DISEÑO Y DESARROLLO DE UNA APLICACIÓN WEB PARA LA COMPARACIÓN DE PERTURBACIONES TRANSCRIPTÓMICAS.

Beatriz Ranera Beltrán
Máster Bioinformática y Bioestadística
Área 1

Nombre Consultor/a: Guerau Fernández Isern
Nombre Profesor/a responsable de la asignatura: María Jesús Marco Galindo

02/01/2019

Licencias alternativas (elegir alguna de las siguientes y sustituir la de la página anterior)

A) Creative Commons:

Esta obra está sujeta a una licencia de Reconocimiento-NoComercial-SinObraDerivada 3.0 España de Creative Commons

Esta obra está sujeta a una licencia de Reconocimiento-NoComercial-CompartirIgual 3.0 España de Creative Commons

Esta obra está sujeta a una licencia de Reconocimiento-NoComercial 3.0 España de Creative Commons

Esta obra está sujeta a una licencia de Reconocimiento-SinObraDerivada 3.0 España de Creative Commons

Esta obra está sujeta a una licencia de Reconocimiento-CompartirIgual 3.0 España de Creative Commons

Esta obra está sujeta a una licencia de Reconocimiento 3.0 España de Creative Commons

B) GNU Free Documentation License (GNU FDL)

Copyright © AÑO TU-NOMBRE.

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.3 or any later version published by the Free

C) Copyright

FICHA DEL TRABAJO FINAL

Título del trabajo:	*Diseño y desarrollo de una aplicación web para la comparación de perturbaciones trasncriptómicas*
Nombre del autor:	*Beatriz Ranera Beltrán*
Nombre del consultor/a:	*Guerau Fernández Isern*
Nombre del PRA:	*María Jesús Marco Galindo*
Fecha de entrega (mm/aaaa):	01/2019
Titulación::	*Bioinformática y bioestadística*
Área del Trabajo Final:	*Área 1*
Idioma del trabajo:	*Castellano*
Palabras clave	*Perturbagén, ConnectivityMap, Firma.*

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

Los ensayos preclínicos *in basket* se basan en comparar los mecanismos de acción por los que distintos células tumorales desarrollan su actividad para encontrar dianas que sean comunes. El proyecto LINCS trabaja creando un catálogo que recoge los cambios de expresión de genes tras la exposición de distintos tipos de células humanas a agentes, convirtiendo estos agentes en potenciales antitumorales.

Los catálogos recogen informaciones sobre los experimentos que se han llevado a cabo y la variación de expresión de los genes, firmas. El proyecto también proporciona a desarrolladores de software elementos informáticos que permiten la consulta rápida de estos catálogos y extraer información de ellos.

Con estos antecedentes en la presente memoria se plantea el diseño y desarrollo de un prototipo de aplicación web que permita prestar servicio a investigadores en la búsqueda de antitumorales utilizando las herramientas ofrecidas por el Broad Institute, encargado del proyecto LINCS. El prototipo de esta aplicación está desarrollada en el lenguaje Java, utilizando como núcleo principal las clases CMapJ, cuyo fin es el de recorrer ficheros con extensión .gctx que recogen los datos de expresión de genes, firmas, en una matriz de datos, y los metadatos experimentales con las condiciones para obtener dichas firmas.

El resultado del trabajo ha sido un prototipo de aplicación web capaz de recorrer ficheros .gctx y comparar las firmas incluidas en ellos con firmas proporcionadas por el investigador para ayudar en la búsqueda de tratamientos a partir de otros ya existentes, mostrándole la información de esas condiciones.

Abstract (in English, 250 words or less):

In basket preclinical trials aim to compare the pathways through different types of tumoral cells have their activity to find common targets. LINCS Project Works on the creation of a catalog containing gene expression changes after human cell exposure to agents, turning them in potential anti-tumoral agents.

Within the catalogs is possible to find information about experimental conditions and gene expression changes, called signatures. The project also provides some programming tools for software developers to fast query these catalogs to get information from them.

Considering the above description, the current manuscript presents the design and develop of a web application protytpe based on the code provided by the Broad Institute, host of LINCS project, to help researchers to find antitumorals. The prototype of this app is developed in Java programming language, using as main core the CMapJ classes, which are able to read files with .gctx extension that contains the gene expression data in data matrix, the signatures, and the experimental metadata describing the experimental conditions to obtain those signatures.

The final result of this work is a web application prototype able to extract information of .gctx files in order to compare with other signatures obtained, experimentally, by the researcher. In consequence, the researcher might infere antitumoral behaviour by comparing with previous antitumoral investigations.

Índice

Lista de figuras

1. Introducción

1.1 Contexto y justificación del Trabajo

El descubrimiento de sustancias que puedan afectar al crecimiento celular de células tumorales es clave para el diseño de terapias antitumorales. En la literatura se pueden encontrar compuestos que pueden presentar actividades antitumorales en distintos rangos de concentración y probados en gran variedad de líneas celulares tumorales mostrando diferentes resultados [1-5].

Diferentes tipos de tumores, como glioblastomas y leucemias, pueden tener mecanismos de acción más similares entre ellos que entre sus mismos tipos. Por esta razón los ensayos preclínicos *in vitro* "basket" [6] pueden ser considerados una alternativa a los ensayos clínicos habituales. En estos ensayos se utilizan un número de células reducidas de tumores que pueden ser muy diferentes pero que pueden presentar aberraciones genómicas similares. Una de las maneras de detectar las diferencias producidas en la trascriptómica de una célula al enfrentarse a una sustancia es la de determinar su firma asociada a fármacos ("Drug signature").

Teniendo en cuenta esta premisa, para poder llevar a cabo ensayos "basket" es necesario que investigadores que trabajan en el campo de búsqueda de nuevos compuestos, tanto de origen biológico (como pueden ser anticuerpos o siRNA) como químicos (fármacos sintéticos o sustancias naturales) dispongan de la información de otras investigaciones llevadas a cabo para poder comparar con su propio trabajo. En este marco de necesidad surge el proyecto LINCS (Library of Integrated Network-Based Cellular Signatures) [7] para crear un catálogo que recoja los cambios de expresión de genes tras la exposición de distintos tipos de células humanas a agentes que modifican su expresión.

En concreto la filosofía que persigue el consorcio de grupos que trabaja en el proyecto LINCS es la de generar y hacer públicos los datos que muestren como las células responden a estresores genéticos y ambientales, para ayudar a entender de forma más detallada las rutas células y ayudar, de esta manera, a desarrollar terapias que perturban las rutas y redes de sus estados normales [8].

Este proyecto ha desarrollado una plataforma que a través de la web permite acceder a informaciones de ensayos, tipos celulares y perturbagenes que ya forman parte del catálogo que elaboran, y también permite la participación de las entidades asociadas al mismo, así como la incorporación de nuevos datos y el acceso a software especializado para analizar datos.

