La segunda actividad del proceso consiste en importar y estructurar los datos procedentes del fichero de logs en texto plano generado por el servidor web Apache, es decir, saber cuál es el formato de la información y sus tipos (determinar qué significan ciertos códigos).

La información que se registra en los logs de apache tiene varios problemas en cuanto a su explotación:

- **No está estructurada.**
  Los llamados “separadores” de las distintas columnas de cada línea pueden ser espacios en blanco, guiones, interrogantes, etc. Será necesario determinar cada una de ellas y aplicar a cada campo el método correcto para extraer su información y estructurarla.

- **Fichero de texto**.
  Cargar en memoria grandes ficheros de texto conlleva que los procesos del sistema operativo que ejecutan nuestra aplicación consuman memoria en exceso. Es un tema importante a vigilar.

Realizando el proceso de importación deberemos resolver los problemas relativos a la **cantidad ingente de información**, el **soporte de la misma** y así como el **tratamiento de la información desestructurada** aplicando diferentes mecanismos. Consistirá en ir línea por línea del fichero de texto leyendo datos. Más en detalle, se configurarán dos mecanismos para paliar en cierta medida el problema descrito sobre el volumen de información a tratar. Evidentemente, esta cantidad de información es prácticamente inabordable y debemos evitar caer en bucles pseudo-infinitos, por ello se ha habilitado dos mecanismos:

- **Hacer subconjuntos de datos**.

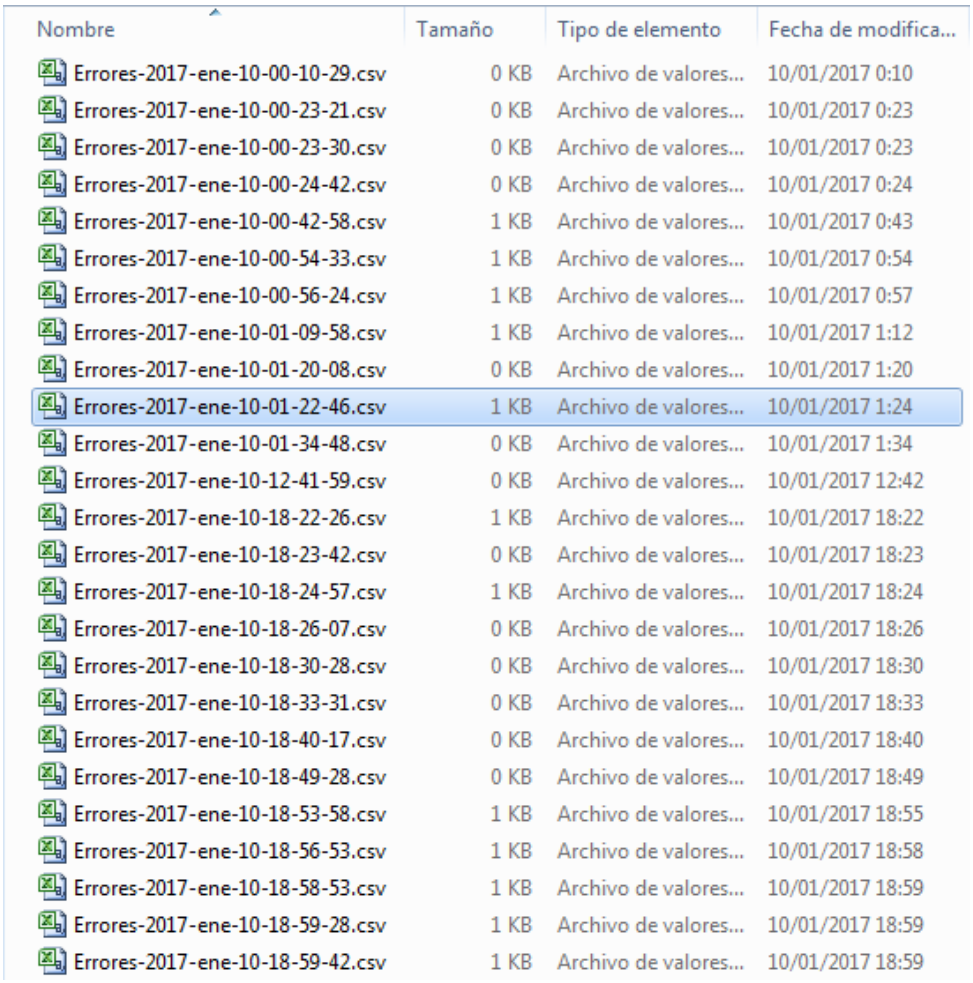
Se habilitarán dos opciones de entrada de datos. La primera opción consistirá en que el usuario edite a mano el fichero LOG total en pequeños subconjuntos de datos manejables y que los vaya importando secuencialmente en el aplicativo de análisis de datos.

- **Leer fichero entre filas**.
  La segunda radica en leer el fichero log completo desde la primera a la última fila (bajo responsabilidad del usuario) o leer fichero entre una fila de inicio X y una fila de fin Y, que leerá el fichero log entre estos límites. Este último caso permite hacer el estudio de datos entre ciertos límites abordables por el programa y que el usuario deberá determinar de acuerdo a la capacidad del host donde se ejecute. Esta parametrización no sólo es muy importante para desarrollar el plan de pruebas de cara a lograr que la arquitectura Cloud dé respuesta al elevado volumen de información a tratar sino también para ver cuál es la capacidad de importación y tratamiento de filas de Weka una vez hallamos generado el fichero arff final.

El proceso leerá línea por línea obteniendo y extrayendo la información de cada una y metiéndola en campos de estructuras de memoria diseñados a tal fin que permanecerá ahí cargada pendiente de un procesamiento previo.

Por otro lado, todas las aplicaciones en entornos profesionales tienen un módulo de auditoría que registra las acciones del usuario y del sistema con más o menos detalle en función de una parametrización, guardando en ficheros de texto fechados en el momento en que se realiza el apunte, una línea por acción. En nuestro caso, no guardaremos un log de la aplicación en cuanto a las operaciones acertadas ya que nuestra prioridad es la velocidad de procesamiento y dicha escritura a disco sería innecesaria.

Creemos, eso sí, necesario que el sistema informe de las filas que han **no han podido evaluarse** de cara a una posible depuración de fallos. Por ello, el sistema generará un fichero de salida con las líneas que han sido desestimadas. Las operaciones correctas estarán en el fichero intermedio CSV así como en el ARFF definitivo.



Nombre	Tamaño	Tipo de elemento	Fecha de modifica...
Errores-2017-ene-10-00-10-29.csv	0 KB	Archivo de valores...	10/01/2017 0:10
Errores-2017-ene-10-00-23-21.csv	0 KB	Archivo de valores...	10/01/2017 0:23
Errores-2017-ene-10-00-23-30.csv	0 KB	Archivo de valores...	10/01/2017 0:23
Errores-2017-ene-10-00-24-42.csv	0 KB	Archivo de valores...	10/01/2017 0:24
Errores-2017-ene-10-00-42-58.csv	1 KB	Archivo de valores...	10/01/2017 0:43
Errores-2017-ene-10-00-54-33.csv	1 KB	Archivo de valores...	10/01/2017 0:54
Errores-2017-ene-10-00-56-24.csv	1 KB	Archivo de valores...	10/01/2017 0:57
Errores-2017-ene-10-01-09-58.csv	1 KB	Archivo de valores...	10/01/2017 1:12
Errores-2017-ene-10-01-20-08.csv	0 KB	Archivo de valores...	10/01/2017 1:20
Errores-2017-ene-10-01-22-46.csv	1 KB	Archivo de valores...	10/01/2017 1:24
Errores-2017-ene-10-01-34-48.csv	0 KB	Archivo de valores...	10/01/2017 1:34
Errores-2017-ene-10-12-41-59.csv	0 KB	Archivo de valores...	10/01/2017 12:42
Errores-2017-ene-10-18-22-26.csv	1 KB	Archivo de valores...	10/01/2017 18:22
Errores-2017-ene-10-18-23-42.csv	0 KB	Archivo de valores...	10/01/2017 18:23
Errores-2017-ene-10-18-24-57.csv	1 KB	Archivo de valores...	10/01/2017 18:24
Errores-2017-ene-10-18-26-07.csv	0 KB	Archivo de valores...	10/01/2017 18:26
Errores-2017-ene-10-18-30-28.csv	0 KB	Archivo de valores...	10/01/2017 18:30
Errores-2017-ene-10-18-33-31.csv	0 KB	Archivo de valores...	10/01/2017 18:33
Errores-2017-ene-10-18-40-17.csv	0 KB	Archivo de valores...	10/01/2017 18:40
Errores-2017-ene-10-18-49-28.csv	0 KB	Archivo de valores...	10/01/2017 18:49
Errores-2017-ene-10-18-53-58.csv	1 KB	Archivo de valores...	10/01/2017 18:55
Errores-2017-ene-10-18-56-53.csv	1 KB	Archivo de valores...	10/01/2017 18:58
Errores-2017-ene-10-18-58-53.csv	1 KB	Archivo de valores...	10/01/2017 18:59
Errores-2017-ene-10-18-59-28.csv	1 KB	Archivo de valores...	10/01/2017 18:59
Errores-2017-ene-10-18-59-42.csv	1 KB	Archivo de valores...	10/01/2017 18:59

Figura 20 Ficheros de errores

El fichero de errores contendrá toda la información de la línea evaluada para que podamos establecer las causas del error.

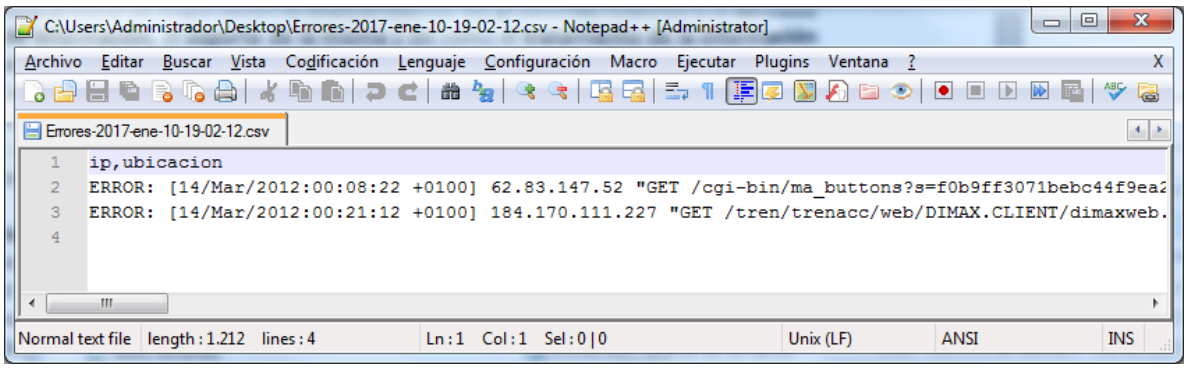


Figura 21 Detalle fichero error.

Hemos identificado los siguientes campos contenedores de información en una línea de registro de log:

Campo	Descripción
Fecha	Fecha en que se produjo el registro delo LOG. fecha en que se realiza la petición al servidor web de la página
Hora	Hora en que se produjo el registro del LOG, es decir, momento en que se solicita el recurso.
Zona Horaria	Diferencia horaria respecto GMT
Ip	Dirección IP en formato de cuatro octetos desde la cual el usuario se conecta
Method	Método de acceso al recurso. GET/POST
Url	Dirección completa de acceso a la página, parámetros incluidos
Recurso	Recuso al que accede la web. Sin parámetros.
TipoRecurso	Tipo de recurso al que se accede. Extensión. Por ejemplo jpg, gif, html,…
Sesión	Identificador de la sesión de navegación asignado por el servidor web. En el momento de realizar el procesamiento de los logs, será muy importante distinguir las sesiones de trabajo de los usuarios cuyas operaciones se han registrado. Por ello, en este paso del preprocesamiento se extraerá a un nuevo campo con el identificador de sesión que figura en el elemento 'ruta recurso solicitado'.
Parámetros()	Array o tabla de parámetros incluidos en el querystring de la llamada web. De 0 a N elementos
Protocolo	Habitualmente HTTP
Versión protocolo	Habitualmente 1.1
Código respuesta	Código de la respuesta devuelta por el servidor web.
Referer	Dirección web completa del origen de la petición web.
Explorador	Explorador web en donde se está navegando. IE, Firefox, Chrome,…
Tamaño	Tamaño en bytes del recuso solicitado.
Otros	Resto de información no relevante.

2.7.3. Procesamiento previo de los datos

En este punto ya tendremos toda la información de forma estructurada, por lo que el siguiente paso que es la generación del fichero intermedio, sin embargo hay que tener en cuenta que **no toda la información contenida en los logs es interesante de cara al análisis de patrones**, hay que hacer previamente un proceso de limpieza.

Para ello, se interactúa con cada dato, determinando su conveniencia a incluir en el estudio de acuerdo a unos criterios que vamos a describir a continuación. Como salida

de datos tendremos una estructura de memoria gemela a la de entrada, pero con los datos válidos filtrados, es decir, un subconjunto del conjunto de datos de entrada.

Figura 22 Procesamiento previo de los datos.

En la entra de datos log tendríamos los datos tal cual han sido obtenidos de los servidores Apache y como salida de datos estructurados tendríamos una estructura con los datos filtrados según el criterio del usuario. El flujograma relativo al proceso principal (**filtrar datos**) sería el que se detalla a continuación.

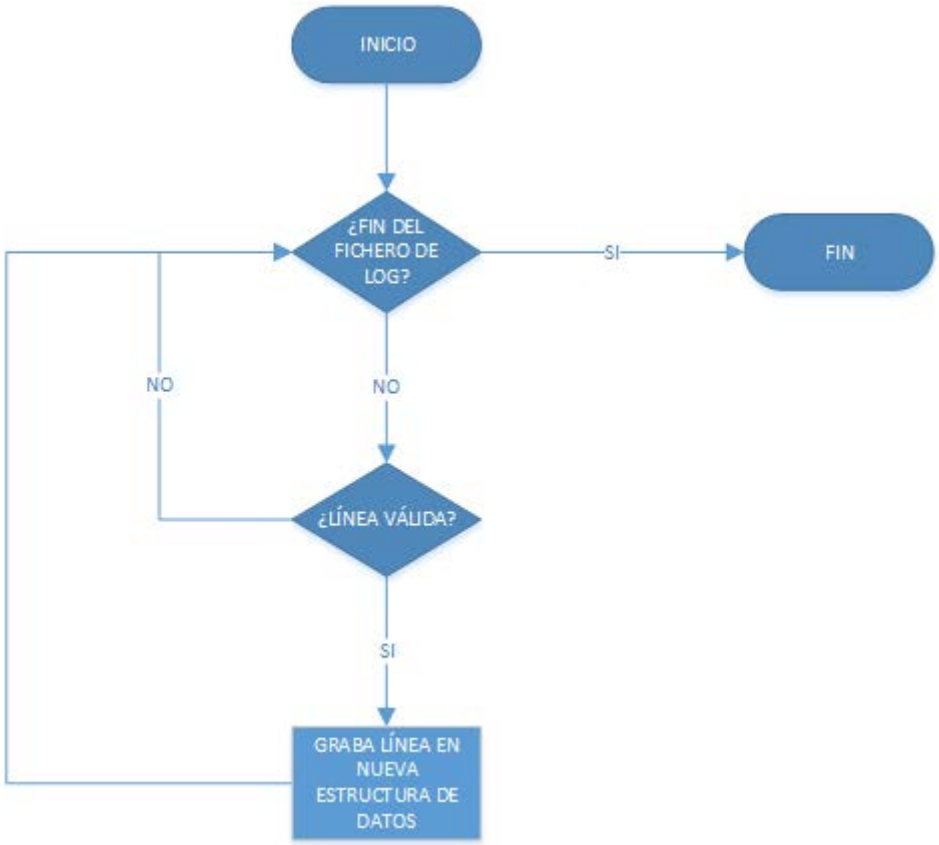


Figura 23 Detalle filtración datos.

Para realizar el procesamiento de los logs, interesará determinar los accesos realizados durante una sesión de trabajo. Los **identificadores de sesión** pueden distinguirse por ir precedidos por los caracteres ?s= o &s= en función de si el parámetro s es el primero o no respectivamente en una determinada url.

Aunque lo deseable es descartar todas aquellas **líneas** en las que el elemento 'ruta recurso solicitado' o 'ruta origen' **no incluya una referencia al identificador de sesión**, el análisis inicial de los ficheros de log ha desaconsejado hacer esto ya que implicaría la pérdida de excesivas líneas. En cualquier caso, no se ha eliminado la referencia a esta actividad para tenerla presente en futuros estudios.

2.7.4. Generar fichero intermedio

El punto de partida de este apartado es el del fichero log (o un subconjunto de él) estructurado, filtradas las líneas inútiles y cargados en memoria. Ahora se deberá, en función de las columnas de salida seleccionadas, generar el fichero intermedio adecuado dependiendo de sus datos. Es decir, el **usuario final de la aplicación podrá seleccionar las columnas de salida de los ficheros tanto intermedios (csv) como finales (arff)** que desee para ser valoradas en la herramienta Weka.

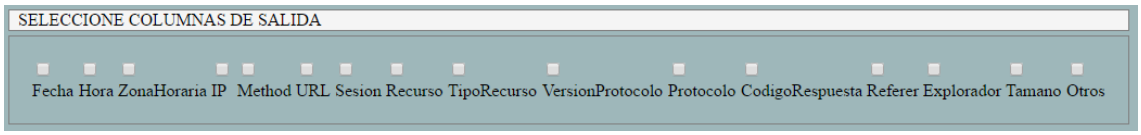


Figura 24 Selección de columnas de salida.

Aun así, destacamos los seis casos, entre todos los posibles, que hemos valorado en nuestro proyecto:

- **Recursos accedidos**.