Uno de los trabajos que se encuentran dentro de LINCS es el ensayo L1000. Se trata de un experimento a gran escala de expresión génica que mide la abundancia de los mRNA de 978 genes denominados "landmark" de la especie humana, utilizando beads de 500 colores de Luminex (dos tránscritos son detectados por cada bead de color). A partir de los resultados obtenidos de estos genes representativos se aplicó un algoritmo de inferencia para inferir la expresión de otros 11350 genes en el transcriptoma [9]. También se midieron en este experimento la expresión de 80 genes, cuya expresión permaneció sin alterar en las condiciones medidas. Las condiciones experimentales que se recogen en este ensayo agrupan a 51219 perturbagenes, 76 líneas celulares diferentes, 89 dosis diferentes y 12 tiempos diferentes [10], lo que hace un total de 1.3 millones de firmas distintas.

Los genes denominados "landmark" son aquellos genes cuya expresión se ha considerado ser informativa y representativa para caracterizar el transcriptoma, y cuya expresión es la que se ha medido de forma directa en el experimento llevado a cabo por el consorcio LINCS en el ensayo L1000, de hecho, la presencia L en el nombre L1000 hace referencia a estos genes "landmark". Estos genes fueron seleccionados por capturar más del 80% de la información de los perfiles de expresión de aproximadamente 22000 genes [11]. Por tanto, se consideraron como buenos predictores para inferir la expresión del resto de genes que no se midieron directamente en el ensayo.

Actualmente el acceso a la base de datos sólo se puede realizar a través de autentificación en la plataforma clue.io con fines académicos, por lo que para la elaboración del prototipo que se describe en la presente memoria se ha procedido a registrarse en dicha plataforma para tener acceso libre a los datos contenidos en ella.

Estos resultados se encuentra almacenados en ficheros .gctx para permitir el almacenamiento de expresión, de metainformación (datos sobre el experimento como tipo de sustancia que ha provocado el cambio, líneas celulares empleadas, etc…). Para poder acceder a estas informaciones contenidas en el fichero binario es preciso hacer uso del software Connectivity Map (CMap). Se trata paquetes de software de código abierto en diversos lenguajes (Python, R, Matlab y Java) desarrollado por uno de los equipos integrantes. En concreto, para la elaboración de este prototipo se han utilizado las clases de Java desarrolladas a tal fin, que reciben el nombre de CMapJ y pueden ser descargadas del repositorio github [12].

Los ficheros .gtcx (Figura 1) tienen compilada la información referente a los ensayos anteriormente mencionados y la matriz de datos resultante de dichos ensayos. Estos datos están distribuidos en distintos directorios, encontrándose por una parte la matriz y por otra los metadatos experimentales. El objetivo que se persigue a través de un fichero compilado de esta manera, es que la búsqueda de información dentro de ellos es más rápida que si se tratase de un archivo de texto

plano. Esto se demostró en el trabajo de Enache y colaboradores [10] en el que se compara la velocidad de procesamiento de la información contenidas en un fichero .gct, de texto plano y el .gctx, compilado. Se determinó que se tardaba 4 veces más en leer las 100000 primeras muestras en el .gct.

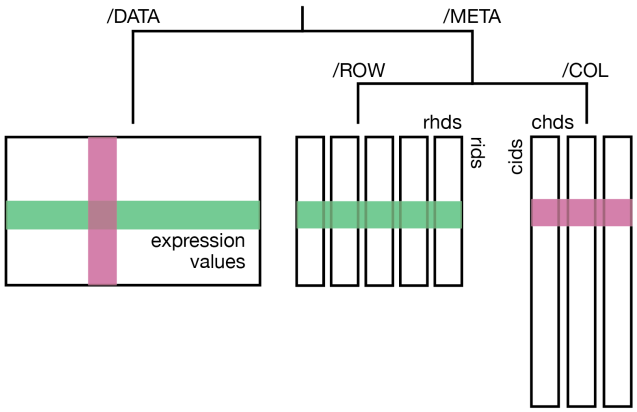


Figura 1. Estructura del fichero .gctx

El fichero .gtcx está basado en la tecnología HDF5 (desarrollada por el HDF Group [13] durante los últimos 20 años) que centra sus esfuerzos en la creación, manejo y mantenimiento de matrices de datos grandes y densas con metadatos que recogen las anotaciones para que éstas sean fáciles de almacenar y explorar. El incremento de rapidez a la hora de recorrer el fichero se basa en que el código capaz de procesar este fichero realiza una extracción parcial de los datos contenidos en el fichero sin necesidad de cargarlo entero en memoria.

Los ficheros .gtcx, como se ha nombrado anteriormente, tienen por un lado el contenido de los metadatos, que se corresponderían con las anotaciones, y por otro lado el resultado de los experimentos. Existen distintos niveles de resultados (Figura 2) que pueden almacenarse en la matriz de datos.

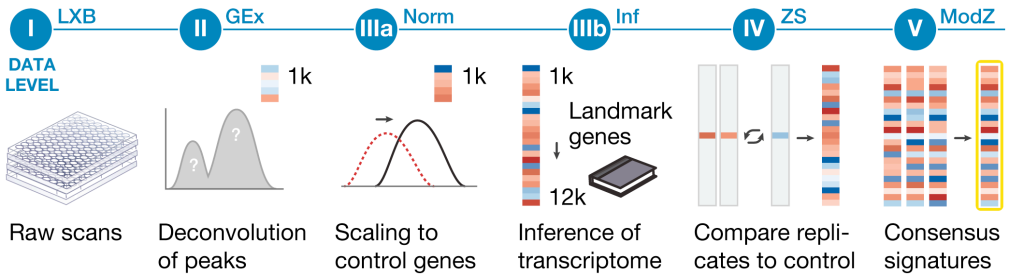


Figura 2. Esquema de los distintos niveles de datos que pueden encontrarse en los ficheros .gctx [14].

- Nivel 1: medidas datos de fluorescencia.
- Nivel 2: datos brutos de expresión génica con sondas epsilón o sondas delta.
- Nivel 3: datos normalizados o inferidos.

- Nivel 4: datos normalizados con los controles de placa de experimento.
- Nivel 5: valores de Z-score de cada gen tras calcular la media pondera de las réplicas. El valor Z-score es una forma de estandarizar datos a través de una transformación de los mismos [15]. La forma de transformar los datos responde a la siguiente fórmula:

$$X_{new} = \frac{X - media(X)}{desviacion\ standar\ (X)}$$

  Donde X es el valor de expresión obtenido para cada gen cual se le resta la media de todas las expresiones de un experimento y se divide por la desviación estándar calculada para todos esos datos. De esta manera se reescalan todos los valores de expresión en términos de desviación estándar para que estén repartidos por encima y por debajo de la media. Valores de z-score por debajo de -3 y por encima de +3 indican valores atípicos.