  Este patrón permitirá identificar a aquellos **recursos solicitados de manera más usual** por los usuarios del campus virtual de la UOC cuyas acciones se han registrado en los logs. Hemos escogido tanto IP como IdSesion ya que nos puede interesar estudiar la frecuencia con la que entra en diferentes recursos desde la misma IP pero con distinta sesión. Así pues relacionará IP-Sesión-Recurso descartando en este caso el resto de elementos. Generando línea por línea y ordenado por identificador de sesión un fichero CSV (y ARFF) con los siguientes campos separados por comas:

  IP, IdSesion, Recurso

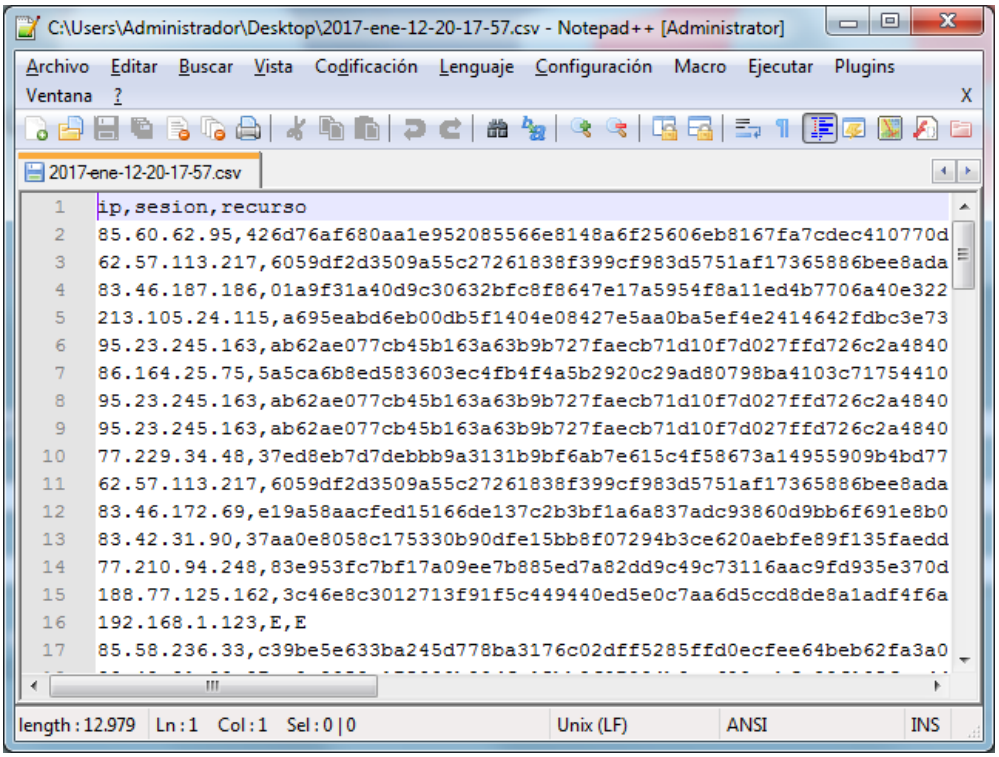


Figura 25 Fichero intermedio IP, Sesión, Recurso

- **Periodos de tiempo de navegación**.

  Este patrón de navegación permitirá identificar los tiempos en los que un usuario accede a los recursos, así como los periodos temporales donde se concentra la mayor cantidad de tráfico.

  Producirá un fichero de datos de exportación de acuerdo a una estructura determinada que se utilizará para introducirlo en el aplicativo Weka (que describiremos más adelante) o como entrada para generar el fichero final. La única diferencia son los datos analizados. La estructura del fichero de salida será la siguiente:

Fecha, Hora, IP, IdSesion

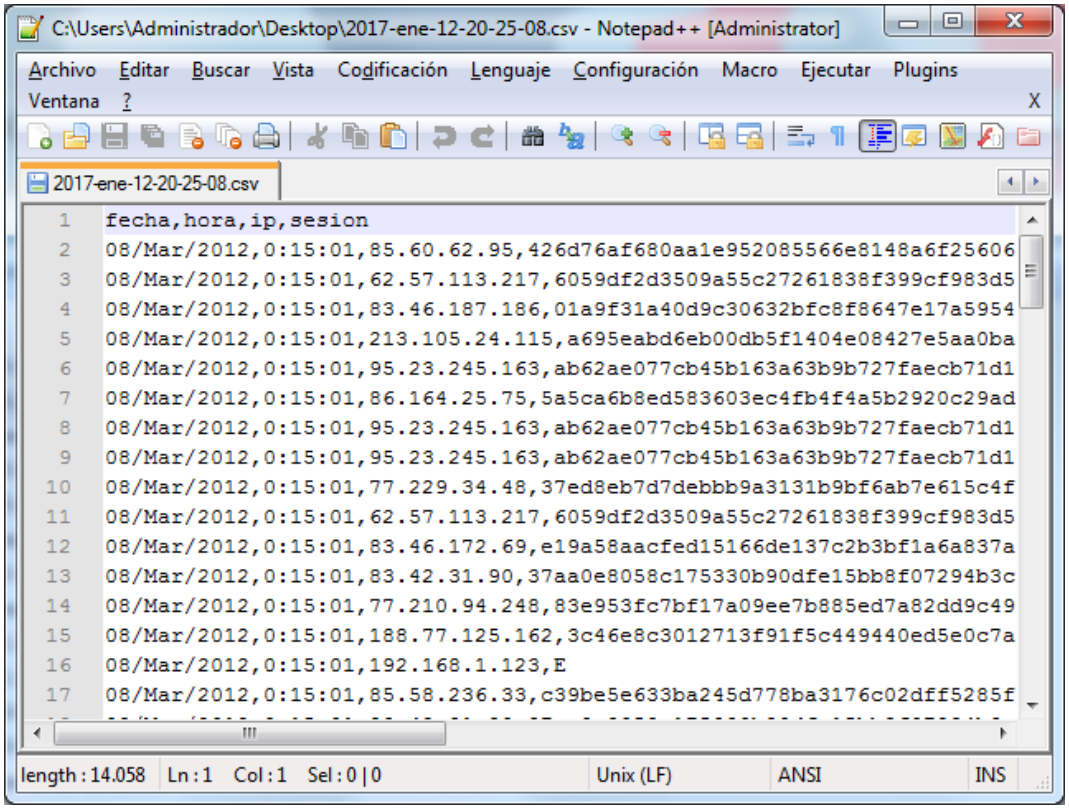


Figura 26 Fichero intermedio Fecha, Hora, IP, Sesión.

- **Respuestas del servidor**.

  Este patrón de navegación permitirá identificar las respuestas del servidor apache del sitio web del Campus de la UOC que obtiene el usuario en una o varias sesiones. Esto tendrá especial interés en caso de los accesos erróneos y poder determinar comportamientos particulares.

La estructura del fichero CSV que generará será la siguiente (ordenado por idSesion):

IP, IdSesion, Código de Respuesta

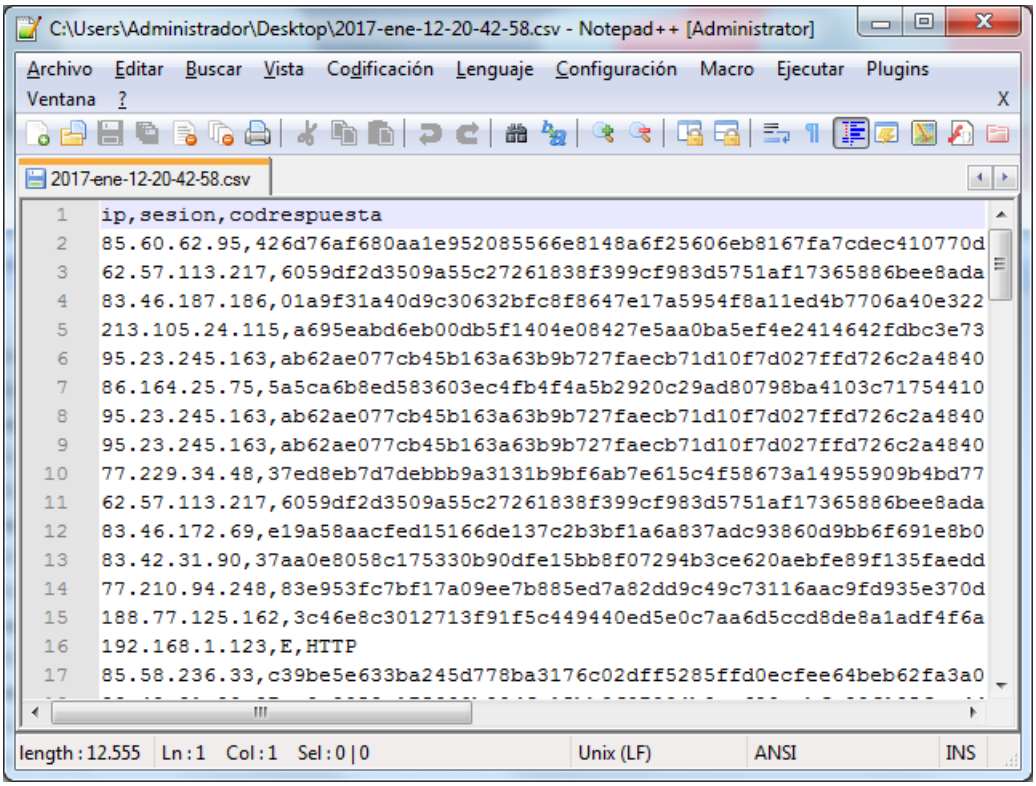


Figura 27 Fichero intermedio IP, Sesión, Código Respuesta.

- **Tipos de recursos accedidos**.

  Este patrón de navegación permitirá determinar los tipos de recursos a los que se accede mayoritariamente en cada dispositivo y en cada sesión.

  La estructura del fichero CSV que generará será la siguiente:

  IP, IdSesion, Tipo de Recuso

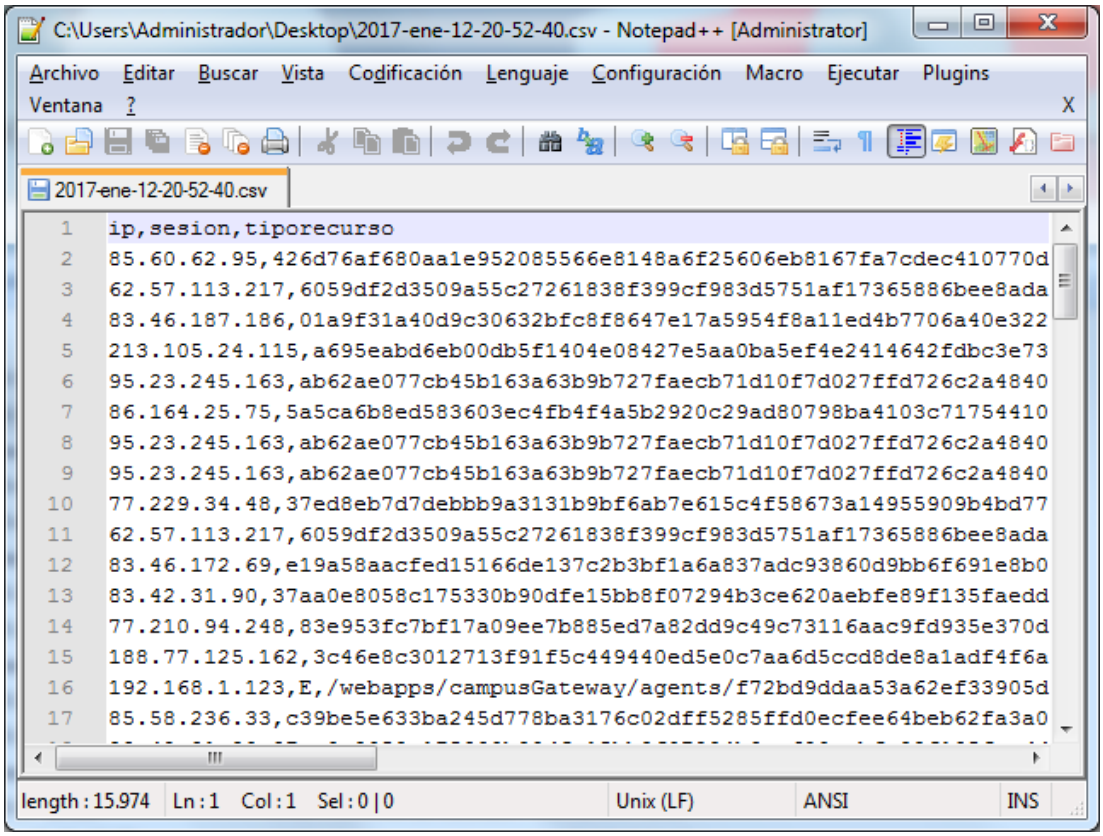


Figura 28 Fichero intermedio IP, Sesión, Tipo Recurso.

- **Exploradores web usados**.

  Este patrón de navegación establecerá los exploradores web más usados y vinculará las diferentes sesiones con diferentes dispositivos con el explorador usado. Tiene especial interés para definir patrones de uso de navegadores web en las diferentes sesiones del usuario.

  La estructura del fichero CSV que generará será la siguiente:

  IP, IdSesion, Explorador web

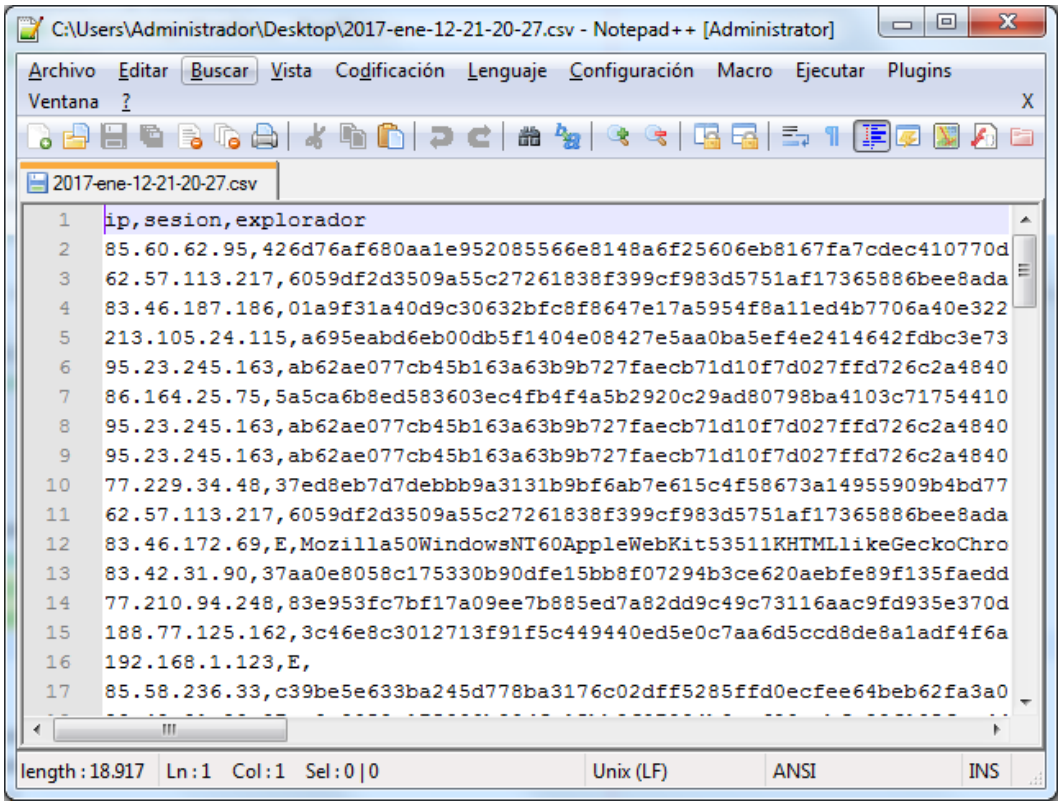


Figura 29 Fichero intermedio IP, Sesión, Explorador

- **Ubicación desde donde se navega.**

  Este patrón de navegación establecerá las ubicaciones geográficas correspondientes a una determinada IP para una sesión en concreto. En este caso es importante tener en cuenta que el dato de la ubicación geográfica **no viene definido en los logs del servidor Apache**, por ello tendremos que diseñar mecanismos que permitan obtener de manera externa este dato a través de la dirección IP. No debemos olvidar que la velocidad de procesamiento es vital por el elevado nº de registros.

  Como opción válida puede estudiarse si son servicios externos o internos. Pero el objetivo es que devuelva una ubicación geográfica a partir de una dirección IP.

  Un posible ejemplo del fichero intermedio generado sería:

  IP, Ubicación

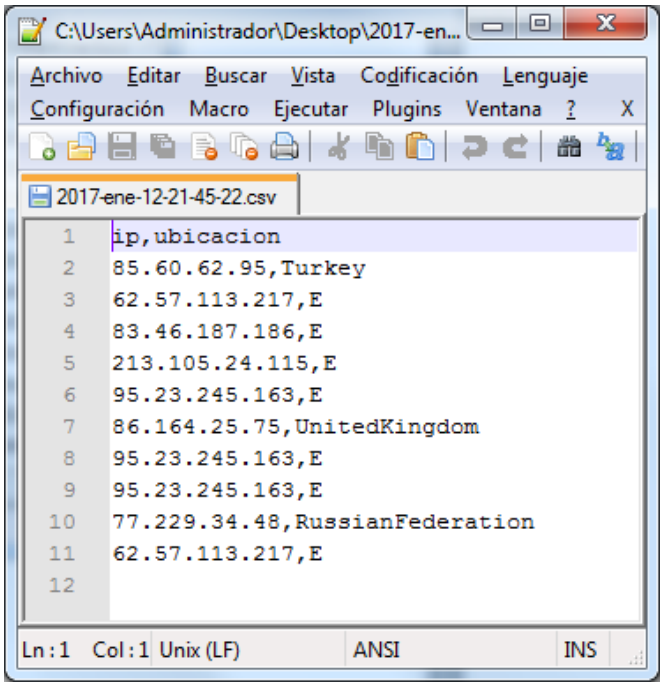


Figura 30 Fichero intermedio IP, Ubicación.

Con el **fichero intermedio generado**, caben dos opciones:

1. Introducirlo en el analizador de datos, Weka, y estudiar los resultados obtenidos.

2. A partir de él, generar el fichero final del tipo Arff.

2.7.5. Generación del fichero final.

Llegados a este punto ya hemos generado el fichero intermedio y hemos decidido generar el fichero final para introducirlo posteriormente en la herramienta de análisis de datos Weka.

A partir del fichero intermedio generado previamente, generaremos otro fichero de exportación de datos, ARFF, para su posterior introducción en WEKA.

Como hemos descrito en capítulos anteriores, este tipo de archivo confecciona lo que podríamos llamar, una matriz con los resultados. A modo de ejemplo, hemos confeccionado cómo debería ser dicho fichero si seleccionamos en nuestra aplicación todas las columnas de salida:

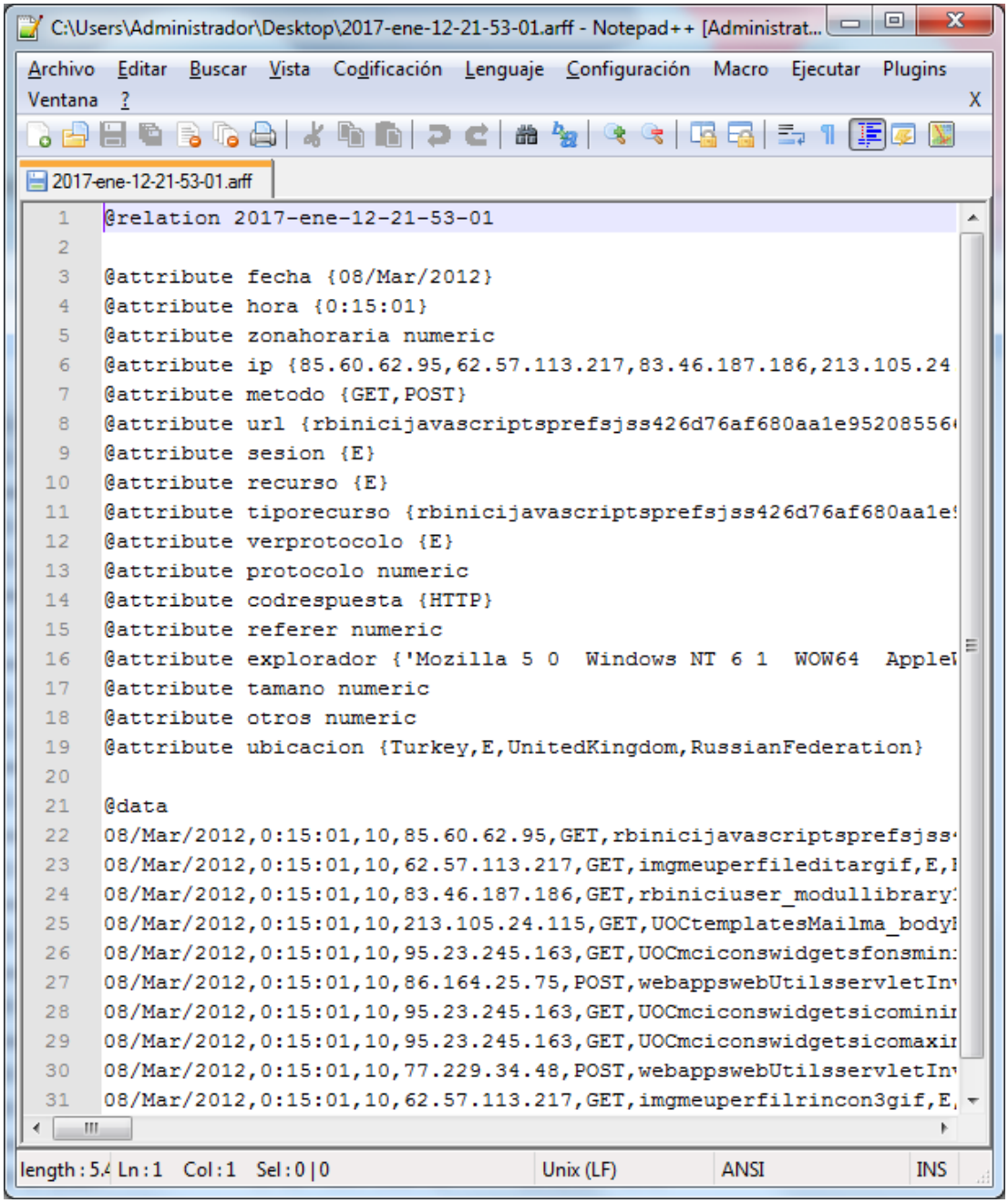


Figura 31 Formato de salida del fichero Arff.

El formato será una matriz de datos con una estructura muy concreta, adecuada a la aplicación destino.

2.7.6. Análisis de datos con Weka.

Como última tarea a realizar ya solo queda analizar los datos obtenidos después del procesamiento previo de la información. Esto se hará, como hemos indicado anteriormente con una herramienta de análisis de datos y modelado predictivo (Weka) que cumple con los requisitos establecidos.

2.8. Casos de uso.

Establecemos como punto partida una secuencia de interacciones entre el sistema y uno o más actores en la que se considera al sistema como una caja negra y en la que los actores obtienen resultados observables.

Los actores son personas u otros sistemas que interactúan con el sistema cuyos requisitos se están describiendo.

Los casos de uso presentan ciertas ventajas sobre la descripción meramente textual de los requisitos funcionales, ya que facilitan la delimitación de requisitos y son fácilmente comprensibles por los clientes y usuarios. Además, pueden servir de base a las pruebas del sistema y a la documentación para los usuarios.

En nuestro caso, tendremos los siguientes casos de uso:

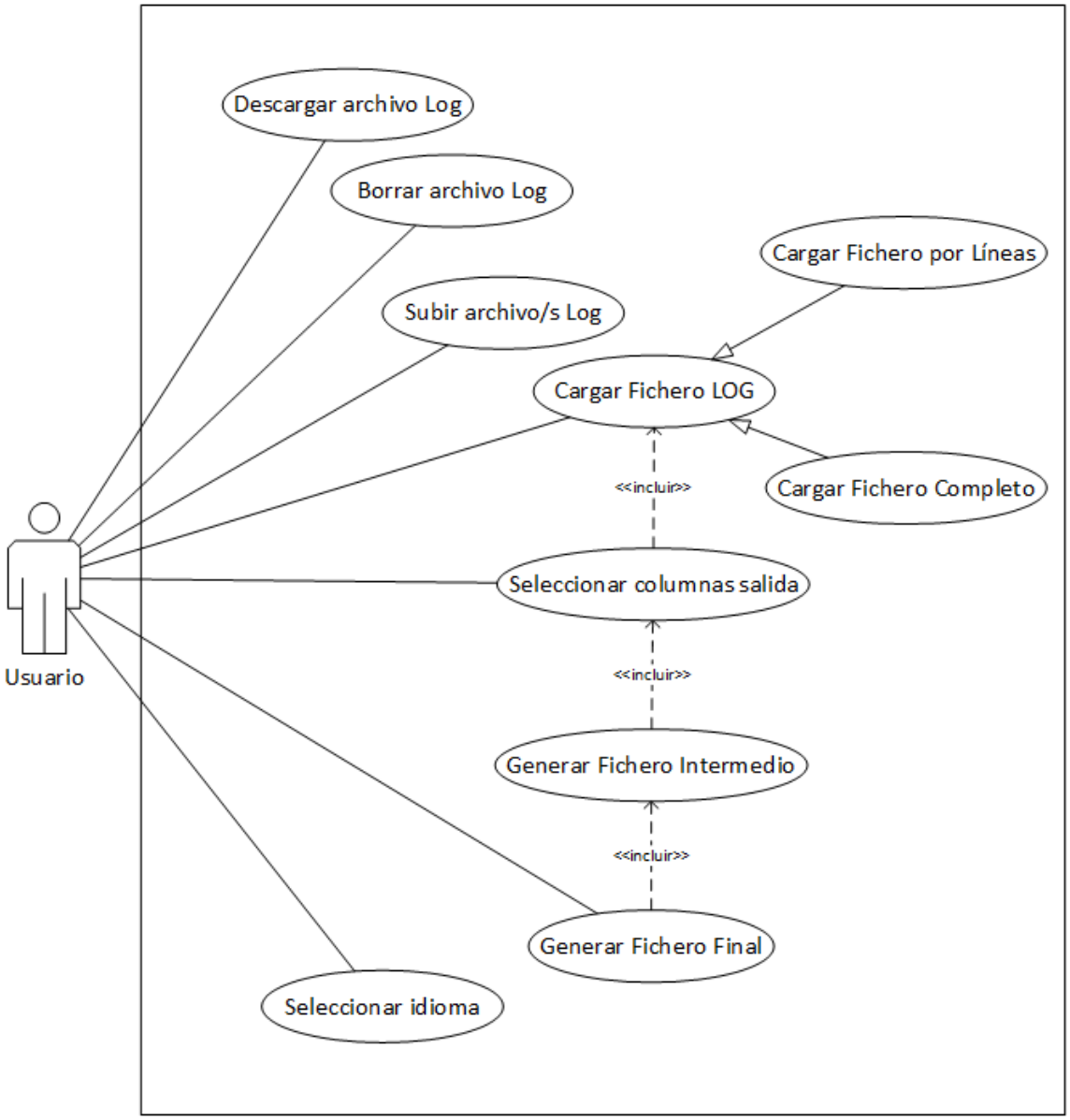


Figura 32 Casos de Uso: Aplicación Completa.

2.8.1. Cargar fichero Log.

El único actor que actuará en todos los casos de uso es aquel usuario que necesite obtener datos estadísticos de las navegaciones de los usuarios a través del portal de la UOC.

Para ello, a partir del fichero LOG que haya generado el servidor Web Apache, se cargará en la aplicación, se procesarán las líneas y se generará un archivo de exportación (CSV y ARFF), que posteriormente se podrán estudiar mediante Weka.

Como veremos, más adelante, sea cual sea la opción de entrada, nos encontramos con el siguiente caso de uso:

CU001 Cargar Fichero LOG		
Actores	Usuario (U)	
Precondición	Fichero con formato correcto	
Descripción	Apertura y carga en memoria de un fichero LOG del Campus Virtual	
Secuencia	**Paso**	**Acción**
	1	El caso de uso comienza cuando (U) pulsa en la opción "Cargar archivo" alojado en la ruta correcta.
	2	El sistema lee el fichero en su totalidad, procesa la información (estructurándola) y guardándola en memoria.
	3	El sistema actualiza el estado del proceso.
Postcondición	La pantalla de la carga muestra el proceso completado	
Excepciones	**Paso**	**Acción**
	2	2.1 La lectura del fichero se ve interrumpida. 2.1.1 El sistema indica error. 2.1.2 Finalizar el caso de uso.

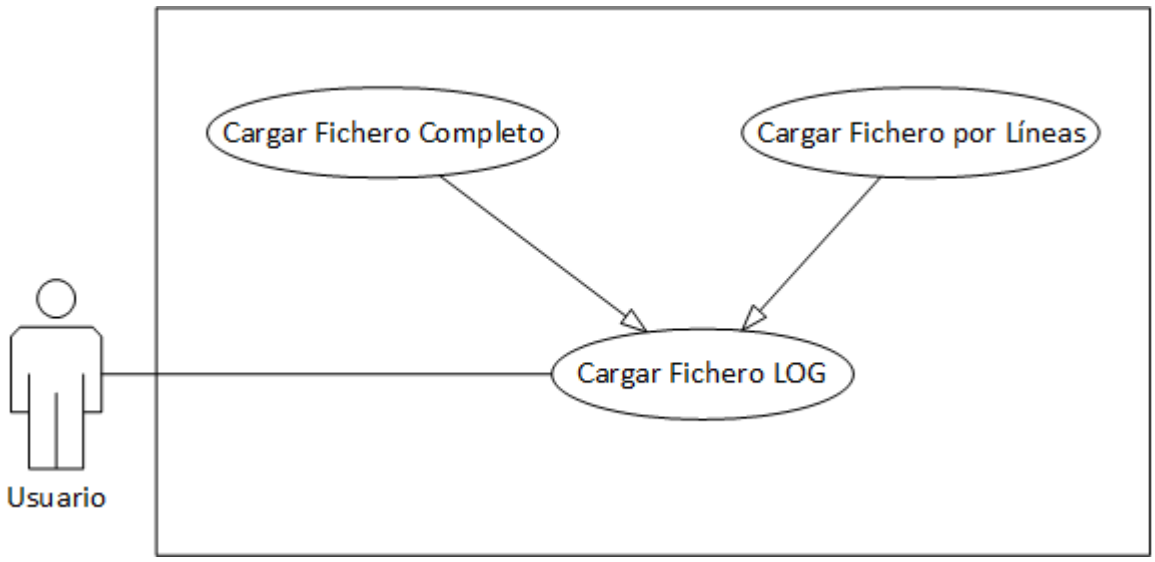


Figura 33 Caso de Uso: Cargar fichero log

Para la carga de un fichero completo, nos encontramos el siguiente caso de uso:

CU002 Cargar Fichero LOG completo		
Actores	Usuario (U)	
Precondición	Fichero con formato correcto	
Descripción	Apertura y carga en memoria de un fichero LOG del Campus Virtual en su totalidad.	
Secuencia	**Paso**	**Acción**
	1	El caso de uso comienza cuando (U) pulsa en la opción "Cargar archivo" alojado en la ruta correcta.
	2	El sistema lee el fichero en su totalidad, procesa la información (estructurándola) y guardándola en memoria.
	3	El sistema actualiza el estado del proceso.
Postcondición	La pantalla de la carga muestra el proceso completado. Además, el fichero quedará estructurado en memoria.	
Excepciones	**Paso**	**Acción**
	2	2.1 La lectura del fichero se ve interrumpida. 2.1.1 El sistema indica error. 2.1.2 Finalizar el caso de uso.

Para una carga por líneas, se sigue el siguiente caso de uso:

CU003 Cargar Fichero LOG por filas		
Actores	Usuario (U)	
Precondición	Fichero con formato correcto	
Descripción	Apertura y carga en memoria de un fichero LOG del Campus Virtual	
Secuencia	**Paso**	**Acción**
	1	El caso de uso comienza cuando (U) pulsa en la opción "Ruta de archivo" alojado en la ruta correcta.
	2	El sistema lee el fichero entre las líneas establecidas, procesa la información (estructurándola) y guardándola en memoria.
	3	El sistema actualiza el estado del proceso.
Postcondición	La pantalla de la carga muestra el proceso completado	
Excepciones	**Paso**	**Acción**
	2	3.1 La lectura del fichero se ve interrumpida. 3.1.1 El sistema indica error. 3.1.2 Finalizar el caso de uso.

2.8.2. Selección de las columnas de salida.

CU004 Seleccionar columnas de salida		
Actores	Usuario (U)	
Precondición	Fichero cargado en memoria correctamente (Cargar fichero Log).	
Descripción	Generación del fichero intermedio.	
Secuencia	**Paso**	**Acción**
	1	El caso de uso comienza cuando (U) selecciona las columnas de salida.
	2	El sistema valida que el usuario haya seleccionado alguna columna.
	3	El sistema muestra las columnas seleccionadas.
	4	El sistema actualiza el estado del proceso.
Postcondición	La pantalla de la carga muestra las columnas seleccionadas.	
Excepciones	**Paso**	**Acción**
	2	2.1 El proceso se ve interrumpido. 2.1.1 El sistema indica error. 2.1.2 Finalizar el caso de uso.

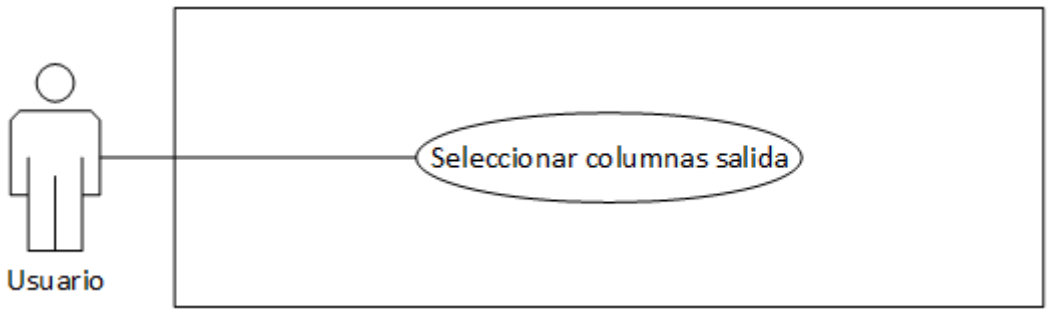


Figura 34 Caso de Uso: Seleccionar Columnas de Salida.

2.8.3. Selección del idioma de la aplicación.

CU005 Seleccionar idioma		
Actores	Usuario (U)	
Precondición	Ninguna.	
Descripción	Selección del idioma de la aplicación	
Secuencia	**Paso**	**Acción**
	1	El caso de uso comienza cuando (U) pulsa en la opción asociada al idioma que desea.
	2	El sistema cambia todos los textos de la aplicación al idioma deseado.
	3	El sistema actualiza el estado del proceso.
Postcondición	La pantalla en la que se encuentra el usuario muestra la información en el idioma deseado (todas las páginas de la sesión de usuario).	
Excepciones	**Paso**	**Acción**
	2	2.1 El proceso se ve interrumpido. 2.1.1 El sistema indica error. 2.1.2 Finalizar el caso de uso.

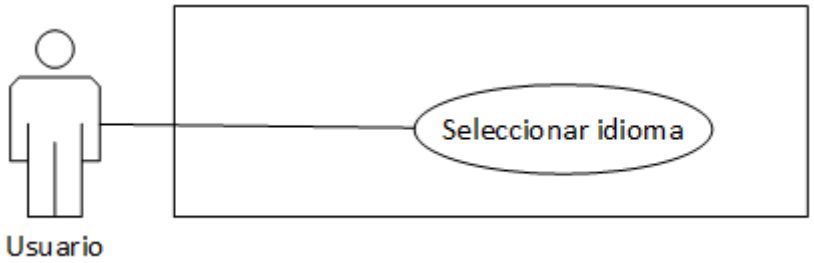


Figura 35 Caso de Uso: Seleccionar Idioma.

2.8.4. Generación del fichero intermedio.

CU006 Generación del fichero intermedio.		
Actores	Usuario (U)	
Precondición	Fichero cargado en memoria correctamente.	
Descripción	Generación del fichero intermedio.	
Secuencia	**Paso**	**Acción**
	1	El caso de uso comienza cuando (U) selecciona la opción para generar el fichero intermedio.
	2	El sistema lee el fichero, procesa la información (estructurándola) y genera el fichero CSV con los datos adecuados a la elección.
	3	El sistema actualiza el estado del proceso.
Postcondición	La pantalla de la carga muestra el proceso completado contemplando la generación del fichero intermedio de exportación (CSV)	
Extensiones	**2.1** El sistema valida la existencia del sistema externo de ubicación.	
	2.2 El sistema recoge del sistema externo la **ubicación** geográfica de la dirección IP si es una de las columnas de salida requeridas.	
Excepciones	**Paso**	**Acción**
	2	21 El proceso se ve interrumpido. 2.1.1 El sistema indica error. 2.1.2 Finalizar el caso de uso.

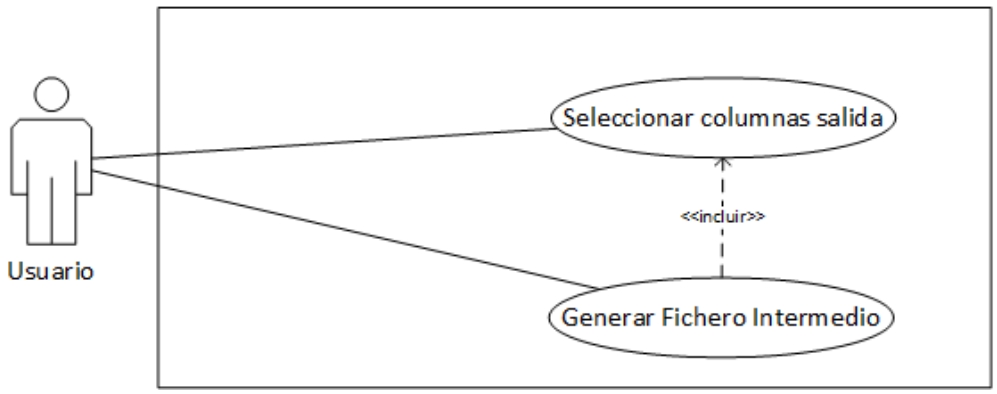


Figura 36 Caso de Uso: Generar Fichero Intermedio.