El prototipo diseñado y desarrollado en el presente trabajo está orientado a leer ficheros .gtcx que almacenan datos de Nivel 5. Para ello se necesitará que los datos que el usuario aporte sean también datos de expresión diferencial de genes transformados en Z-score para poder realizar la comparación con los contenidos en el fichero. El algoritmo más adecuado sería el llevado a cabo por los autores del artículo de presentación del ensayo L1000 [16]. En este artículo Subramanian y colaboradores ponen de manifiesto que uno de los objetivos que persigue este ensayo es el de realizar una comparación entre una query, que es un grupo de genes que ha mostrado expresión diferencial en un experimento determinado, y las firmas que se obtienen de la base de datos CMap.

Estos autores desarrollaron un algoritmo basado en un dato cuantitativo denominado Connectivity Score, debido a que determinar el grado de similitud no arrojaba datos estadísticos de significancia respecto para determinar la magnitud del cambio de expresión o la especificidad de las relaciones observadas. La medida de este score constituye un marco estadístico que proporciona una cuantificación holística de la relación entre la query y el perturbagen.

El algoritmo que calcula el Connectivity Score cuantifica similaridades aplicando el estadístico Kolmogorov-Smirnov basado en ponderaciones, descrito anteriormente en un artículo previo del mismo autor, Subramanian [18]. Resumiendo, el valor del score será resultado de la diferencia de los Enrichment Scores (ES) de genes sobreexpresados y los inhibidos dividido entre 2 si tienen distintos signo los ES y 0 si son de igual signo. El ES refleja el grado que un conjunto de genes están sobrerrepresentados en los extremos de una lista de genes ordenada. Para realizar este cálculo es necesario disponer de valores de expresión

de genes que sean de referencia (los de una firma determinada) y otro grupo de genes que sean sobre los que se quiere determinar la similitud (los genes de la query). Sobre estos últimos genes se calcula la correlación que muestran con respecto a un fenotipo de referencia. Aplicando una serie de sumatorios se llega al resultado de ES, que no es otro que una diferencia entre probabilidades de que haya genes que están en los dos grupos de genes, y otros que sólo están en el grupo de la query. En conclusión, ES representa la máxima desviación de 0 cuando en el camino aleatorio estadístico.

La elección de Java para el desarrollo de la aplicación Web se basa en su utilización multiplataforma, de forma que puede ser ejecutado desde cualquier dispositivo que tenga una maquina virtual de Java, otorgando una gran flexibilidad de uso a los usuarios. Además, se trata de un lenguaje que está bien documentado y en continuo crecimiento, poseyendo gran cantidad de software disponible, de gran calidad y en muchos casos con una licencia de código abierto o de software libre para cualquier tipo de necesidad en una aplicación bioinformática.

Los paquetes de software CMap para Java (CMapJ), desarrollados a inicios de 2018 [16], ofrecen la posibilidad de ser integrados en una aplicación con un patrón arquitectura Modelo-Vista-Controlador, siendo el habitual en los últimos años en aplicaciones web. Este diseño de aplicación permite separar la lógica de negocios de los datos que se presentan al usuario y que están almacenados en una base de datos, y es el más indicado para la reutilización del código y el mantenimiento de aplicaciones.

El planteamiento de la aplicación como web se debe a las posibilidades de crecimiento que proporcionar hacerlo de esta manera. Los ficheros de configuración, las clases de la lógica de negocio del proyecto, además de los datos tanto de la base de datos para la funcionalidad de la aplicación como los ficheros .gtcx se encontrarán alojados en un servidor, que facilitará el acceso al usuario desde cualquier ordenador. (Figura 3)

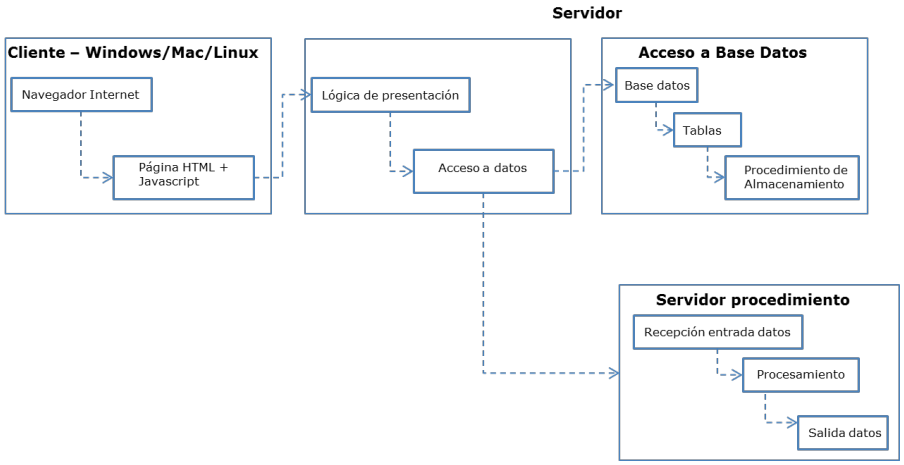


Figura 3. Arquitectura de la aplicación web

El entorno proporcionado por este esquema es modular permitiendo una escalabilidad sin límites; en función del crecimiento de negocio nos permite adecuar el entorno. Podremos implementar fácilmente la capacidad de procesamiento añadiendo nuevos nodos al clúster central y de procesamiento, multiplicando así su potencia y disponibilidad.

La creación de la interfaz de usuario se llevará a cabo utilizando el framework Java Server Faces 2.2 que permite desplegar las páginas web en el servidor donde se encuentra alojada la aplicación, ya que se trata de una API de servlets. Este framework puede utilizar distintas implementaciones que proporcionan una serie de componentes en forme de etiquetas. Los servlets se encargan de procesar las peticiones HTTP, obtener los datos de entrada, validarlos y convertirlos, colocarlos en los objetos del modelo, invocar las acciones del controlador y renderizar la respuesta utilizando el árbol de componentes. El siguiente esquema muestra el funcionamiento del framework [21] (Figura 4).

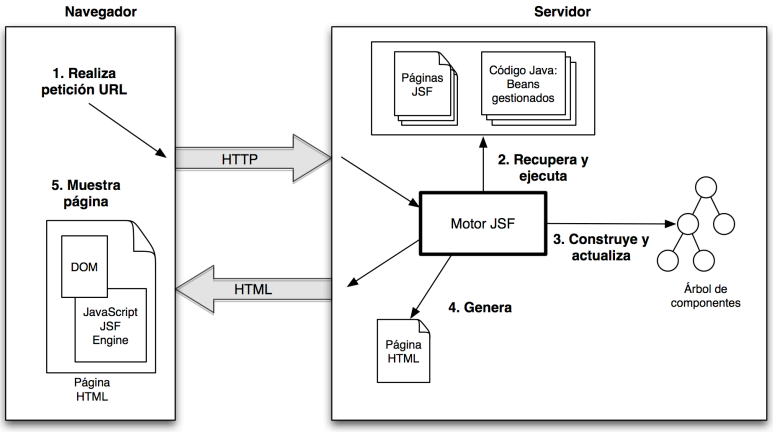


Figura 4. Funcionamiento JSF para generar una página web por primera vez

La interfaz de usuario, que no es otra cosa sino una vista en xhtml utilizando el lenguaje de componentes. Esta interfaz está en el servidor (en el caso de esta aplicación se ha escogido Tomcat) en el que se encuentra en forma de fichero de texto. Cuando el servidor recibe una petición desde el navegador a la URL donde está la página JSF, siguiendo el protocolo http, un servlet (motor JSF) procesa éste fichero y genera un árbol de componentes equivalente pero en forma de objetos de Java. Ejemplo, si en la página JSF se define un botón con la etiqueta <h:commandButton> el objeto que se creará será uno de la clase javax.faces.component.html.CommandButton. Tras esto, se ejecutará el código Java en el servidor para rellenar los elementos del árbol con los datos de la aplicación. Los componentes son los encargados de recoger los datos que introduce el usuario y de mostrar los resultados que los controladores generan a partir de los primeros. La forma de unir estos datos es a partir de otra clase de Java que muestra la anotación @ManagedBean y que se denomina habitualmente controlador, que será donde a través de métodos se procesen los datos de entrada para generar la salida adecuada a cada caso. Finalmente, a partir del árbol de