Por razones de mejor procesamiento de la información, en la etapa de desarrollo, **hemos decidido incluir este caso de uso en el siguiente** (generar fichero final) ya que puede inducir a error por parte del usuario. De esta manera simplificamos la interfaz de usuario de cara a una mejor eficacia del producto final, es decir, solamente tendrá disponible una opción de generar fichero final o Weka en la que se generarán dos tipos de archivos: CSV o intermedio y ARFF o final. En su ejecución, es un proceso secuencial.

2.8.5. Generar fichero final.

CU007 Generar fichero final		
Actores	Usuario (U)	
Precondición	Fichero cargado en memoria correctamente y el fichero intermedio debe estar generado.	
Descripción	Generación del fichero final	
Secuencia	**Paso**	**Acción**
	1	El caso de uso comienza cuando (U) selecciona las opción “generar fichero final o Weka”
	2	El sistema lee el fichero intermedio CSV con los datos adecuados a la elección y genera el fichero final de tipo Weka (Arff).
	3	El sistema actualiza el estado del proceso.
Postcondición	La pantalla de la carga muestra el proceso completado contemplando la generación del fichero final (ARFF)	
Extensiones	**2.1** El sistema valida la existencia del sistema externo de ubicación.*	
	2.2 El sistema recoge del sistema externo la **ubicación** geográfica de la dirección IP si es una de las columnas de salida requeridas.*	
Excepciones	**Paso**	**Acción**
	2	2.1 El proceso se ve interrumpido. 2.1.1 El sistema indica error. 2.1.2 Finalizar el caso de uso.

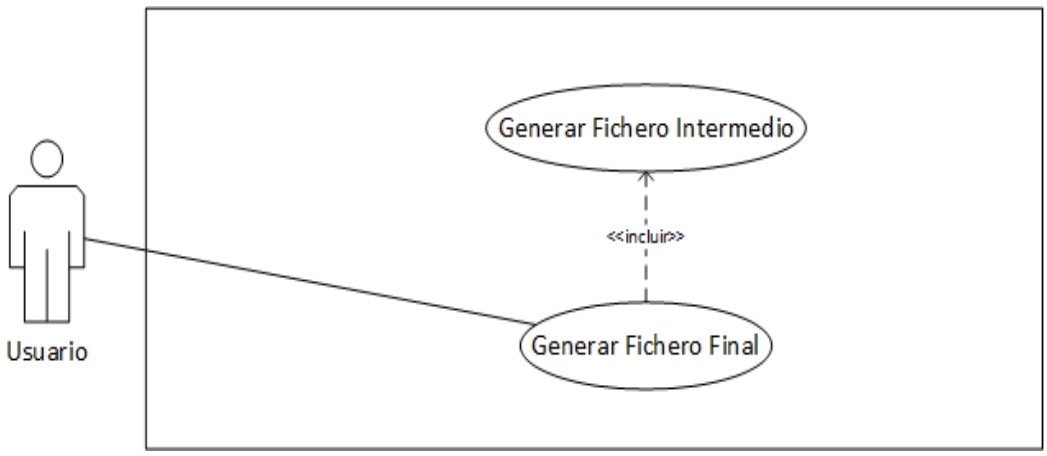


Figura 37 Caso de Uso: Generar Fichero Final.

* Cabe mencionar que en la etapa de desarrollo, hemos decidido usar las extensiones 2.1 y 2.2 solamente en el paso anterior, generación del fichero intermedio, ya que no tiene sentido y penaliza la fluidez del programa, volver a obtener la ubicación geográfica asociada a la línea.

2.8.6. Descargar fichero log.

CU008 Descargar fichero log.		
Actores	Usuario (U)	
Precondición	Al cargar la página de gestión de ficheros log en la nube se muestra un listado paginado de los ficheros almacenados.	
Descripción	Descargar fichero log	
Secuencia	**Paso**	**Acción**
	1	El caso de uso comienza cuando (U) clica sobre el link de un fichero almacenado.
	2	El sistema propone para la descarga http del mismo.
	3	El sistema actualiza el estado del proceso.
Postcondición	El fichero pasa a ser descargado a través del navegador.	
Extensiones	**2.1** El sistema valida la existencia del fichero en destino.	
	2.2 El sistema guarda el fichero con un nombre único.	
Excepciones	**Paso**	**Acción**
	2	2.1 El proceso se ve interrumpido. 2.1.1 El sistema indica error. 2.1.2 Finalizar el caso de uso.

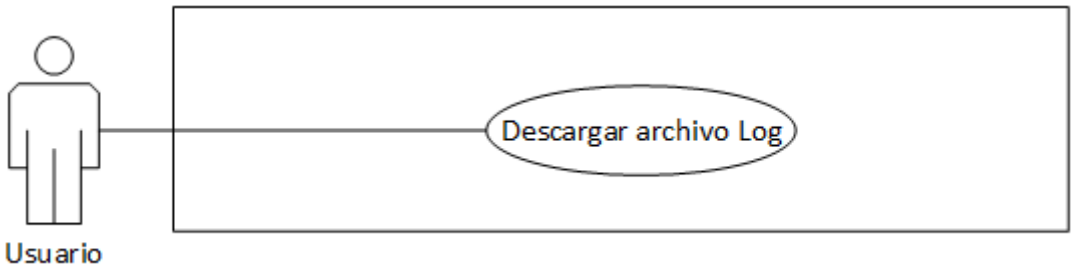


Figura 38 Caso de Uso: Descargar Archivo Log.

2.8.7. Borrar fichero log.

CU009 Borrar fichero log		
Actores	Usuario (U)	
Precondición	Al cargar la página de gestión de ficheros log en la nube se muestra un listado paginado de los ficheros almacenados.	
Descripción	Borrar fichero log	
Secuencia	**Paso**	**Acción**
	1	El caso de uso comienza cuando (U) selecciona la opción “borrar” asociada a uno de los ficheros.
	2	El sistema borra el fichero en el alojamiento remoto.
	3	El sistema actualiza el estado del proceso y refresca la página.
Postcondición	El fichero pasa a ser borrado.	
Excepciones	**Paso**	**Acción**
	2	2.1 El proceso se ve interrumpido. 2.1.1 El sistema indica error. 2.1.2 Finalizar el caso de uso.

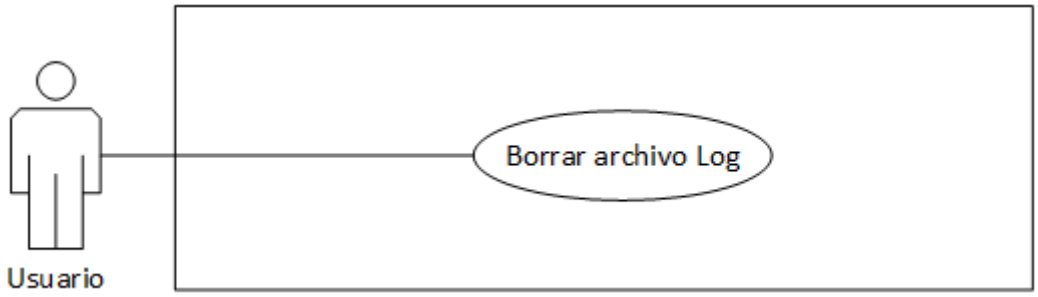


Figura 39 Caso de Uso: Borrar Archivo Log.

2.8.8. Subir fichero log.

CU010 Subir fichero log		
Actores	Usuario (U)	
Precondición	(U) selecciona uno o varios ficheros para subir al almacenamiento.	
Descripción	Subir fichero log	
Secuencia	**Paso**	**Acción**
	1	El caso de uso comienza cuando (U) selecciona los ficheros log a subir.
	2	El sistema va subiendo uno a uno hasta completar la operación.
	3	El sistema actualiza el estado del proceso y refresca la página.
Postcondición	Los ficheros aparecen en el almacenamiento en la nube.	
Excepciones	**Paso**	**Acción**
	2	2.1 El proceso se ve interrumpido. 2.1.1 El sistema indica error. 2.1.2 Finalizar el caso de uso.

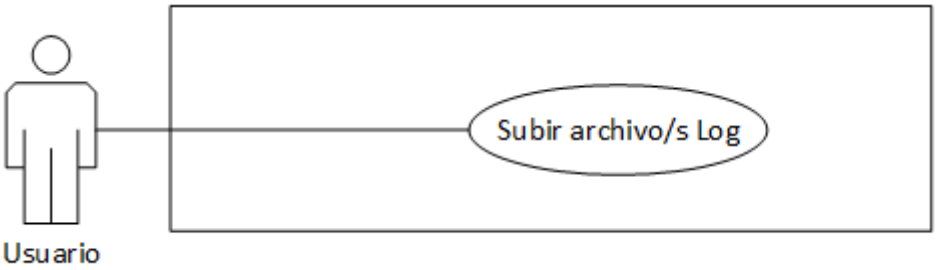


Figura 40 Caso de Uso: Subir Fichero/s Log

2.9. Alcance tecnológico.

Aunque parezca obvio el principal objetivo de nuestro desarrollo debe permitir al usuario estudiar los patrones de navegación dentro del Campus de la UOC.

Además, el proyecto debe cumplir los siguientes objetivos dentro de su alcance tecnológico:

2.9.1. Flexibilidad para modificar la arquitectura.

En este punto es cuando entra en juego **Cloud Computing** ya que consiste en una colección de ordenadores virtualizados e interconectados que son suministrados dinámicamente y presentados como uno o más recursos computacionales unificados, conforme acuerdo de nivel de servicio negociado entre el proveedor de servicios y el consumidor (en este caso, nosotros).

Dado nuestro alcance funcional, es necesario contar con una **escalabilidad elevada** ya que la flexibilidad suministrada por este modelo permite que las aplicaciones adquieran más recursos dinámicamente para alojar sus servicios, a fin de lidiar con los picos de trabajo y, de la misma forma, liberarlos cuando la carga disminuye. La modificación de los recursos puede ser realizada de forma manual (por una interfaz web o a través de la línea de comandos) y vía programación (a través de un software que ajusta automáticamente la capacidad de atender la demanda real), lo que representa una gran ventaja sobre el modelo tradicional de computación.

El punto más fuerte se encuentra en el dimensionamiento dinámico que puede ocurrir de dos formas: (I) **Pro-activo**, en el que, con base en la demanda proyectada, los cambios en la infraestructura se realizan en una(s) concreta(s); y (II) **Reactivo**, en que la propia infraestructura reacciona añadiendo y eliminando capacidad de acuerdo a los cambios en la demanda.

Dado que el plan de pruebas es crítico en este proyecto será necesario contar con gran **agilidad** para acometerlo. Será determinante que nuestra plataforma cloud suministre una amplia infraestructura permitiendo a los usuarios de esos servicios realizar cambios, experimentar e interactuar con agilidad, sin tener la preocupación de la adquisición o la mejora de la infraestructura.

Otro punto a tener en cuenta es que nuestro alcance tecnológico **no debe contemplar la necesidad de inversiones anticipadas**. La creación de un nuevo proyecto se hace mucho más simple con el uso de entornos de Cloud Computing. El modelo elimina los costes de hardware, permitiendo que ganemos agilidad para ejecutar nuestra aplicación, pagando un precio de acuerdo con la cantidad de recursos utilizados.

Será necesaria una **infraestructura Self-Service dentro de nuestra plataforma en la nube**, por lo que tendremos que poder añadir a nuestra infraestructura nuevos servidores, clusters,… de forma mucho más ágil que en el contexto tradicional y sus costosas etapas de creación de un nuevo Data Center.

Deberá contar con una **alta disponibilidad** ya que nuestra aplicación puede requerir una demanda inesperada o por encima de lo previsto inicialmente por lo que es un escenario en los que la utilización de servicios de Cloud Computing puede ser interesante.

2.9.2. El sistema diseñado siguiendo estándares abiertos.

El software de desarrollo debe distribuido libremente sin costes y la licencia no debe ser específica de un producto: el programa no puede licenciarse sólo como parte de una distribución mayor.

La licencia no debe restringir otro software por lo que no puede obligar a que algún otro software que sea distribuido con el software abierto deba también ser de código abierto.

2.9.3. Arquitectura en tres capas.

El objetivo primordial es la separación de la lógica de negocios de la lógica de diseño; un ejemplo básico de esto consiste en separar la capa de datos de la capa de presentación al usuario. La ventaja principal de este estilo es que el desarrollo se puede llevar a cabo en varios niveles y, en caso de que sobrevenga algún cambio, solo se ataca al nivel requerido sin tener que revisar entre código mezclado. Un buen ejemplo de este método de programación sería el modelo de interconexión de sistemas abiertos.

Además, seguiremos el modelo **Model-View-Controller (MVC)** que es un principio de diseño arquitectónico que separa los componentes de una aplicación web. Esta separación ofrece más control sobre las partes individuales de la aplicación, lo cual permite desarrollarlas, modificarlas y probarlas más fácilmente.

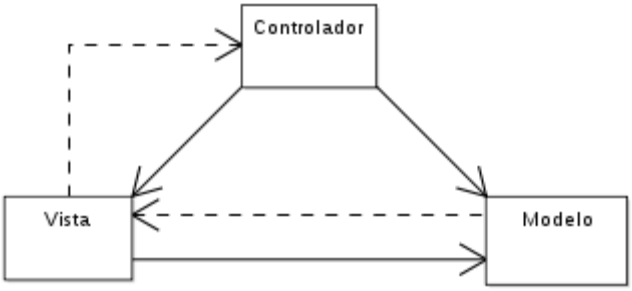


Figura 41 Estructura básica MVC

En ese mismo sentido, MVC define en qué bloques (o capas) estructuramos lógicamente nuestra aplicación (Modelo, Vista y Controlador), pero además detalla las responsabilidades exactas de cada capa y la forma que tienen de relacionarse entre sí. Es básicamente una forma más estructurada de encarar un desarrollo.

3. Desarrollo proyecto.

3.1. Diseño de la solución.

Debemos contar con una solución que no contemple la necesidad de inversiones anticipadas en licencias para el desarrollo de nuestro proyecto por lo que hemos establecido como punto de partida la **arquitectura java para su desarrollo**.

3.1.1. Java Server Faces – Modelo Vista Controlador.

En esa misma línea, comenzamos nuestro diseño con el uso de **Java Server Faces (JSF)** ya que es una tecnología y framework para aplicaciones Java basadas en web que simplifica el desarrollo de interfaces de usuario en aplicaciones Java EE. JSF usa Java Server Pages (JSP) como la tecnología que permite hacer el despliegue de las páginas, pero también se puede acomodar a otras tecnologías como XUL (acrónimo de XML-based User-interface Language, lenguaje basado en XML para la interfaz de usuario).

JSF incluye un conjunto de APIs para representar componentes de una interfaz de usuario (**conjunto por defecto de componentes para la interfaz de usuario**) y administrar su estado, manejar eventos, validar entrada, definir un esquema de navegación de las páginas y dar soporte para internacionalización y accesibilidad.

Además, podemos usar dentro de JSF el framework MVC (Modelo-Vista-Controlador) para gestionar las aplicaciones web, esto es:

- **Vista**.
  Usaremos ficheros XHTML con etiquetas especiales que definen los componentes JSF. Estos componentes se convierten al final en código HTML (incluyendo JavaScript) que se pasa al navegador para que lo muestre al usuario. El navegador es el responsable de gestionar la interacción con el usuario.

En el siguiente ejemplo podemos ver cómo son las primeras líneas del fichero **GenerarFichWeka.xhtml**.

```
<?xml version='1.0' encoding='ISO-8859-1' ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/T
<html lang="#{language.language}"
      xmlns="http://www.w3.org/1999/xhtml"
      xmlns:ui="http://xmlns.jcp.org/jsf/facelets"
      xmlns:f="http://xmlns.jcp.org/jsf/core"
      xmlns:h="http://xmlns.jcp.org/jsf/html"
      xmlns:p="http://primefaces.org/ui">
    <f:view locale="#{language.locale}">
        <h:head class="header">
            <title>#{msg['titulo_uoc']}</title>
            <h:outputStylesheet name="style.css"/>
            <script type="text/javascript">
                function GetInputValue() {
                    if (document.getElementById("file") !== null) {
                        var input = document.getElementById("file");
                        var output = document.getElementById("rutafich");
                        var filename = input.value;
                        filename = filename.replace("\\fakepath", "");
                        output.innerHTML = filename;
                    }
                }
                function validarnumero(numero) {
                    if (!/^([0-9])*$/.test(numero))
                        alert(numero + " #{msg['noesnumero']}");
                }
            </script>
        </h:head>
        <h:body>
            <h:form id="myform" enctype="multipart/form-data" prependId="false">
                <div id="box">
```

Figura 42 Ejemplo de código xhtml

Un aspecto muy importante de JSF es la conexión de las vistas con la aplicación mediante los denominados beans gestionados. Dicha conexión entre las clases Java y las páginas JSF se realiza mediante el Lenguaje de Expresiones JSF (JSF EL), una versión avanzada del lenguaje de expresiones de JSP. Con este lenguaje, podemos definir conexiones (*bindings*) entre las propiedades de los beans y los valores de los componentes que se muestran o que introduce el usuario.

En la figura 40 hemos podido ver cómo la vista utiliza el bean *language*. Como veremos a continuación, se usa para definir la variable *locale* (idioma seleccionado por el usuario) a lo largo del tiempo en el que esté activa la sesión de usuario. Más concretamente, se define un *binding* entre el componente **f:view** y la propiedad **locale** del bean **language** mediante la expresión:

<f:view locale="#{language.locale}">

De esta forma, el idioma seleccionado por el usuario se guardará en esa propiedad del bean y podremos utilizarlo en otro momento de la sesión.

- **Modelo: beans gestionados.**
  No deja de ser una clase con un conjunto de atributos (denominados propiedades) y métodos getters y setters que devuelven y actualizan sus valores. Las propiedades del bean se pueden leer y escribir desde las páginas JSF utilizando el lenguaje de expresiones EL.

  Por ejemplo, en la anterior página GenerarFichWeka.xhtml se utiliza el bean ***LanguageBean*** para mostrar el idioma (***locale***) establecido por el usuario. La definición del bean es la siguiente:

```
package cdm;

import java.io.Serializable;
import java.util.Locale;
import javax.faces.bean.ManagedBean;
import javax.faces.bean.SessionScoped;
import javax.faces.context.FacesContext;

@ManagedBean(name="language")
@SessionScoped
public class LanguageBean implements Serializable{

        private static final long serialVersionUID = 1L;
        private Locale locale = FacesContext.getCurrentInstance().getViewRoot().getLocale();

    public Locale getLocale() {
        return locale;
    }
    public String getLanguage() {
        return locale.getLanguage();
    }
    public void changeLanguage(String language) {
        locale = new Locale(language);
        FacesContext.getCurrentInstance().getViewRoot().setLocale(new Locale(language));
    }
}
```

Figura 43 Ejemplo de Bean gestionado.