componentes construido y los datos generados, el servlet se encarga de generar una página HTML que se envía al navegador para mostrársela al usuario.

Esta aplicación web está programada en Java con el framework JSF, utilizando una base de datos MySQL para almacenar información relevante para su funcionamiento y se encontrará alojada en el servidor de aplicaciones web Tomcat.

Debido a la complejidad que supone extraer datos del maremagnun de informaciones disponibles sobre experimentos realizados, a los usuarios les puede resultar su labor investigadora complicada y laboriosa. De forma que la creación de un entorno que facilite el acceso a estos datos sin necesidad de conocimientos informáticos podría aumentar las oportunidades de descubrimiento.

1.2 Objetivos del Trabajo

Tomando como antecedentes lo anteriormente descrito para la realización de este TFM se plantean los siguientes objetivos generales, y para la consecución de estos los específicos.

1.2.1 Objetivos generales

1. Integración de las clases CMapJ dentro de un proyecto web para la utilización de ficheros .gctx.
2. Desarrollo de una interfaz de usuario que permita la interacción con los ficheros .gtcx y obtener la información contenida en ellos.

1.2.2 Objetivos específicos

1.1 Creación de proyecto web e inclusión de las librerías necesarias para permitir despliegue de la aplicación de manera local.
1.2 Exploración del funcionamiento de las clases de CMapJ para la lectura de ficheros .gtcx con distintos tipos de tamaño de fichero, de forma local.
1.3 Obtención de los tipos de metadatos a extraer de los ficheros .gctx utilizados durante las pruebas.
1.4 Determinación del tipo de inputs de datos para interrogar a los ficheros.
1.5 Concreción de la funcionalidad de la aplicación web.

En relación al segundo objetivo:
2.1. Diseño de la base de datos relacional SQL asociada al funcionamiento de la aplicación web.
2.2. Creación de las clases y vistas asociados al patrón Modelo-Vista-Controlador para la creación de la interfaz de usuario.
2.3. Creación e Implementación de estilos .css para el aspecto de la aplicación web.

2.4. Pruebas de la aplicación con introducción de inputs distintos tipos de inputs para la obtención de las diferentes posibilidades de output.
2.5. Subida de información a servidor, despliegue en él y pruebas que garanticen el funcionamiento de la aplicación.

1.3 Enfoque y método seguido

- Estrategia 1:
  Realizar un análisis funcional de la aplicación web: definir el tipo de inputs que se van a proporcionar a la aplicación, por ejemplo, listado de genes upregulated-downregulated obtenidos de un análisis previo en un experimento de exposición de un tipo celular a una sustancia que ha provocado cambios transcripcionales. Definir que resultados deseamos conocer.

  Tras definir los input y outputs se realizaría el diseño de la base de datos para configurar los módulos que tendría la aplicación web con el objetivo de mostrar los resultados.

  Explorar las clases CMapJ con el objetivo de poder obtener los resultados deseados a partir de los inputs introducidos y su adaptación a ser utilizadas en un entorno web.

- Estrategia 2:
  Analizar con detalle las clases CMapJ para determinar cómo se puede obtener el mayor rendimiento de los ficheros .gtcx, estableciendo que inputs y outputs se pueden obtener, con el mayor rendimiento de procesamiento.

  Adecuación de estas clases para su utilización en un entorno web.
  En función de los inputs/outputs obtenidos, proceder a la fase de análisis de la aplicación con el objetivo de definir las tablas que van a configurar la base de datos asociada al funcionamiento correcto de la aplicación.

  Desarrollo de las clases, vistas y hojas de estilo que permitan de la forma más intuitiva y ergonómica la introducción de datos a interrogar y la visualización de los resultados obtenidos tras el procesamiento del fichero .gctx.

  Para la realización de este proyecto se va a seguir la segunda estrategia. Al haber sido desarrolladas por terceros clases CMapJ para un entorno ejecutable en consola será preciso una exploración profunda de su funcionamiento y de la utilización correcta de las librerías con las que vienen, para poder ser adaptadas a un entorno web.

  Finalmente, para un mayor aprovechamiento del tiempo de trabajo del TFM es preferible un mayor conocimiento de lo que ofrecen estas clases para poder realizar un planteamiento de la aplicación a posteriori, para

evitar posibles futuras modificaciones de base de datos y de entidades asociadas a las tablas.

1.4 Planificación del Trabajo

1.4.1 Recursos

Los recursos a utilizar para llevar a cabo el prototipo de aplicación para comparar firmas son los siguientes:

- Ordenador con al menos 8 GB de RAM.
- Entorno de desarrollo (IDE) para Java, por ejemplo, NetBeans 8.2
- JDK 1.8
- Base de datos MySQL 5.7
- Servidor de aplicaciones web Tomcat.

1.4.2 Tareas y planificación temporal

Para la consecución de los objetivos enumerados en el apartado 1.2.2 las tareas que se han de llevar a cabo son las siguientes:

- Objetivo 1.1:
  - Tarea 1: Creación con un entorno de desarrollo apropiado para Java un proyecto web, Java Web. Se utilizará el IDE Netbeans en su versión 8.2, con el framework Java Server Faces 2.2 como tecnología para el desarrollo en la parte cliente-servidor.
  - Tarea 2: Determinar la configuración del proyecto: contexto, esquema de directorios, servidor en el que se realizarán los despliegues (presumiblemente un Apache Tomcat versión 5).
  - Tarea 3: Instalación de librerías de componentes (Primefaces, Tomahawk, Myfaces, Facelets) hasta conseguir un despliegue de aplicación en local sin errores.
- Objetivo 1.2:
  - Tarea 4: Descarga del código CMapJ desde Github e inclusión en el árbol de directorios creado en la aplicación.
  - Tarea 5: Inclusión de las librerías con tecnología HDF5 (desarrolladas por el Grupo HDF) necesarias para utilizar las clases de lectura y creación de ficheros .gtcx.
  - Tarea 6: Prueba de las clases ejemplos proporcionadas por los desarrolladores de CMapJ para testear su funcionamiento.
  - Tarea 7: Realizar distintas lecturas sobre un fichero .gtcx sencillo para determinar la forma de obtener la información que contiene, tanto datos de expresión de la matriz de uno de los directorios, como los metadatos alojados en el otro directorio. (De forma local).
  - Tarea 8: Realizar las lecturas determinadas como fundamentales es en la tarea anterior sobre el fichero .gtcx que se utilizará para desarrollar la aplicación.
- Objetivo 1.3:
  - Tarea 9: Realizar un resumen que refleje el tipo de metadatos que existen en el fichero.