En las expresiones JSF EL de la página en la que se usa el bean se puede acceder a sus propiedades utilizando el nombre de la propiedad. Si la expresión es sólo de lectura se utilizará internamente el **método get** para recuperar el contenido del bean y mostrarlo en la página. Si la expresión es de **lectura y escritura, se utilizará además el set** para modificarlo con los datos introducidos por el usuario de la página.

Es importante resaltar que es posible utilizar la inicialización de propiedades para crear relaciones entre distintos beans, almacenando un bean en una propiedad de otro. Esto es muy útil para **implementar correctamente un modelo MVC**.

- **Controlador.**

Debemos diferenciar dos tipos de acciones, las acciones del **componente** y las acciones de la **aplicación**. En el primer tipo de acciones es el propio componente el que contiene el código (HTML o JavaScript) que le permite reaccionar a la interacción del usuario. Es el caso, por ejemplo, de la función ***validarnumero*** que hemos visto en la figura 40. Para validar si el usuario ha introducido un número en el campo "línea" no es necesario usar lógica en el lado del servidor, es decir, no hay ninguna petición al controlador de la aplicación para obtener datos o modificar algún elemento, sino que toda la interacción la maneja el propio componente. Con JSF es posible utilizar eventos JavaScript para configurar este comportamiento.

Las **acciones de la aplicación** son las que determinan las funcionalidades de negocio de la aplicación. Se trata de código que queremos que se ejecute en el servidor cuando el usuario pulsa un determinado botón o pincha en un determinado enlace. Este código realizará llamadas a la capa de negocio de la aplicación y determinará la siguiente vista a mostrar o modificará la vista actual.

Como hemos visto anteriormente, las acciones se definen en beans gestionados de la página JSF. Son métodos del bean que se ligan al elemento ***action*** del componente que vaya a lanzar esa acción.

Por ejemplo, en la página GenerarFichWeka.xhtml tenemos un ejemplo claro. En el momento en que el usuario pinche en uno de los dos iconos asociados a los idiomas disponibles, se lanzará el método ***changeLanguage*** del bean ***language***.

```
<tr>
    <td><h:commandButton action="#{language.changeLanguage('')}"
                         value="English" image="resources/imagenes/icon/uk.png" />
        <h:outputText value=" " />
        <h:commandButton action="#{language.changeLanguage('es')}"
                         value="Spanish" image="resources/imagenes/icon/es.png" /></td>
</tr>
```

Figura 44 Ejemplo de Controlador.

Al tratarse de un entorno web y orientado a un entorno más cerrado (**intranet**), la tecnología JSF cobra importancia en nuestro desarrollo ya que la simplicidad que nos entrega JSF es muy alta y claramente permite mantener **alto enfoque en la solución y no en la tecnología usada**. En nuestro caso, **modelo y controlador estarán constituidos en la misma clase** ya que no es un diagrama de clases altamente complejo que conlleve una separación obligada de funcionalidades para su mejor mantenimiento.

Por último, debemos contemplar dos alternativas para cargar los ficheros log de origen:

1. Desde el lado del cliente
2. Desde un repositorio (Amazon S3).

3.2. Localizador de ubicación de IP.

Como vimos en la fase de análisis, una de las salidas posibles a evaluar es la ubicación de la dirección IP almacenada en cada línea del fichero Log origen. Es por ello que precisamos de un servicio o funcionalidad que nos devuelva dicha ubicación pero sin olvidar las condiciones y decisiones tomadas respecto a la calidad del software, protocolos abiertos, flexibles…

Además, deberá ser parametrizado para que un pueda ser desplegado tanto en el entorno valorado (entorno en la nube) como en un sistema físico del que dispongamos.

Después de valorar diversas opciones, nos hemos decantado por usar un sistema en forma de BBDD al que accederemos para obtener dicha información. De esta manera podremos indexar la obtención de la información que nos interese para ganar en velocidad en la consulta. Actualmente en el mercado existen diferentes opciones pero nos hemos decantado por la siguiente opción:

www.ip2location.com

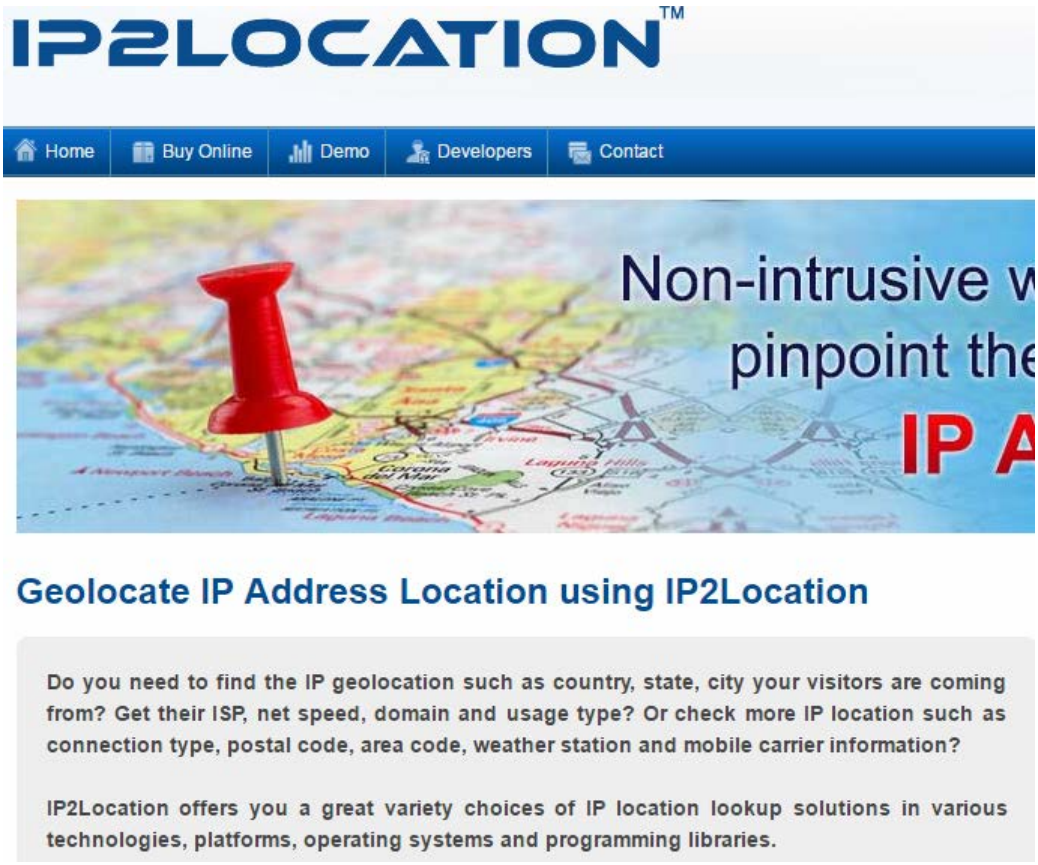


Figura 45 Página principal de Ip2Location.

Se trata de una web desde donde podemos descargar una base de datos MySql o Sql Server con los diferentes rangos de IPs asociadas a cada ubicación geográfica mundial. Obviamente, hemos escogido la versión MySql para llevarlo a cabo por seguir los requisitos de desarrollo marcados para este proyecto. Descargaremos la **versión LITE de las base de datos** ya que es una versión gratuita pese a tener una precisión limitada y número de registros inferior si se compara con la liberación comercial (cuyo coste sería de unos 650€).

Como podemos ver en la siguiente tabla, el modelo de datos es sumamente sencillo por lo que será fácilmente abordable su implementación.

Ip2location_db11		
Pk	Ip_from	Ip de inicio de rango de IPs
Pk	Ip_to	Ip de fin de rango de IPs
	Country_code	Código del País. ES, GB, FR, …
	Country_name	Nombre del País
	Region_name	Región dentro del país
	City_name	Ciudad
	Latitude	Latitud donde se encuentra la IP
	Longitude	Longitud donde se encuentra la IP
	Zip_code	Código postal
	Time_zone	Diferencia horaria según GMT

Como veremos en nuestro entorno de desarrollo más tarde, una vez creada y definida la base de datos, realizaremos la carga del fichero csv descargado cuyo formato es el siguiente.

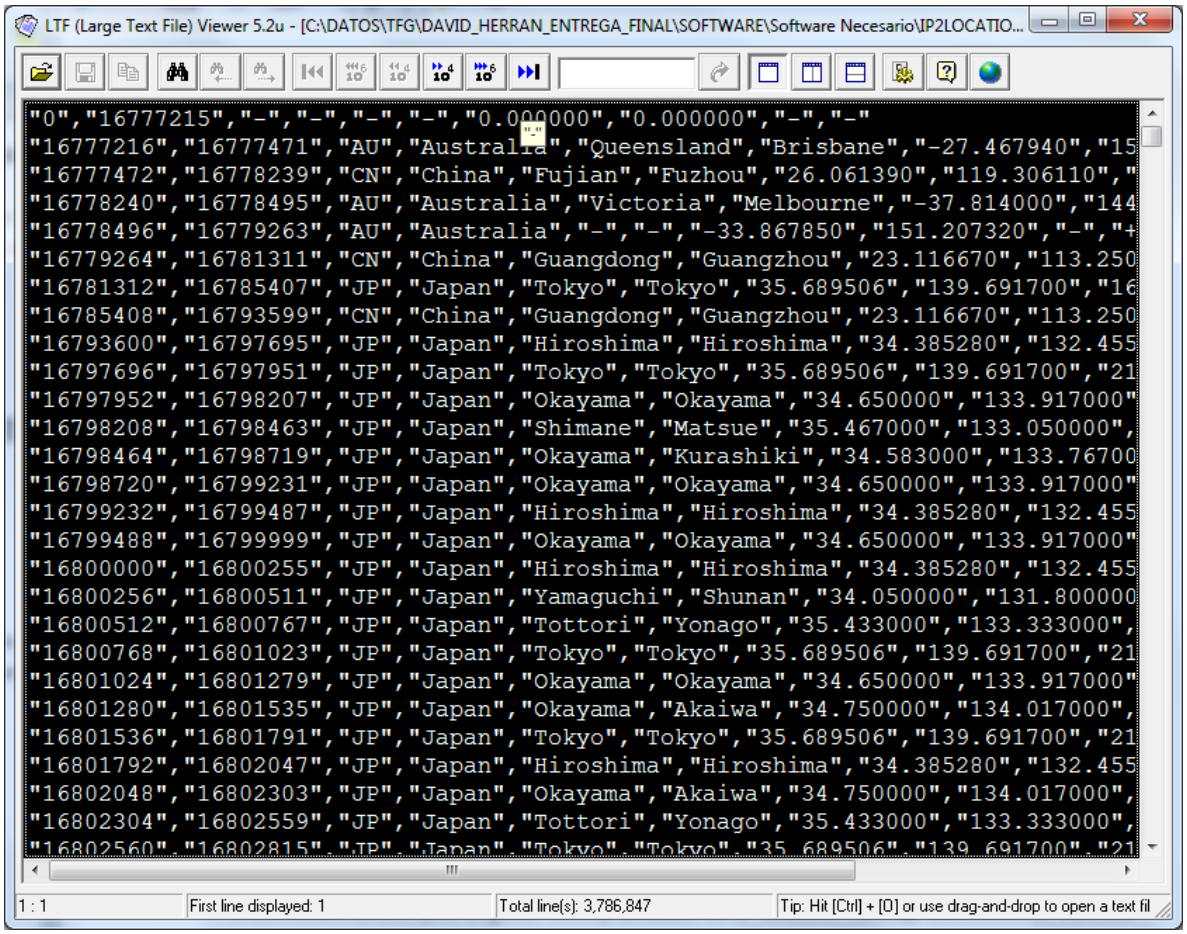


Figura 46 Formato fichero Ip2Location

El flujo de trabajo es muy sencillo ya que simplemente tendremos que formatear el campo IP recibido del log Apache (quitar puntos de la expresión), consultar la base de datos y registrar la ubicación encontrada.

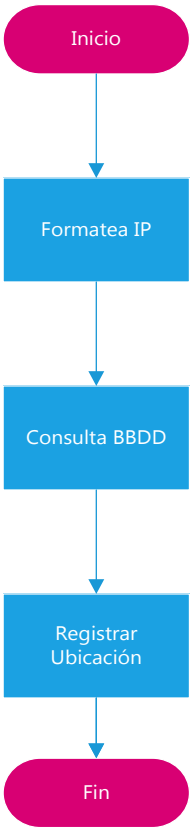


Figura 47 Flujo trabajo Ip2Location

Una vez creada la tabla y cargado el fichero a través de sentencias DDL, tendremos definido el servicio de localización geográfica necesario.

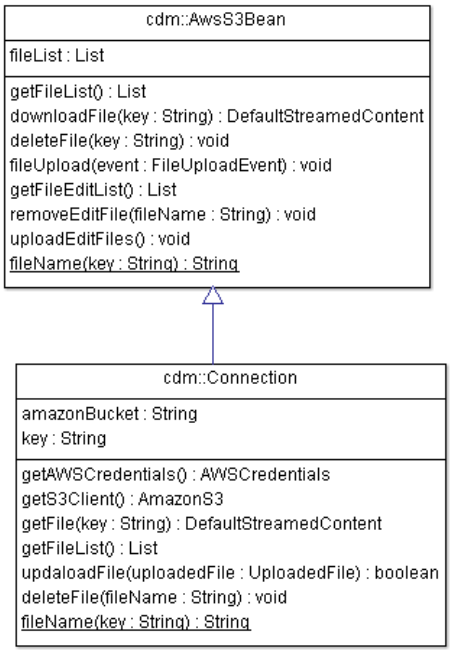
```
CREATE DATABASE cloudtaminingdb;
USE ip2location;
CREATE TABLE `ip2location`(
    `ip_from` INT(10) UNSIGNED,
    `ip_to` INT(10) UNSIGNED,
    `country_code` CHAR(2),
    `country_name` VARCHAR(64),
    `region_name` VARCHAR(128),
    `city_name` VARCHAR(128),
    `latitude` DOUBLE,
    `longitude` DOUBLE,
    `zip_code` VARCHAR(30),
    `time_zone` VARCHAR(8),
    INDEX `idx_ip_from` (`ip_from`),
    INDEX `idx_ip_to` (`ip_to`),
    INDEX `idx_ip_from_to` (`ip_from`, `ip_to`)
) ENGINE=MyISAM DEFAULT CHARSET=utf8 COLLATE=utf8_bin;

LOAD DATA LOCAL
    INFILE 'C:\datos\tfg\mysql\IP2LOCATION-LITE-DB11.CSV'
INTO TABLE
    `ip2location`
FIELDS TERMINATED BY ','
ENCLOSED BY '"'
LINES TERMINATED BY '\r\n';
```

Figura 48 Script Ip2Location.

3.2.1. Diagrama de clases.

Nuestro diseño de clases dentro de NetBeans contempla las siguientes clases de gestionarán todos nuestros procesos:



- **Connection.**

  Se encargará de establecer la conexión con el repositorio del servicio Amazon S3.

  Cabe destacar el ***métodoUpdaLoadFile()*** que será capaz de enviar un fichero seleccionado por el usuario en el sistema de archivos local. Para ello primero tendrá que constituir el llamado "clienteAS3" con las funcionalidades necesarias, entre ellas, las credenciales necesarias para acceder a la plataforma AWS. Todas las operaciones serán parametrizadas por la clase AWSS3Bean.

- **AwsS3Bean**.

  Contempla todas las operaciones relacionadas con el servicio S3: listado de objetos, subida, borrado y descarga.

  Merece la pena pararse en el método ***removeEditFile (),*** donde de manera sencilla, el objeto se encarga de mantener el listado de objetos almacenados en remoto, dejando la complejidad de la operaciones al cliente AS3 definido en el objeto conexión. AWS es una plataforma que

permite, con relativamente poco código de desarrollo, poder realizar implementaciones complejas.

```
private List<UploadedFile> fileList = new ArrayList<UploadedFile>();

public void removeEditFile(String fileName){
    for(int i = 0; i < this.fileList.size(); i++){
      if(this.fileList.get(i).getFileName().equals(fileName)){
        this.fileList.remove(i);
        return;
      }
    }
  }
```

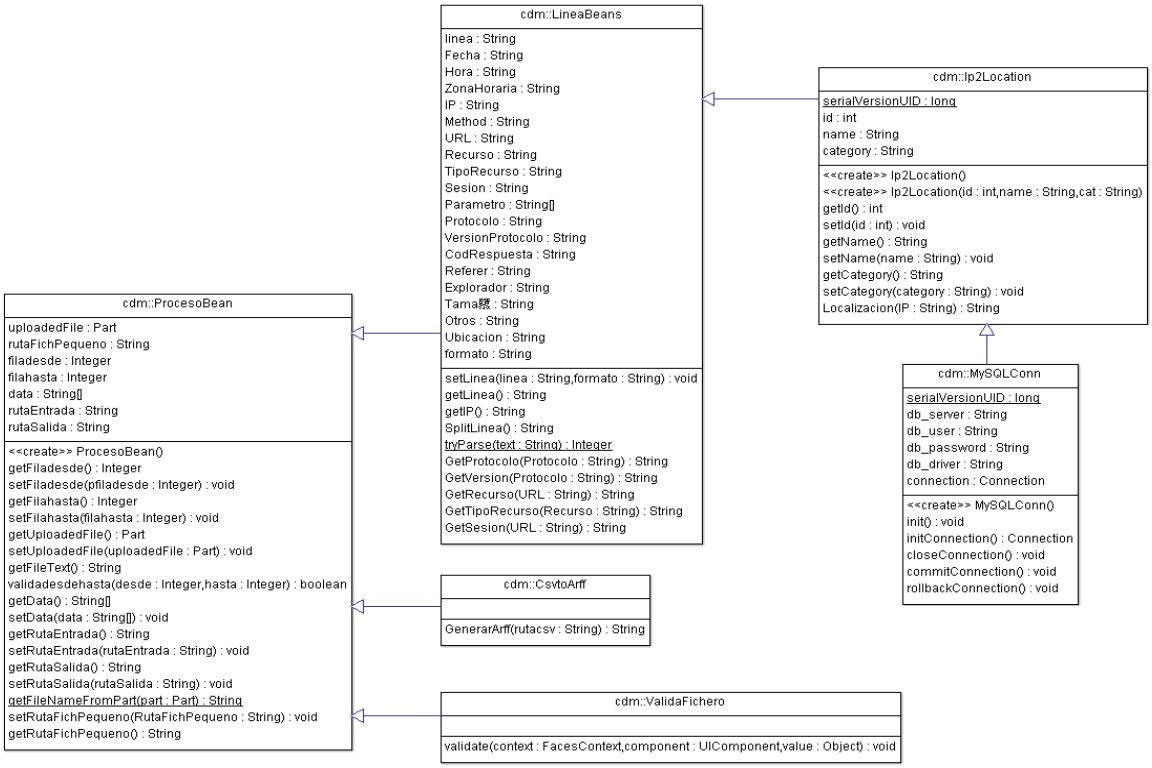


Figura 49 Clases Proceso Weka

- **ProcesoBean**.