- Tarea 10: Determinar la manera de obtener de forma ordenada aquellos valores de Z-Score mayores y menores para un gen.

- Objetivo 1.4:
  - Tarea 11: Establecer la relación entre los identificadores de columnas y filas obtenidos para los valores deseados del Z-score y relacionarlos con los valores de los metadatos para obtener la información del experimento correspondiente a ese valor.
  - Tarea 12: Determinar si dada un valor de identificador de columna qué metadatos se pueden obtener de la fila correspondiente, y viceversa.
  - Tarea 13: Testear con diferentes posibilidades de metadatos como entrada: genes, valores de Z-score, líneas celulares, compuestos… y escoger una de ellas para proseguir con el desarrollo. El metadato gen tendrá preferencia sobre el resto.
- Objetivo 1.5:
  - Tarea 14: Dado un metadato de entrada (presumiblemente gen) determinar cuáles son los outputs posibles (condiciones experimentales, líneas celulares…) y escoger dos de ellos para mostrar los resultados tras la búsqueda en el fichero.
  - Tarea 15: Realizar un esquema de la interfaz de la aplicación: validación de usuarios, menú principal, mantenimientos de tablas maestras, módulo de procesamiento, módulo de visualización, módulo de histórico de búsquedas…
- Objetivo 2.1:
  - Tarea 16: La base de datos que sustentará la integridad del proyecto será una base de datos relacional SQL, MySQL. En ella se definirán que tablas son necesarias para el funcionamiento de la interfaz de usuario: establecer claves primarias y campos.
  - Tarea 17: Establecer la relación entre tablas: establecer las claves foráneas.
  - Tarea 18: Generar las sentencias SQL y ejecutarlas para la creación de la base de datos.
- Objetivo 2.2:
  - Tarea 19: Generación de las entidades (ficheros .java) provenientes de las tablas de base de datos.
  - Tarea 20: Creación de las vistas (ficheros .xhtml) asociadas a los diferentes módulos/menús de la aplicación.
  - Tarea 21: Creación de los controladores asociados a las vistas para recoger y mostrar la información deseada por el usuario (ficheros .java)
- Objetivo 2.3:
  - Tarea 22: Generación de hojas de estilos (ficheros .css) y aplicación a las vistas.
- Objetivo 2.4:
  - Tarea 23: Adecuación del código de las clases de CMapJ para realizar la introducción de datos y mostrar la salida de los resultados a través de las vistas en lugar de por consola.

- Tarea 24: Búsqueda de datos de experimentos que se encuentre en la base de datos de Gene Expression Omnibus (GEO), para que realicen la función de controles positivos de la aplicación.
- Tarea 25: Realización de pruebas de entrada y comprobación de los datos de salida con los datasets escogidos para la validación de la aplicación.

• Objetivo 2.5:
  - Tarea 26: Instalación del software del servidor (presumiblemente Tomcat) para poder realizar el despliegue de la aplicación en el equipo que se vaya a utilizar.
  - Tarea 27: Generación de la base de datos en el servidor con las sentencias SQL previamente codificadas.
  - Tarea 28: Subida de los ficheros .gctx explorando diversas posibilidades: subida por ftp, copia a través de VPN.
  - Tarea 29: Realización de test de validación del funcionamiento final de la aplicación.

Los hitos parciales a conseguir en las dos PECs durante el desarrollo del TFM:

1. Integración funcional de las clases CMapJ en un proyecto web de Java. Utilización y extracción de contenido de metadatos de ficheros .gctx.
2. Creación interfaz de usuario de proyecto web funcional. Validación de la aplicación desde el punto de vista de usuario para interrogación de datos. Despliegue de aplicación en servidor para ser utilizada fuera del entorno local.

A continuación se muestra el cronograma que se ha seguido para completar las tareas que se siguieron en el desarrollo del prototipo, así como los hitos parciales alcanzados en las PECs entregadas :