  Es nuestra **clase principal**. Se encargará de gestionar, llamar y recoger las respuestas de las operaciones más importantes: leer fichero Log, crear fichero resultado así como los errores encontrados.

  Como método principal nos encontramos ***getFileText()***, donde se realizarán

las validaciones de la carga de fichero log apache, esto es, fichero **completo** o **ruta** de fichero en el sistema de archivos local. Además dicho método validará cada línea a través del objeto **LineaBeans** (dependiente de él) y generará el fichero intermedio CSV y el final ARFF (creación de un objeto ***CsvtoArff***)

- **LineaBeans**: Clase dependiente de ProcesoBean encargada de procesar una línea a la vez según los criterios de salida establecidos por el usuario y devolverlos con el formato correcto. Si el usuario ha seleccionado como columna de salida "ubicación", definirá un objeto **Ip2Location** que a su vez usará el objeto **MySqlConn** para obtener dicho valor de la base de datos definida para ello.

  El método estrella será sin duda, ***SplitLinea()***, ya que es el encargado de separar y validar cada línea recibida del fichero origen. Irá recorriendo cada posición de dicha línea obteniendo las columnas necesarias para el fichero de salida.

  Merece la pena detenerse un segundo en el siguiente código de dicho método donde de manera sencilla podemos establecer las columnas que ha seleccionado el usuario para realizar una salida ad-hoc:

```
Integer aux;
    String[] campos = formato.split("\\s+");
    int j = 0;
    String columna = "";
    while (j < campos.length) {
       aux = tryParse(campos[j]);
       if (aux != null) {
          columna = salida.get(aux);
          if (columna == "") {columna = "E";}
           textosalida = textosalida + "," + columna;
       }
       j++;
    }
```

- **ValidaFichero**.

  Validará el estado así como el formato de los ficheros log de origen de pequeño tamaño que podremos adaptar al formato Weka.

  Solamente contiene un método llamado ***validate()*** que actuará como controlador del fichero cargado en memoria completamente por el usuario.

- **MySQLConn.**

  Se encargará de establecer la conexión con la base de datos MySql alojada en el servicio Amazon RDS. Será usada solamente por un objeto padre, Ip2Location donde se encuentran encapsuladas las operaciones básicas.

  Contempla todas las operaciones de un objeto conexión, como **rollback**, **commit**, etc… pero queremos destacar el método ***init()*** que es donde se recogen los parámetros definidos y necesarios para establecer la conexión:

```
private void init()throws Exception{
      FacesContext fc = FacesContext.getCurrentInstance();
      db_server   = fc.getExternalContext().getInitParameter("DB-SERVER");
      db_user     = fc.getExternalContext().getInitParameter("DB-USER");
      db_password = fc.getExternalContext().getInitParameter("DB-
PASSWORD");
      db_driver   = fc.getExternalContext().getInitParameter("JDBC-DRIVER");
      Class.forName(db_driver);
   }
```

  Para ello usará ***FacesContext*** que es una clase del propio framework de desarrollo de java que sirve al bean de puente al exterior, ya que le permite acceder no solo al **contexto JSF** sino también al **contexto HTTP**. Esto permite al bean el acceso a los demás beans de la aplicación, a las **propiedades de la aplicación** (como el caso que nos ocupa) e incluso a la petición HTTP que se está ejecutando. El propio framework establece los mecanismos de seguridad para que dichos parámetros no sean accesibles desde fuera.

- **Ip2Location**: Centraliza las operaciones sobre la base de datos MySql alojada en el servicio Amazon RDS. A partir de una dirección IP de nuestra línea de Log origen nos devolverá su ubicación.

  El método más importante que muestra la clase Ip2Location no puede ser otro más que ***Localizacion().*** Éste recibirá por parámetro la dirección Ip de origen.

- **CsvtoArff**.

  Tendrá un único medio método llamado ***GenerarArff()*** que recibirá la ruta donde el fichero intermedio ha sido generado.

  Queremos destacar que dicha clase se basa en las propias de Weka:

```
import weka.core.Instances;
import weka.core.converters.ArffSaver;
import weka.core.converters.CSVLoader;
```

Como podemos ver, usaremos el espacio de nombres ***CsVLoader*** para cargar por completo el fichero intermedio generado y convertirlo en formato Arff de manera sencilla (fichero final). Debemos contemplar otro diseño si el nº de valores de atributos como por ejemplo, IP, origina un cuello de botella en la ejecución del programa.

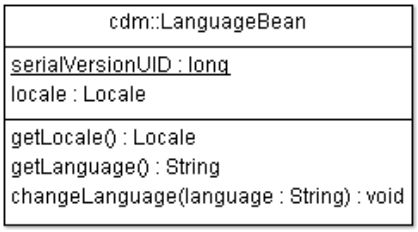


Figura 50 Clase LanguageBean

- **LanguageBean**.

  Clase independiente encargada de establecer el idioma seleccionado por el usuario en cualquier vista de la aplicación.

  Uno de sus métodos más importantes es ***changeLanguage*** ya que nos permite establecer el valor del ***caption*** de los todos los controles de la aplicación a partir de los ficheros de propiedades ***messages*** (uno por cada idioma definido en la aplicación).

```
public void changeLanguage(String language) {
      locale = new Locale(language);
      FacesContext.getCurrentInstance().getViewRoot().setLocale(new
Locale(language));
   }
```

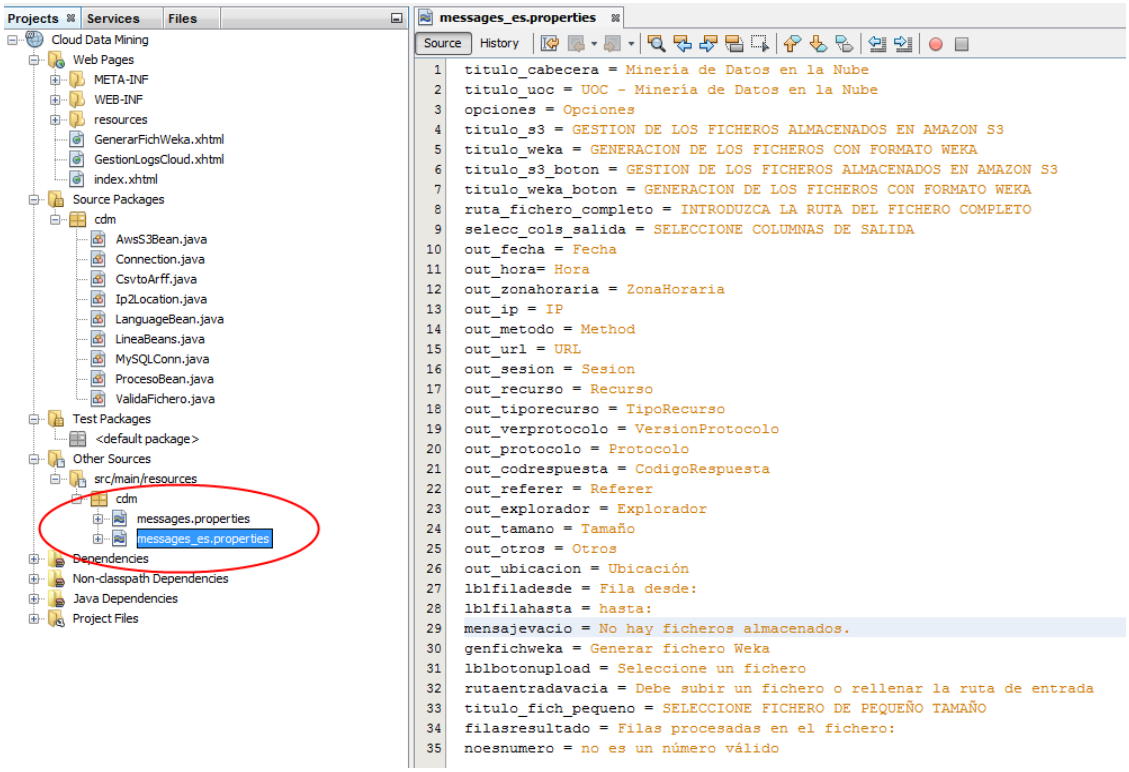


Figura 51 Parametrización de controles

3.2.2. Vistas.

Nuestra aplicación contendrá tres vistas. El punto de entrada será Index desde donde podremos navegar a las dos opciones realizadas en este trabajo: gestión de los ficheros logs de origen en un almacenamiento externo (AWS) y procesar un fichero log de origen en un fichero intermedio o final para su evaluación por la herramienta Weka.

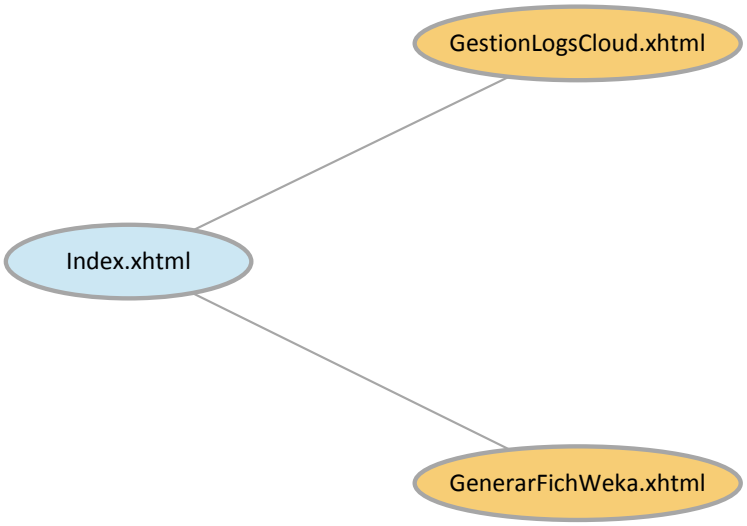


Figura 52 Vistas de la aplicación.

- **Index.xhtml.**

  A partir de ella podremos acceder a las dos vistas siguientes donde se realizan los dos grandes grupos de funcionalidades de la aplicación. Tanto en esta primera vista como en las anteriores, tendremos disponible la operación de cambio de idioma.

- **GestionLogsCloud.xhtml.**

  Pondrá de manifiesto de cara al usuario, todas las operaciones relacionadas con el servicio S3 de Amazon: listado de objetos, subida, borrado y descarga.

- **GenerarFichWeka.xhtml.**

  Contempla el interfaz de usuario para el proceso de carga de logs, selección de columnas de salida así como la generación de los ficheros intermedios y final.

3.3. Diseño tecnológico.

En este punto trataremos de acometer el traslado del documento de análisis (máquinas necesarias, framework,…) al entorno de desarrollo.

Uno de los hitos a acometer es la realización de nuestro **entorno de trabajo sin una necesidad imperante de tener disponibles servidores físicos** ya que su configuración puede acarrear una ingente cantidad de tiempo.

En los siguientes apartados veremos el diseño seguido para acometer el desarrollo para nuestra aplicación en la nube y más concretamente bajo los servicios disponibles de Amazon Web Services (AWS) donde veremos la capacidad de diseñar fácilmente plataformas escalables y con alta disponibilidad usando los mismos, y valorar nuestra plataforma con el fin de mejorar el rendimiento de las mismas.

Cuando planeamos nuestra arquitectura para Amazon Web Services (AWS), escoger dónde y cómo estableceremos nuestra arquitectura es el principal reto al que nos enfrentamos en este TFG.

A modo de resumen, podemos establecer el siguiente esquema tecnológico para nuestra aplicación:

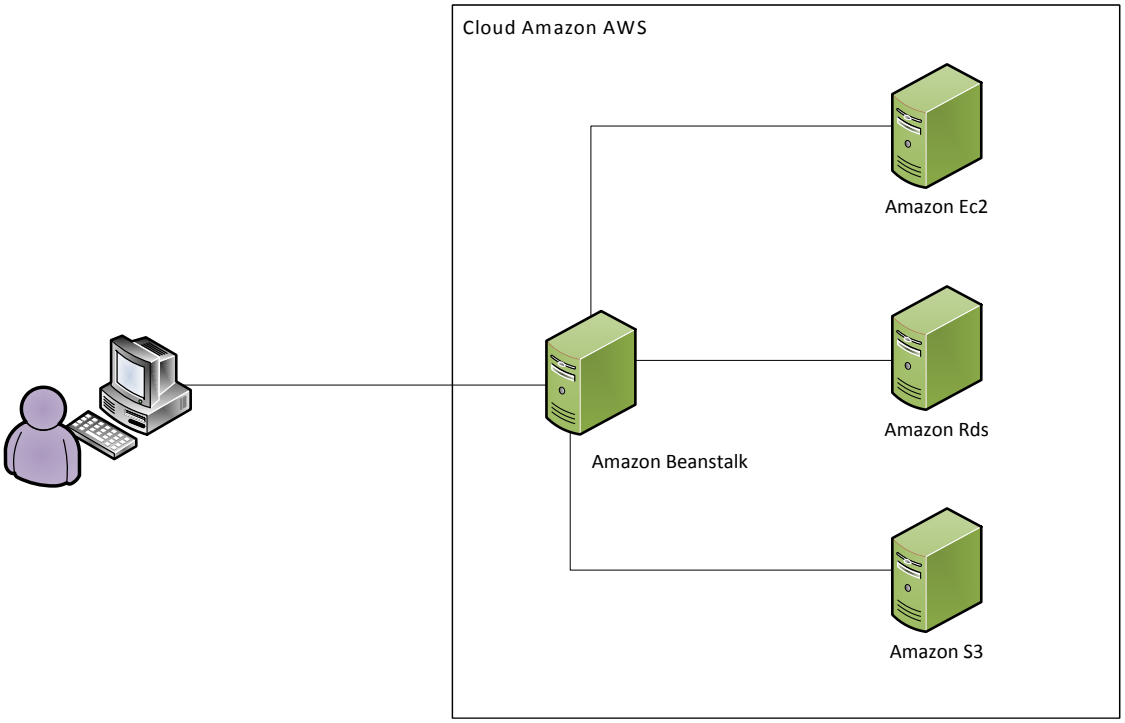


Figura 53 Esquema tecnológico aplicación.

3.4. Preparación entorno de trabajo.

3.4.1. IDE para el Desarrollo y Administración.

Para poder desarrollar la lógica de nuestra aplicación de manera lo más sencilla posible, contaremos con un IDE (Entorno de Desarrollo Integrado) gratuito y orientado a Java, como es **NetBeans**, ya que es una herramienta consolidada que nos ayuda a desarrollar de una manera amigable nuestras aplicaciones, brindándonos opciones de manera sencilla para probar y depurar de nuestro código.

Pero además, y muy importante, Netbeans cuenta además con una alta integración con los distintos servicios que necesitaremos en el desarrollo de nuestro software (Cloud incluidos) como podemos ver en la siguiente captura lo que a priori nos facilitará diversas tareas de administración:

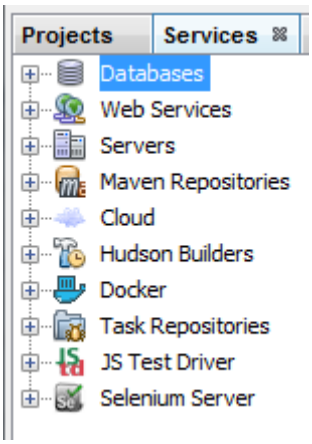


Figura 5 - Servicios asociados a NetBeans

Es muy importante escoger **herramientas consolidadas** como la mencionada ya que muchos IDE apena tienen recorrido en el tiempo o incurren en errores difíciles de solucionar una vez arrancado el desarrollo del proyecto.

3.4.2. Amazon AWS.

El siguiente paso nos lleva a Amazon Web Services (AWS) para dar respuesta a dónde y cómo se van a gestionar los servicios implicados en nuestra aplicación.

Crear el entorno que nos hace falta en AWS es en principio una tarea sencilla. Basta con registrarse en la plataforma para tener acceso a todos sus servicios. Conviene poner de manifiesto que AWS dispone de una capa gratuita y limitada que es la que en principio usaremos para nuestro desarrollo. Dicha capa incluye servicios de manera gratuita durante 12 meses a partir de la fecha de inscripción en AWS, así como ofertas de servicios adicionales que no vencen al final del periodo de 12 meses de dicha capa gratuita.

A partir del registro en dicha plataforma, podremos comenzar a usar los servicios en la nube de AWS escogiendo uno de los productos de entre los 38 disponibles. Nos centraremos en los que hemos seleccionado para nuestra aplicación:

1. **Amazon EC2:** Servicio que proporciona capacidad de cómputo con tamaño modificable en la nube. Está diseñado para facilitar a los desarrolladores la informática en la nube escalable basado en web, es decir, una vez registrados, proporciona un control completo sobre los **recursos informáticos** (**servidores y software asociados los mismos**) que necesitamos.

   Cabe destacar que EC2 como el resto de servicios ofrecidos por Amazon cuenta con varios **sistemas para garantizar la seguridad** de nuestra aplicación:

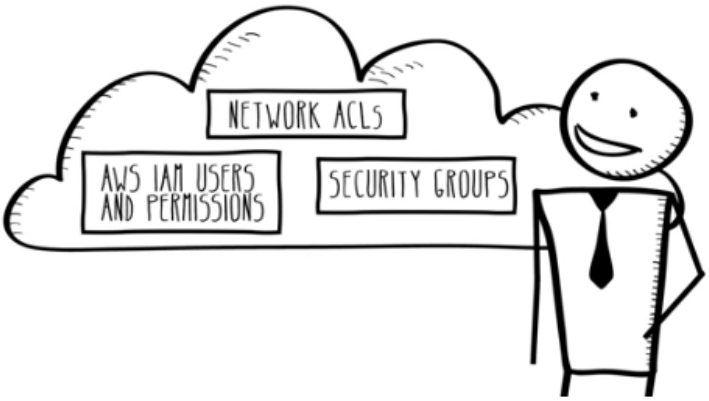


Figura 6 - Políticas de seguridad EC2

2. **AWS Elastic Beanstalk:** Servicio para **implementar y escalar** servicios y aplicaciones web desarrollados con **Java**, .NET, PHP, Node.js, Python, Ruby, Go y Docker en servidores familiares como Apache, Nginx, Passenger e IIS.