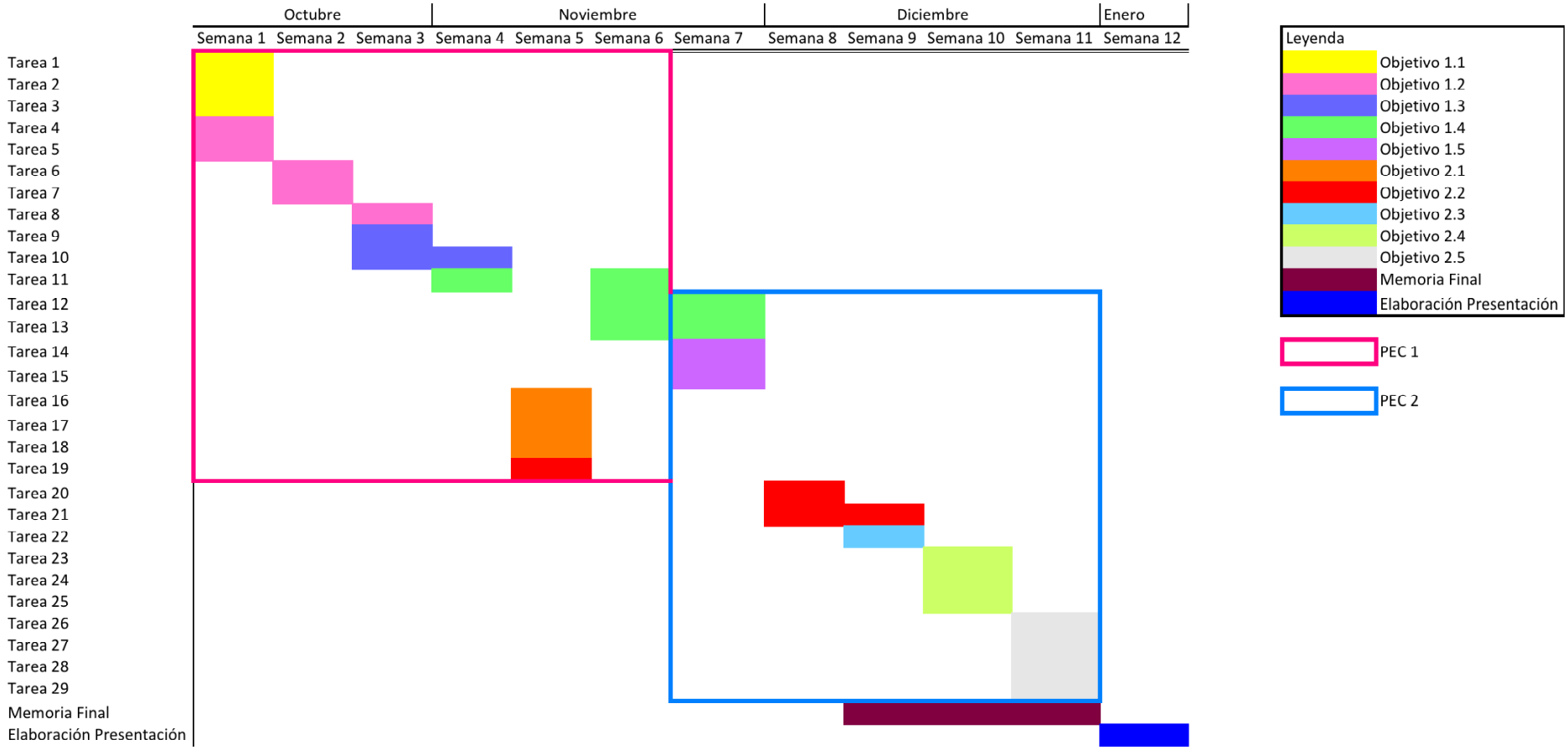


Figura 5. Diagrama de Gantt de la planificación de tareas e hitos a entregar.

1.5 Breve sumario de productos obtenidos

El producto obtenido en la presente memoria se trata de un prototipo de aplicación web que permite a los usuarios comparar sus resultados de expresión diferencial génica con los ensayos realizados en el proyecto L1000 del consorcio LINCS.
De esta manera el usuario podrá conocer a qué otras firmas son más similares su patrón de expresión encontrado, obteniendo así el contexto experimental donde se obtuvieron y la funcionalidad asociada a los genes con mayor alteración de expresión.

1.6 Breve descripción de los otros capítulos de la memoria

- Estructura de la aplicación: las aplicaciones web desarrolladas en Java deben presentar un árbol de directorios que facilita al servidor encontrar los ficheros de configuración y los que tienen el código correspondiente a la programación de la aplicación. Se describe el contenido de los directorios y el papel que desempeñan los ficheros. Para que la aplicación funcione los ficheros deben estar ubicados en los directorios adecuados para que las librerías sean capaces de recurrir a rutas relativas para invocarlos.

- Explorando los ficheros .gctx con las clases y librerías de CMapJ: se estudia el contenido de las distintas clases de CMapJ, centrándose en especial en la GctxReader que es la que sirve de eje para poder explorar los ficheros. Es esencial conocer el funcionamiento de estas clases para poder extraer la máxima información posible del fichero, de una forma adecuada, así como conocer las limitaciones que el código desarrollado tiene para poder ofrece al usuario, que desea comparar su fichero de resultados de experimentación, la información de la forma más concreta y acertada posible.

- Base de datos: se muestra el diseño de la base de datos relacional MySQL que respalda el prototipo de aplicación web. Se muestran las tablas, los campos de las tablas, incluyendo claves primarias y foráneas que determinan la relación que existe entre las tablas. La utilización de una base de datos permite que la aplicación web tenga un sistema de control mediante usuario para poder mostrarle un histórico de resultados de comparaciones previas realizadas según distintos parámetros.

- Algoritmo comparador: se describe paso a paso el algoritmo que la aplicación utiliza para poder comparar el conjunto de genes aportado por el usuario con los grupos de firmas que se encuentran dentro del fichero .gtcx. Este algoritmo da como resultado un grupo de experimentos que presentan mayor similitud según una serie de parámetros escogidos por el usuario.

Es el núcleo de la aplicación, ya que la aplicación gira en torno al output resultante de implementar este código.

- Desarrollo de la arquitectura Modelo-Vista-Controlador: se explica cuales son los principales ficheros que forman parte de la arquitectura interna que presenta el prototipo de aplicación web. La interconexión de los ficheros que se encajan en cada una de las distintas partes es fundamental para que pueda existir el flujo de información entre lo que usuario realiza a nivel de navegador y lo que llega a almacenarse en la base de datos con respecto a los resultados obtenidos.

- Descripción funcionamiento prototipo: se detalla a través de una serie de capturas de la interfaz creada como funciona la aplicación y los resultados que el usuario puede esperar de ella.

- Validación con datos reales: descripción de los resultados obtenidos al realizar una comparativa de genes diferencialmente expresados en distintos cánceres para demostrar la funcionalidad del prototipo desarrollado.

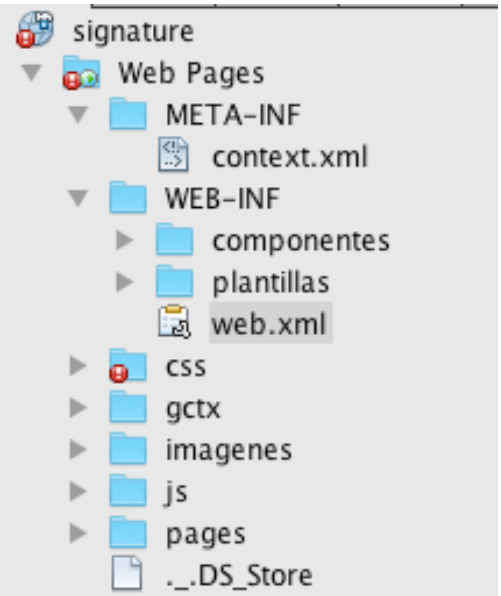
2. Resto de capítulos

2.1 Estructura de la aplicación

El prototipo se ha desarrollado en el lenguaje de programación Java, por lo cual se escogió el entorno de desarrollo Netbeans, en su versión 8.2, que permite la programación con Java 1.8, última versión de este lenguaje.

La aplicación web se compone de ficheros, que pueden ser clases o archivos de configuración y librerías. El tipo de proyecto sobre el que está basada la aplicación es un Java Web – Web Application.

La estructura está basada en un árbol de ficheros como el que aparece en las figuras 6, 7 y 8.



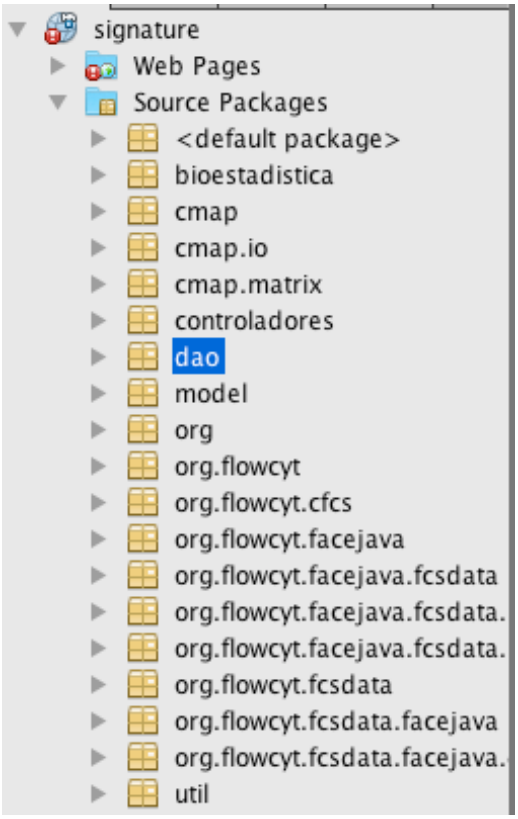
Por un lado se encontrarían contenidos en un directorio (Web Pages) todo a lo que se necesita acceder para configurar las vistas o páginas web, así como las propias vistas y los recursos que necesita la aplicación para su funcionamiento:

- En META-INF, fichero de configuración del contexto de la aplicación.
- En WEB-INF, carpetas con plantillas y vistas consideradas componentes. También un fichero con los parámetros de configuración de las librerías asociadas al Framework de desarrollo de la parte web.

Figura 6: Distribución directorios en Web Pages.

- En css, los ficheros .css con las referencia a los estilos a cargar en las páginas web.
- En gctx se hayan los ficheros de referencia que van a utilizarse para extraer la información.
- En imágenes, las figuras que se utilizan en la aplicación.
- En js, una serie de ficheros javascript también utilizados en la aplicación.
- En pages, los ficheros .xhtml donde se ha llevado a cabo la programación de las páginas a las que el usuario accede.

Los ficheros relativos a lo que es la programación se encuentran en la capa Source packages, en ellos radica la actividad principal de la aplicación ya que es desde donde se realizan los procesos que hacen funcionar la aplicación (Figura 7).



Los directorios que se encuentran aquí contenidos se pueden agrupar en los CMapJ y los desarrollados en el proyecto.

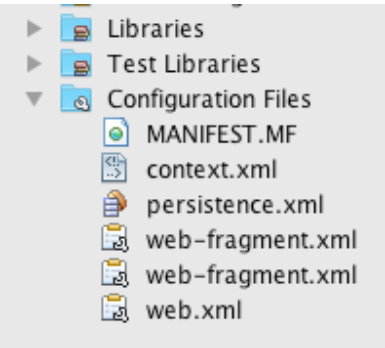
Los CMapJ son los ficheros obtenidos desde el repositorio GitHub, y como se ha nombrado anteriormente, fueron desarrollados por el HDF group para permitir el acceso a la información contenida en los archivos .gctx. Estos serían los que están contenidos en los paquetes: cmap, camp.io, camp.matrix, org, orgflowcyt, org.flowcyt.cfcs, org.flowcyt.facejava, org.flowcyt.facejava.fcsdata, org.flowcyt.facejava.impl, org.flowcyt.facejava.io, org.flowcyt.fcsdata, org.flowcyt.fcsdata.facejava y .exception.

Figura 7: Conjunto de paquetes que forman parte del código Java.

El resto de paquetes han sido creados durante la elaboración del prototipo con distintas funciones:

- Bioestadística: recoge clases que contienen los algoritmos estadísticos necesarios para poder realizar los cálculos pertinentes en la comparación de firmas.
- Controladores: son clases de Java que se encuentran en relación directa con las vistas. Son capaces de recoger la información que el usuario introduce en las páginas web y también de mostrar la información en ellas. Reciben el nombre de ManagedBeans.
- Dao: recoge las clases e interfaces creadas para poder acceder al objeto que java que muestra y envía información a la base de datos con la que se ha conectado la aplicación para recoger información.
- Model: contiene todas las clases de Java que representan a las tablas diseñadas en la base de datos. Estas clases reciben el nombre de Entidades.
- Util: este paquete contiene clases e interfaces que realizan funciones que se utilizan en más de un controlador, reutilizando código de esta manera.

El resto de directorios, Librerías y Configuration Files (Figura 8) se tratan de directorios que contienen las librerías que necesitan las clases para poder llevar a cabo sus funciones (archivos compilados en .jar) y los archivos de configuración para poder establecer conexión con la base de datos (persistence.xml) y los ficheros que se han nombrado



anteriormente en Web Pages. En el Anexo 1 se encuentra la relación de todas las librerías utilizadas para el desarrollo del prototipo.

Figura 8: Resto de directorios y ficheros contenidos en ellos que son necesarios para el despliegue de la aplicación web en un navegador.

2.2 Explorando los ficheros .gctx con las clases y librerías de CMapJ.

Como se ha descrito en la introducción los ficheros .gctx son un tipo de ficheros binarios que contienen información de resultados de experimentos en una matriz de datos y de los metadatos que recogen las condiciones experimentales por las cuales se obtuvieron esos resultados, así como las anotaciones relativas a los genes asociados.

La clase desarrollada más importante para poder leer estos ficheros es la “GctxReader”. Para poder acceder a la información contenida en el fichero .gctx es necesario crear un objeto de esta clase. El método constructor de esta clase recibe como parámetro el path donde se encuentra el fichero, creando a su vez en dicha clase un objeto de la clase FileFormat con configuración hdf5. En el caso de esta aplicación, los ficheros .gctx están localizados en una carpeta dentro del directorio de Web Pages, ya que los archivos allí alojados son susceptibles de ser invocados desde cualquier controlador del proyecto.

Una vez que se ha creado el objeto de la clase GctxReader ya es posible acceder a las propiedades y métodos que se encuentran codificados en dicha clase. En el desarrollo de este prototipo, el método más importante a utilizar es el denominado **read()**. Este método se encarga de crear a su vez otro objeto de la clase Dataset, parseando el objeto de la clase FileFormat tipo hdf5. Tanto las clases FileFormat como Dataset se encuentran compiladas en la librería ncsa.hdf.object. Del objeto dataset creado se extraen el número de filas y columnas gracias al método **getDims()** de la clase Dataset, y de esta manera se hace una primera comprobación de que el fichero al que se esta accediendo tiene información.

A continuación en este método se crean 3 objetos vacíos: un nuevo objeto Dataset, un objeto de la clase Object y un arraylist de objetos tipo Hyperslab. Al haber creado un objeto FileFormat tipo hdf5 dentro de la clase GctxReader, se pueden seleccionar grupos de datos del total contenido en el fichero, para acceder a ellos. Los objetos de la clase FileFormat del tipo hdf5 disponen de métodos para hacer esta selección,

uno de ellos es a través de selección de regiones Hyperslab. Por medio de código escrito en C (un nivel de programación más bajo que Java) que selecciona puntos que están contiguos en el espacio, es posible acceder a la información que se guarda en estos espacios. En los dos objetos Hyperslab, por el momento, se almacenan los índices de las columnas o filas, indicando si se refiere a columna o fila ese dato.
A continuación, se almacenan los datos que contiene el objeto FileFormat del tipo hdf5 en el objeto de la clase Object a través de los objetos Hyperslabs creados. Este Object será convertido al objeto de la clase DataSet gracias a la creación de otro objeto que hereda de la clase AbstractDataset y que es capaz de extraer los valores que hay en una determinada posición de un Hyperslab.
Una vez almacenados en el objeto de la clase Dataset los valores de la matriz de datos, se obtienen los metadatos. Para ello se extrae la información de filas y columnas con la ayuda de la clase Metamodel, que permite guardar esta información en objetos para filas y columnas. Finalmente, el método **read()** devuelve un objeto de la clase Dataset, pero al que se la implementado métodos de otra nueva clase, denominada DatasetAdapter que es capaz de combinar la información de filas, columnas y los datos de la matriz.