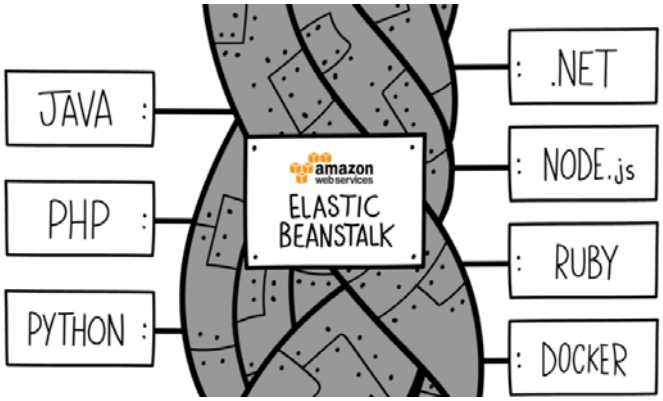


Figura 7.- Manejados Beanstalk.

Desde Netbeans (o manualmente en el propio servicio) solamente tenemos que cargar el código y Elastic Beanstalk **administrará de manera automática la implementación**, desde el aprovisionamiento de la capacidad, el equilibrio de carga y el escalado automático hasta la monitorización del estado de la aplicación. Por otro lado, tendremos el control absoluto de los recursos de AWS que alimentan su aplicación y podrá obtener acceso a los recursos subyacentes cuando quiera.

De cara al coste del proyecto, es importante remarcar que un escenario real (fuera de las limitaciones de la capa gratuita) no se recibirán cargos adicionales por Elastic Beanstalk; **solo paga por los recursos de AWS que necesitemos para almacenar y ejecutar nuestra aplicación**.

Como servidor de aplicaciones de software libre hemos escogido GlassFish desarrollado por Sun Microsystems, (Oracle), que implementa

tecnologías en la plataforma Java EE y permite ejecutar aplicaciones que siguen esta especificación como es nuestro caso.

3. **Amazon S3:** Servicio de almacenamiento de objetos con una sencilla interfaz de servicios web para almacenar y recuperar la cantidad de datos que desee desde cualquier ubicación de la web.

   Nuestra aplicación usará S3 como un repositorio masivo o "lago de datos" para tareas de análisis de nuestro proyecto, es decir, tanto para los ficheros origen sin procesar como los ya generados.

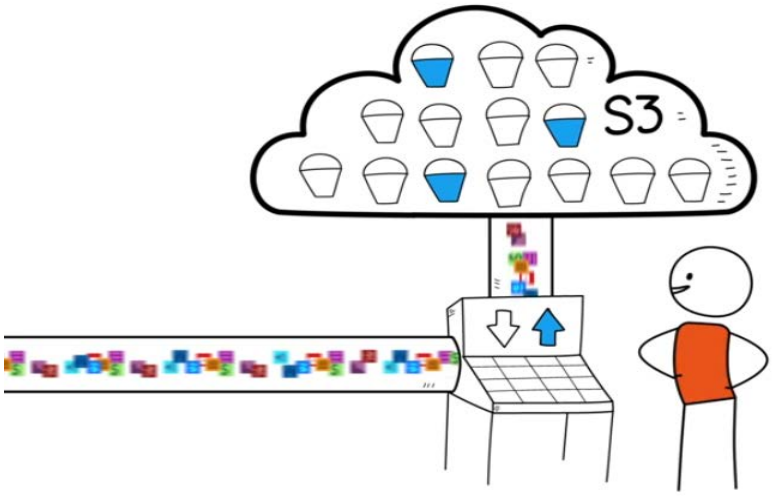


Figura 8.- Rápida escalabilidad en S3

4. **Amazon RDS:** Servicio que usaremos para configurar, utilizar y escalar la base de datos relacional en la nube. Proporciona capacidad rentable y de tamaño modificable y, al mismo tiempo, reduce tareas de administración de la base de datos, lo que nos permitirá centrarnos en nuestra aplicación. Proporciona seis motores de bases de datos para elegir, incluido Amazon Aurora, PostgreSQL, MySQL, MariaDB, Oracle y Microsoft SQL Server. En nuestro proyecto hemos escogido **MySql por la premisa del código libre**.

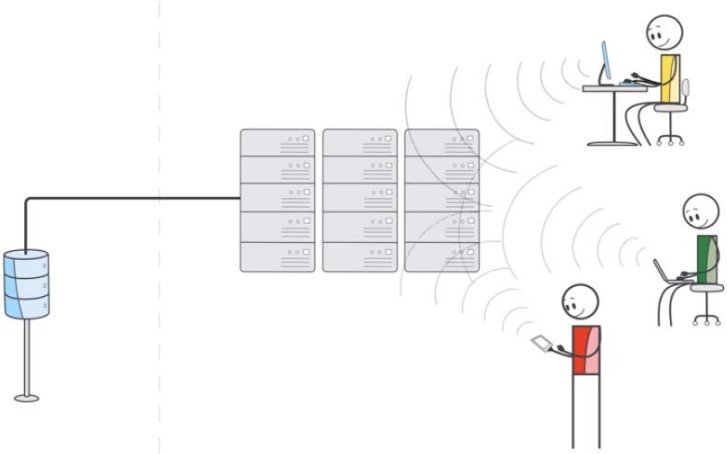


Figura 9.- Sencillez de gestión en RDS

Es importante recalcar que dicha BBDD recoge las características de la edición Community donde podemos llegar a almacenar hasta 6 TB , con instancias de hasta 32 virtual CPU y 244 GB de memoria, respaldo automático y recuperación en un punto determinado (Snapshot) y réplica de la misma.

No hemos necesitado una consola para realizar las labores DDL y DML sobre nuestra base de datos. Desde Netbeans podemos realizarlas sin necesidad de instalar herramientas adicionales como Mysql WorkBench, por ejemplo.

3.5. Implementación del código.

Una vez tenemos el entorno preparado, procedemos a realizar los objetos que realizan las funcionalidades descritas en el alcance funcional de nuestro proyecto.

En esa misma línea, nos encontramos que podemos reducir dicho alcance por dos cambios significativos que aportarán más valor a nuestro proyecto:

1. **Los ficheros no se cargan en memoria.**

   Uno de los grandes problemas de trabajar con grandes ficheros es el relacionado con la cantidad de memoria necesaria para la gestión del mismo ya que la mayoría de lenguajes de programación usan objetos que deben **cargar dichos ficheros en memoria completamente para recorrerlos**.

   Tiempo atrás, se introdujo el concepto de buffer en el sentido de “trocear” dichos ficheros a la hora de leerlos, por ejemplo. Pero dicho funcionamiento suele acarrear cuellos de botella o llenado de memoria de ejecución por dichos búferes.

   En nuestra aplicación hemos preferido usar clases de java que permiten obviar los problemas anteriores: la clase **Scanner de Java**. Nos permite analizar textos mediante patrones para descubrir tipos de datos primitivos en él. Se puede configurar un delimitador para saber hasta dónde tiene que leer el texto (una

línea a la vez sin tener en cuenta el tamaño del archivo total) guardando cada línea en un objeto de tipo String. Es decir, mediante la invocación de distintos métodos permiten leer datos de cualquier tipo (String, enteros, reales, etc.) con diversas posibilidades de separadores.

2. **Selección por el usuario de cualquier combinación de salida del fichero que será cargado en la aplicación Weka.**

   Creemos que tiene una buena relación "coste de desarrollo – uso" el que el usuario pueda escoger las columnas de salida entre todas las disponibles ya que si tenemos una selección ya marcada de antemano, nos obligará a realizar cambios en nuestra aplicación para acometerlos.

Con los dos cambios anteriores, nos adentramos de lleno en la realización del código de nuestra aplicación dentro del IDE NetBeans.

3.6. Ejecución plan de pruebas

3.6.1. Definición del plan de pruebas.

Nuestro plan de pruebas tendrá varias líneas de ejecución diferenciadas. Por un lado, haremos pruebas funcionales de la aplicación desarrollada con datos reales y por otro, realizaremos un estudio piloto con la herramienta seleccionada para la evaluación de los procesos definitivos de minería de datos (Weka).

3.6.2. Pruebas funcionales con datos reales.

No cabe duda que la prueba más importante será el procesamiento de los ficheros log del campus. Para realizar las siguientes prueba usaremos alternativamente varios ficheros facilitados por la UOC al respecto. El objetivo es llegar a probar hasta ficheros **12,5 GB** de tamaño alcanzando las **26.760.534 filas** en algunos de ellos.

En principio realizaremos las pruebas seleccionando **todas las columnas salvo ubicación** y posteriormente incluyendo ésta. De esta manera podremos analizar el coste de la llamada a la base de datos en el peor escenario (pruebas de integración).

P001 Carga de fichero de pequeño tamaño sin ubicación	
Tipo	Carga de fichero de pequeño tamaño
Precondición	Fichero de log de 100 líneas y 100KB. Seleccionadas todas las columnas propias del fichero (**salvo Ubicación**).
Resultado esperado	Generación de fichero intermedio, final y de errores.
Resultado obtenido	**Generación correcta en 3 segundos.**

El resultado es más que satisfactorio. Veamos a continuación, incluyendo la columna calculada:

P002 Carga de fichero de pequeño tamaño con ubicación	
Tipo	Carga de fichero de pequeño tamaño
Precondición	Fichero de log de 100 líneas y 100KB. Seleccionadas todas las columnas propias del fichero (**Ubicación incluida**).
Resultado esperado	Generación de fichero intermedio, final y de errores.
Resultado obtenido	**Generación correcta en 15 segundos.**

Como podemos ver el tiempo de cálculo se ha elevado considerablemente. Desde la consola de administración de Amazon RDS, vemos la evolución de los recursos asignados durante el proceso.

Figura 54 Consola AWS RDS

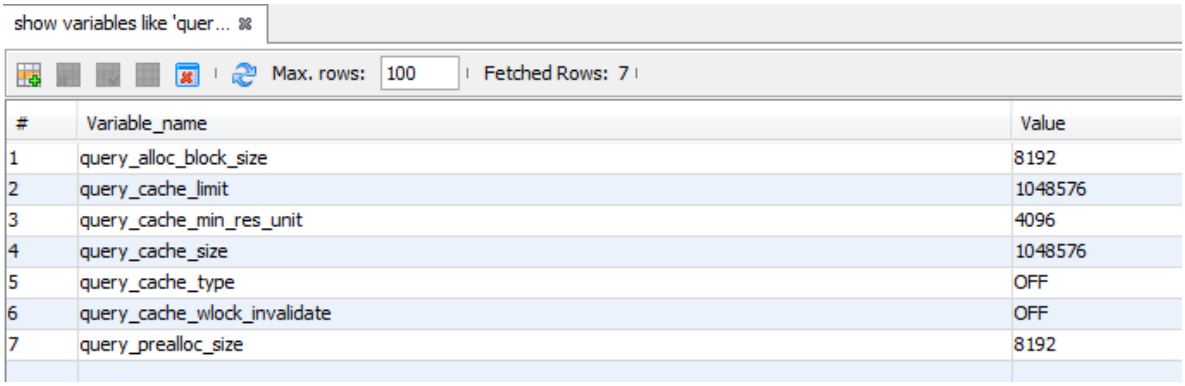
No apreciamos cargas elevadas en el servidor de BBDD (apenas tienes operaciones de lectura ni consumo). Se revisa el código fuente de la consulta para:

- Evitar que se establezca nueva conexión a la base de datos con cada línea sino que se mantenga en modo conectado.

- Usar los propios métodos de establecimiento de parámetros (IP) del objeto StatePreparement.

Comprobamos que el tiempo de respuesta tras verificar el apartado anterior, no cambia. Además, el servidor de aplicaciones tampoco registra grandes cargas de trabajo.

Por último, hemos descubierto que nuestra base de datos no tiene activada la característica **Query Caché**, siendo ésta muy utilizada para consultas muy sencillas ejecutadas un alto número de veces:



show variables like 'quer... | Max. rows: 100 | Fetched Rows: 7

#	Variable_name	Value
1	query_alloc_block_size	8192
2	query_cache_limit	1048576
3	query_cache_min_res_unit	4096
4	query_cache_size	1048576
5	query_cache_type	OFF
6	query_cache_wlock_invalidate	OFF
7	query_prealloc_size	8192

Figura 55 Variables Query Cache en MySql.

Para poder activarla, debemos ejecutar el siguiente script:

```
GRANT SUPER ON *.* TO adminuoc@'localhost' IDENTIFIED BY 'adminuoc';
SET GLOBAL query_cache_limit = 1048576;
SET GLOBAL query_cache_type = 1;
```

Por fallos en el ordenador de desarrollo, no hemos podido actualizar dicha característica, por lo que trasladamos dicho cambio al apartado de **propuestas de mejora del sistema** y continuamos con el plan de pruebas (diferenciando entre la selección de dicha columna como hemos venido haciendo hasta ahora).

P003 Carga de fichero de pequeño tamaño sin ubicación	
Tipo	Carga de fichero de pequeño tamaño
Precondición	Fichero de log de 5000 líneas y 2,5 MB. Seleccionadas todas las columnas propias del fichero (**salvo Ubicación**).
Resultado esperado	Generación de fichero intermedio, final y de errores.
Resultado obtenido	**Generación correcta en 3,5 segundos.**

El resultado de la carga para 5000 filas es más que satisfactorio.

P004 Carga de fichero de pequeño tamaño con ubicación	
Tipo	Carga de fichero de pequeño tamaño
Precondición	Fichero de log de 5000 líneas y 2,5 MB. Seleccionadas todas las columnas propias del fichero (**Ubicación incluida**).
Resultado esperado	Generación de fichero intermedio, final y de errores.
Resultado obtenido	**Generación correcta en 103 segundos.**

Volvemos a comprobar que la llamada la bbdd penaliza de nuevo nuestro aplicativo.

P005 Carga de fichero corrupto por rotura de fichero log	
Tipo	Carga de fichero de pequeño tamaño corrupto
Precondición	Fichero de log de 5000 líneas y 1,5 MB. Seleccionadas todas las columnas propias del fichero (**salvo Ubicación**). Hacia la línea 2000, eliminamos parte de la línea y el resto del fichero.
Resultado esperado	Generación de fichero intermedio, final y de errores.
Resultado obtenido	**Generación correcta en 3,3 segundos de tres ficheros (fichero de errores correcto con la fila no válida).**

Es habitual que los procesos del servidor apache sufran caídas que generen ficheros corruptos. En este caso, la gestión del error ha sido satisfactoria.

P006 Carga de fichero corrupto por caracteres especiales	
Tipo	Carga de fichero de pequeño tamaño corrompido
Precondición	Fichero de log de 5000 líneas y 1,5 MB. Seleccionadas todas las columnas propias del fichero (**salvo Ubicación**). Hacia la línea 2000, introducimos comillas y espacios en varias partes de la línea.
Resultado esperado	Generación de fichero intermedio, final y de errores.
Resultado obtenido	**Generación correcta en 3,5 segundos de tres ficheros (fichero de errores correcto con las filas no válidas).**

La carga de un fichero de pequeño tamaño nos puede servir de ayuda para valorar pequeños datasets como origen de patrones en los que se basará Weka para su aplicación.

El siguiente paso es evaluar ficheros de varios millones de filas. Para ello, estableceremos unos valores de filas inicial y final acordes con el número de filas procesadas. La prueba será ejecutada con el fichero campusF5.-192.168.18.2.log-20120315 que contiene 26.760.534 filas. Si el resultado es satisfactorio, no será necesario acotar a la baja el nº de filas o incrementar la capacidad del hardware de nuestro servidor de aplicaciones para su procesamiento.

P006 Carga de fichero desde ruta sin ubicación seleccionada.	
Tipo	Carga de fichero a partir de su ruta en sistema de archivos del servidor.
Precondición	Fichero de log de 26.760.534 líneas y 1,5 MB. Seleccionadas todas las columnas propias del fichero (**salvo Ubicación**).
Resultado esperado	Generación de fichero intermedio, final y de errores.
Resultado obtenido	**Generación incorrecta del fichero ARFF. Generación correcta del fichero intermedio y de errores.**

Hemos detectado el principal problema de la carga de ficheros grandes con las propias librerías de Weka. El proceso de crear los pares [atributo – valor] de la clase **CSVLoader** (sobre todo de la columna IP) del espacio de nombres weka.core.converters carga totalmente el fichero en memoria y produce un error "**out of memory**" debido a la

enorme cantidad de instancias que crea dicha clase en espacio heap.

El **heap** es el espacio de memoria en tiempo de ejecución que se usa para almacenar las instancias de clases, objetos y arrays. Se crea en el inicio de la máquina virtual (JVM) y es gestionado por el **Garbage Collector**.

Tenemos **tres opciones** para abordar dicho problema:

- Disminuir el nº de filas a evaluar.

- Dentro de nuestra configuración cloud (Amazon Beanstalk) y nuestro servidor de aplicaciones embebido Glassfish, Podemos mdificar el tamaño del heap mediante comandos de la JVM: mínimo (-Xms512m) y máximo (-Xmx1024m).

  El problema de esta opción radica en que el gasto por consumo de recursos de la capa cloud se verá aumentado. No creemos que sea viable su aumento ya que podemos sacar patrones de navegación con un porcentaje bastante acertado reduciendo el nº de filas a evaluar.

- Para evitar los gastos de la segunda opción y al disponer del fichero intermedio, podemos generar el fichero final ARFF desde un pc normal con Weka instalado donde podremos aumentar la memoria heap de la máquina virtual de java sin costes adicionales, generar dicho fichero y proceder a su evaluación con el algoritmo seleccionado.

3.6.3. Cambio idioma aplicación.

P007 Cambio de idioma de la aplicación.	
Tipo	Lingüistico
Precondición	Nos encontramos en cualquier página de la aplicación.
Resultado esperado	Cambia de idioma todos los controles de la aplicación.
Resultado obtenido	**Cambio correcto.**

El cambio se produce sin problemas gracias a la parametrización que realiza JSF al respecto.

3.6.4. Gestión de ficheros almacenados en Amazon S3.

En la siguiente pantalla, al cargarse, podremos encontrar la pantalla de gestión del repositorio al completo:

Figura 56 Gestión de ficheros Amazon S3

Probamos las opciones del menú con las siguientes pruebas:

P008 Subir un fichero de gran tamaño al repositorio	
Tipo	Repositorio
Precondición	Fichero seleccionado en el sistema de archivos local.
Resultado esperado	Fichero subido y grid actualizado.
Resultado obtenido	**Se produce excepción con el cambio de estado del cliente AS3 pero el fichero aparece en el repositorio.**

P009 Descargar fichero de gran tamaño	
Tipo	Repositorio
Precondición	Fichero seleccionado en el sistema externo.
Resultado esperado	Fichero subido y grid actualizado.
Resultado obtenido	**Fichero descargado correctamente (petición http completada correctamente).**

P010 Preparar bandeja de salida de fichero a actualizar	
Tipo	Repositorio
Precondición	Escogemos varios ficheros en el sistema de archivos local para subir .
Resultado esperado	Ficheros subidos secuencialmente, grid de destino actualizado y bandeja de salida vacía.
Resultado obtenido	Ficheros subidos secuencialmente, grid de destino actualizado y bandeja de salida vacía.

Para poder validar el funcionamiento de la aplicación podemos valernos además de la propia consola de administración de AS3 ofreciendo las mismas funcionalidades.

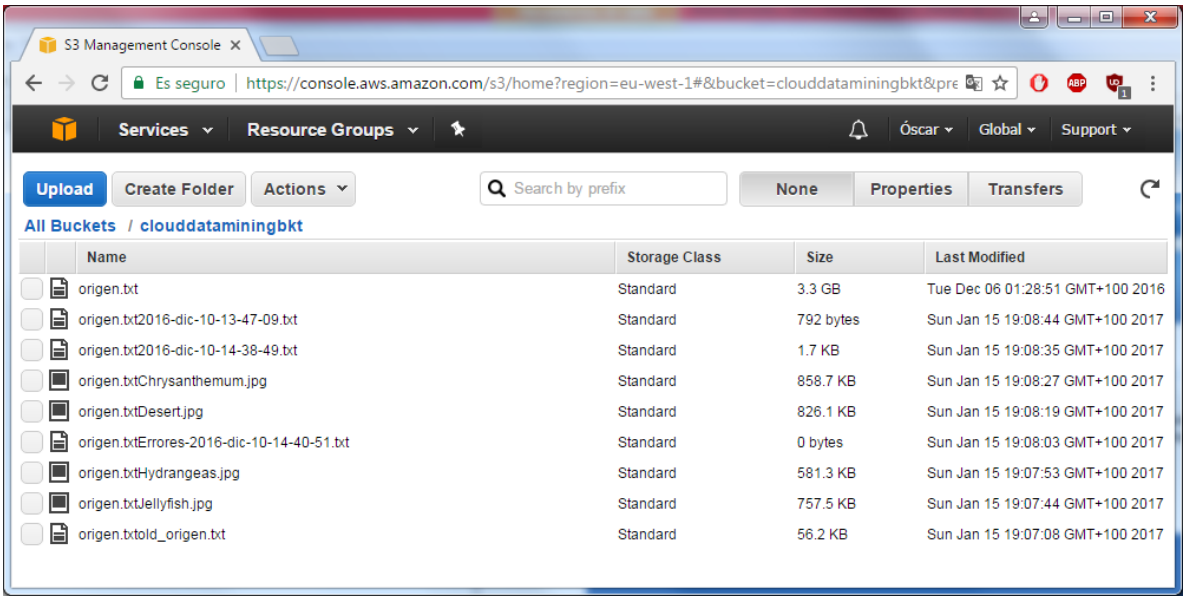


Figura 57 Consola administración Amazon AS3.

El objeto de realizar este desarrollo tiene dos líneas:

- Preparar un entorno para que la generación de los ficheros finales Arff se hagan como origen de datos el repositorio externo sin necesidad de pasar por local.

- Configuración de un cliente que permite vincularse a otros repositorios, como por ejemplo, Sharepoint, por ejemplo, si la universidad valora su uso.

3.6.5. Estudio piloto Weka.

Seleccionamos 5000 filas de un fichero real y generamos el algoritmo sobre el fichero de salida ARF.

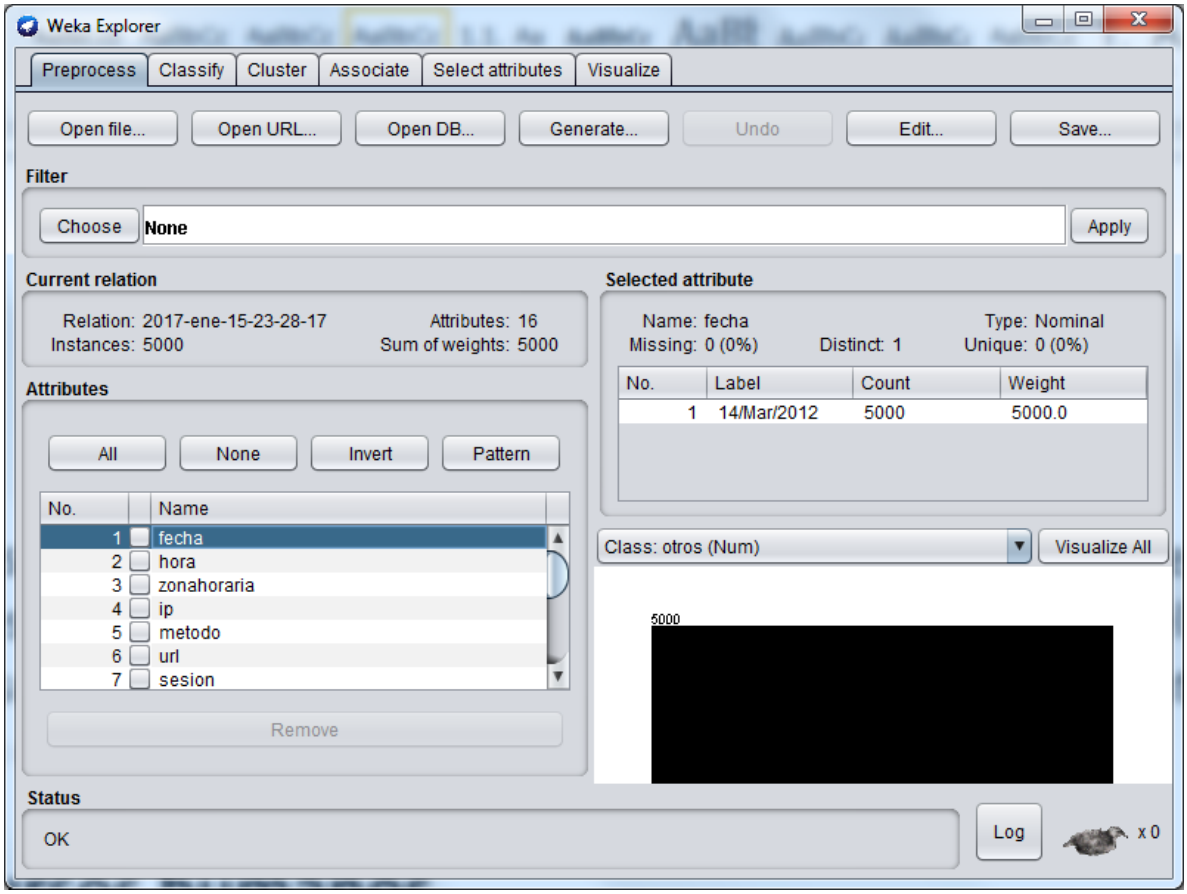


Figura 58 Prueba piloto cargada

Vemos cómo queda la salida.

```
=== Run information ===

Scheme:       weka.clusterers.SimpleKMeans -init 0 -max-candidates 100 -periodic-pruning 10000 -min-density 2.0 -t1 -1.25 -t2 -1.0 -N 2 -A "weka.core.EuclideanDistance -R first-last" -I 500 -num-slots 1 -S 10
Relation:     2017-ene-15-23-28-17
Instances:    5000
Attributes:   16
              fecha
              hora
              zonahoraria
              ip
              metodo
              url
```

```
        sesion
        recurso
        tiporecurso
        verprotocolo
        protocolo
        codrespuesta
        referer
        explorador
        tamano
        otros
Test mode:    evaluate on training data


=== Clustering model (full training set) ===


kMeans
======

Number of iterations: 4
Within cluster sum of squared errors: 22685.579407445894

Initial starting points (random):

Cluster 0:
14/Mar/2012,0:06:04,10,188.78.177.156,GET,rbiniciuser_modullibrary561216jsfeature
slibrarysetprefsdynamicheightse1116911f6693a5c3503dc33eab9ed62d7c59206e549f4
58ae8441d7c38e43288fb3fb06ab74e2c292cb36fb2965a8bb3935c02713e9c4643e1a8
48f5672e0fc,E,E,rbiniciuser_modullibrary561216jsfeatureslibrarysetprefsdynamicheig
htse1116911f6693a5c3503dc33eab9ed62d7c59206e549f458ae8441d7c38e43288fb3f
b06ab74e2c292cb36fb2965a8bb3935c02713e9c4643e1a848f5672e0fc,E,11,HTTP,200,
'Mozilla 5 0  Ubuntu  X11  Linux x86_64  rv 9 0 1  Gecko 20100101 Firefox 9 0
1',2299,16
Cluster 1:
14/Mar/2012,0:06:24,10,87.221.250.67,GET,UOCjsequationEditorASCIIMathMLjs,E,E,U
OCjsequationEditorASCIIMathMLjs,E,11,HTTP,200,'Mozilla 5 0  compatible  MSIE 9 0
Windows NT 6 1  WOW64  Trident 5 0 ',42786,1

Missing values globally replaced with mean/mode

Final cluster centroids:

Cluster#
Attribute
Full  Data 0
1
```

(5000.0) (924.0)
(4076.0)
==========
fecha
14/Mar/2012 14/Mar/2012
14/Mar/2012
hora
0:05:57 0:05:59
0:05:57
zonahoraria
10 10
10
ip
192.168.1.150 88.12.92.185
192.168.1.150
metodo
GET GET
GET
url
webappswebUtilsservletInvitationChatServlet
rbinicitipsrandomformattext
webappswebUtilsservletInvitationChatServlet
sesion
E E
E
recurso
E E
E
tiporecurso
webappswebUtilsservletInvitationChatServlet
rbinicitipsrandomformattext
webappswebUtilsservletInvitationChatServlet
verprotocolo
E E
E
protocolo
10.9876 10.9448
10.9973
codrespuesta
HTTP HTTP
HTTP
referer
213.4658 270.8669
200.4534
explorador

Mozilla 5 0 compatible MSIE 9 0 Windows NT 6 1 WOW64 Trident 5 0 Mozilla 5 0 Windows NT 6 1 WOW64 rv 10 0 2 Gecko 20100101 Firefox 10 0 2 Mozilla 5 0 compatible MSIE 9 0 Windows NT 6 1 WOW64 Trident 5 0
tamano
5868.0386 3308.4481
6448.2794
otros
83.9776 114.1916
77.1283

Time taken to build model (full training data) : 0.13 seconds

=== Model and evaluation on training set ===

Clustered Instances

0 924 (18%)
1 4076 (82%)

El resultado es bastante bueno y rápido en su cálculo.
Si elevamos la cifra de filas a su totalidad volvemos a tener el error de heap de JVM. Aumentando la memoria llegamos a cifras más altas de filas valoradas.

4. Valoración económica.

Analizando los costes del proyecto, se separa el estudio en tres categorías, la referente a los costes de salarios (personal), la infraestructura (HW) y el coste de Amazon AWS. No se suman costes de licencias del software, pues todo el utilizado ha sido software libre sin coste alguno.

4.1. Costes recursos humanos.

En cuanto a la determinación de los salarios, se hace una valoración idéntica entre el trabajo que desarrolla un administrador de sistemas y un analista, igualmente importantes. Aparece lógicamente la figura del programador para llevar a cabo las indicaciones necesarias de los dos roles anteriores. Las horas computadas en el trabajo de cada uno de estos perfiles quedan reflejadas en la siguiente tabla. Así pues, para calcular los gastos totales dedicados a los roles del proyecto, los salarios quedan en 35000€ brutos/año para los dos primeros perfiles y 25000€ brutos/año para el rol de programador. Se estima que se trabajan 1800 horas/año.

Como se puede ver, el coste total de salarios es de 3291,65€, incluyendo las horas de reuniones con el jefe de proyecto para llevar a cabo las revisiones del mismo.

A partir de los importes brutos de cada rol así como el número de horas dedicadas al proyecto, estimamos los siguientes costes totales:

Rol	Horas	Coste
Jefe de Proyecto		
Analista	70	1361,11€
Administrador de sistemas	60	1166,66€
Diseñador de Proyecto		
Programador	55	763,88€
Responsable de pruebas		
Coste Total		**3291,65€**

Figura 59. Coste Recursos Humanos.

4.2. Coste Hardware

Dentro de la categoría de recurso físicos del proyecto se cuenta con un ordenador personal cuyo coste es 500 €.

4.3. Coste Amazon AWS

Estos costes de Amazon AWS engloban los costes tanto en Beanstalk para el funcionamiento de las instancias y despliegues, S3 para el almacenamiento de los ficheros, RDS para albergar la BBDD y el coste de Transferencias de datos.

Amazon AWS tiene una oferta de bienvenida llamada "Capa Gratuita de AWS", diseñada para obtener experiencia práctica sobre los servicios en la nube de AWS sin coste alguno durante 12 meses.

En la realización del proyecto los costes de Amazon AWS han sido mínimos al hacer uso de la oferta "Capa Gratuita de AWS". Aun así, hemos rebasado levemente dicha gratuitad. En la siguiente tabla se detallan los costes de Amazon AWS.

	Noviembre 2016	Diciembre 2016	Enero 2017 (estimado)	Total
Amazon AWS	**12,69€**	**17,89€**	**6,36€**	**36,94€**

4.4. Coste Total.

El coste aunando recursos humanos para su realización, hardware necesario y coste de Amazon AWS alcanza el valor de **3828,59 €.**

5. Conclusiones.

Escogimos el área de **Herramientas para el trabajo colaborativo** ya que queríamos profundizar en el desarrollo de aplicaciones web aunando dos entornos de desarrollo completamente nuevos, esto es, computación en la nube y java. En esa misma línea queríamos diferenciar el confundido término "datawarehouse" de la "minería de datos", es decir, no partimos de datos corporativos totalmente normalizados y conocidos. La minería de datos se enfoca en descubrir patrones nuevos y además sobre datos desnormalizados y no enfocados para su procesamiento (no indexados).

El objetivo de realizar una aplicación con alto grado de usabilidad y simplicidad se ha conseguido mediante el uso de tecnologías que forman parte de estado del arte en el desarrollo de aplicaciones web.

Desde el punto de vista de conocimientos adquiridos, se ha investigado sobre la Computación en la Nube, la tecnología que está detrás de grandes proyectos tecnológicos de hoy en día. Además se ha investigado el estado del arte en aplicaciones web, con el objetivo de realizar una aplicación web sencilla y amigable. Asimismo se han reforzado los conocimientos de tecnologías y el uso del marco de trabajo MVC.

Y por último declarar, que como ingenieros nuestra función principal es la de realizar diseños o desarrollar soluciones tecnológicas a necesidades económicas, industriales o sociales, y son estas últimas las reflejadas en el proyecto "Mierías de Datos en la Nube".

5.1. Líneas de trabajo futuro no abordables en este proyecto.

5.1.1. Weka distribuido.

Una de las características más interesantes de Weka es que permite distribuir la ejecución de un proceso entre varios ordenadores mediante Java RMI. De esta manera podríamos abordar el estudio de grandes ficheros y/u orígenes. El modelo usado es el cliente-servidor.

Como primer paso deberíamos configurar un servidor para después explicar la configuración de los clientes.

Los elementos a configurar en el servidor son los mismos que cualquier otro experimento pero además de esto debemos elegir el tipo de distribución, por **fichero de datos** (*by data set*), o **por iteración** (*by run*). En ese sentido, deberíamos tener en cuenta que el número de archivos en el que realizar los experimentos es pequeño la distribución se haga por iteraciones, ya que distribuirá cada una de las tareas fundamentales de un experimento; sin embargo, si usamos un conjunto de archivos

grande es recomendable el uso de una distribución por archivos. Este último es nuestro caso por tamaño de fichero origen y algoritmo usado.

Además nos permitiría probar como origen de datos una conexión jdbc, ya que dentro de un proceso distribuido lo normal y más cómodo es que los datos estén en una base de datos común a todos los ordenadores que participen en la ejecución del experimento.

5.1.2. Origen de datos Amazon S3.

Considerar un repositorio externo como origen de datos para la aplicación sin que hubiera que cargar nosotros los archivos directamente. Simplemente seleccionarlos del grid del almacenamiento.

5.1.3. Destino o propuesta descarga de fichero final.

Va en la misma línea que el apartado anterior. El resultado del proceso de generación del fichero Weka podría ser albergado en el repositorio destino. Así, podríamos acceder a ellos desde cualquier cliente para usarlos como dataset origen de nuestra aplicación Weka.

5.1.4. Evitar el uso de la clase CsVLoader.

Cualquier desarrollo que evite cargar en memoria todo el fichero intermedio para generar el fichero Arff, debe ser nuestra máxima prioridad. Tal vez, una solución sería dividir el fichero intermedio en varios datasets en su procesamiento para poder realizar su estudio en la herramienta Weka de manera más sencilla. Es un tema a valorar seriamente.

5.1.5. Ubicación.

Una gran mejora sería no usar un acceso de una base de datos remota para calcular la ubicación de la dirección Ip. Debería estudiarse otro tipo de estructura e integración.

6. Glosario

AWS	Amazon Web Services
S3	Simple Storage Service
Buffer	Memoria de almacenamiento temporal de información que permite transferir los datos entre objetos java.
Kmeans	Método de agregación que, como tal, propone a partir de un conjunto de datos, la obtención de una enumeración de grupos de objetos con características similares
Arff	Attribute-Relation File Format. Formato de fichero tipo atributo-relación.
XUL	XML-based User-interface Language, lenguaje basado en XML para la interfaz de usuario
Bean	Modelo de componentes creado por Sun Microsystems para la construcción de aplicaciones en Java.
FacesContext	Clase del propio framework de desarrollo de java para acceder a otros ámbitos de la sesión activa.
Heap	Espacio de memoria en tiempo de ejecución que se usa para almacenar las instancias de clases, objetos y arrays.
JVM	Java Virtual Machine. Proceso ejecutable en una plataforma específica, capaz de interpretar y ejecutar instrucciones expresadas en un código binario especial, el cual es generado por el compilador del lenguaje Java.

7. Bibliografía

1. https://httpd.apache.org/docs/2.4/logs.html (4/12/2016)
2. http://www.oracle.com/technetwork/es/articles/bi/nuevas-caracteristicas-de-obiee-11g-2095449-esa.html (4/12/2016)
3. http://docs.oracle.com/cd/E25054_01/bi.1111/e10541/logging.htm (10/11/2016)
4. http://docs.oracle.com/javase/7/docs/api/java/util/Scanner.html (20/11/2016)
5. http://docs.oracle.com/javase/7/docs/api/java/util/regex/Pattern.html (20/11/2016)
6. http://materials.cv.uoc.edu/daisy/Materials/PID_00237676/html5/modul_1.html#w26aab5b9 (4/1/2017)
7. http://ocw.uc3m.es/ingenieria-informatica/herramientas-de-la-inteligencia-artificial/contenidos/transparencias/TutorialWeka.pdf (2/1/2016)
8. https://weka.wikispaces.com/Use+WEKA+in+your+Java+code (7/12/2016)
9. https://www.mysql.com/ (10/11/2016)
10. http://cloudacademy.com/blog/amazon-ec2-container-service-docker-aws/ (2/1/2017)
11. http://www.mkyong.com/java/find-out-your-java-heap-memory-size/ (14-01-2016)

8. Anexos

- Manual de Administrador
- Manual de Usuario.