El método **read()** tiene en su código la posibilidad de recoger excepciones para controlar errores. Uno de ellos es el ArrayToBigException, que lanza una excepción y detiene la creación del objeto Dataset cuando se está pasando la información desde el objeto FileFormat al Object. En el manejo de esta excepción se intenta hacer partes más pequeñas de la información que se está recuperando en la matriz, y se intentan crear objetos Hyperslabs de dimensiones más reducidas para poder abarcar más matriz. Sin embargo durante la prueba de ficheros .gctx de tamaño del orden de GB el control de excepciones ofrecido resultó insuficiente para salvar el error resultante. De modo, que se se tomó la decisión de utilizar el fichero test_data proporcionado en clue.io para la creación del prototipo. Los datos contenidos en este subgrupo de datos se corresponden con datos contenidos en los datos de la fase I del proyecto LINCS en la serie GSE92742 de GEO Omnibus. Se compone de los 978 genes que conformar los landmark genes y los 1000 primeros experimentos.

Cabe destacar, que aunque en esta memoria se esté describiendo únicamente el funcionamiento de la clase GctxReader ha quedado de manifiesto que en ella se utilizan objetos de otras muchas clases, que a su vez requieren de otras terceras clases y librerías, que hacen que todas las clases y librerías anteriormente nombradas sean necesarias para el funcionamiento de ésta en particular.

Una vez que se dispone de un objeto de la clase Dataset es posible extraer la información para poder ofrecérsela al usuario. Los datos relativos al contenido de los metadatos de las columnas y filas se extrae a partir del método **getColumnMetadata().getColumnName() y getRowMetadata().getColumnName()**, respectivamente. A través de

estos métodos en las etapas iniciales del desarrollo del proyecto se pudo obtener el tipo de información al que se iba a poder tener acceso (Figura 9).

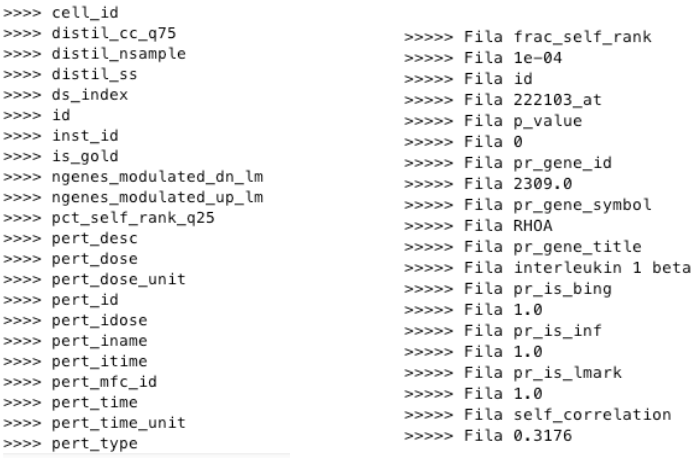


Figura 9. Metadatos contenidos en el fichero .gtcx utilizado en la aplicación para ser consultado.

Así pues se decidieron cuáles iban a ser los datos de columnas y las filas que iban a resultar informativos para el usuario. Con respecto a las columnas estos datos fueron:

- Tipo de célula sobre la que se realizó el ensayo (cell_id).
- Tipo de perturbación (compuesto químico, siRNA u otro compuesto biológico) (pert_type).
- Nombre de la perturbación (pert_iname).
- Dosis (pert_idose).
- Tiempo de exposición (pert_itime).
- Contenido de oro (is_gold).
- Identificador del experimento (id).
- Descripción de la perturbación (pert_desc).

Mientras que para las filas se optó únicamente por extraer el nombre del gen (pr_gene_symbol).

Para acceder a los datos referentes a estos metadatos ha de utilizarse otro método al que se le pasan por parámetros el nombre de la fila/columna y la posición que ocupan.
Por ejemplo: **dataset.getRowMetadata().getValue(j,"pr_gene_symbol")**, devuelve el símbolo del gen en la posición j de las filas.

Para obtener el valor en una posición concreta de la matriz de datos para una columna i, en una fila j, ha de invocarse al método:
dataset.getValue(j, i). Para obtener todos los valores sería preciso llamar a este método en el interior de dos bucles anidados, uno que recorriera las j filas y otro que recorriera las i columnas.

Una vez que se han extraído los datos, es necesario “cerrar” el objeto Dataset con el que se ha trabajado, por medio del método **close()**. Si se deseara volver a extraer alguna otra información habría que volver a crear el objeto dataset por medio del método **read()** y cerrarlo al finalizar.

Durante las pruebas de lectura y extracción de información de los ficheros .gtcx se detectó otra dificultad en relación a una librería dinámica. Como se ha nombrado anteriormente, las clases de CMapJ fueron diseñadas para su uso a nivel de consola en máquinas individuales, no para el uso en forma de aplicación web. De forma que una de las principales librerías para obtener información de los ficheros .gctx, libjhdf5, dependiente del sistema operativo en el que se aloje la aplicación sólo permitía una sola carga al iniciar la aplicación. Para subsanar el problema durante el desarrollo, esta librería se cargó siempre desde la línea de comandos de la máquina virtual al iniciar el servidor. Sin embargo, cómo sólo podía cargarse una única vez al arrancar el servidor al realizar las pruebas pertinentes se prescindió de los redespliegues y se optó por reiniciar el servidor cada vez.

2.3 Base de datos

Para que los usuarios que utilicen la aplicación para realizar comparaciones con sus propios experimentos puedan tener disponible los datos almacenados en el fichero .gctx contra el que lanzan sus consultas, estos datos deberán ser guardados de alguna manera mientras la aplicación ejecuta sus algoritmos.

Por lo que existe la necesidad de crear clases específicas para almacenar estos datos en forma de propiedades. Además, para poder mostrar esta información a posteriori para aquellos usuarios que hubieran realizado comparaciones previas es preciso que estos datos queden almacenados de alguna manera más allá de la sesión en la que están.

Por todo esto se planteó diseñar una base de datos que almacenara las comparaciones realizadas, y además, ofreciera la posibilidad de mostrar información más detallada al usuario sobre su comparación por medio de la utilización de tablas maestras albergando datos de interés.

El prototipo aquí presentado está basado en una base de datos MySQL 5.7 cuyo script de generación se muestra en el Anexo 2, pero el esquema del diseño es el que aparece en la figura 10.

La base de datos está compuesta con 7 tablas. Las tablas CellTypes, Anotación y Geneontology son tablas maestras, es decir, que contienen datos para el funcionamiento del resto de la aplicación y el resto de tablas almacenan datos en los que hay un intercambio de información con la aplicación. A continuación se repasa el contenido de cada una de ellas.

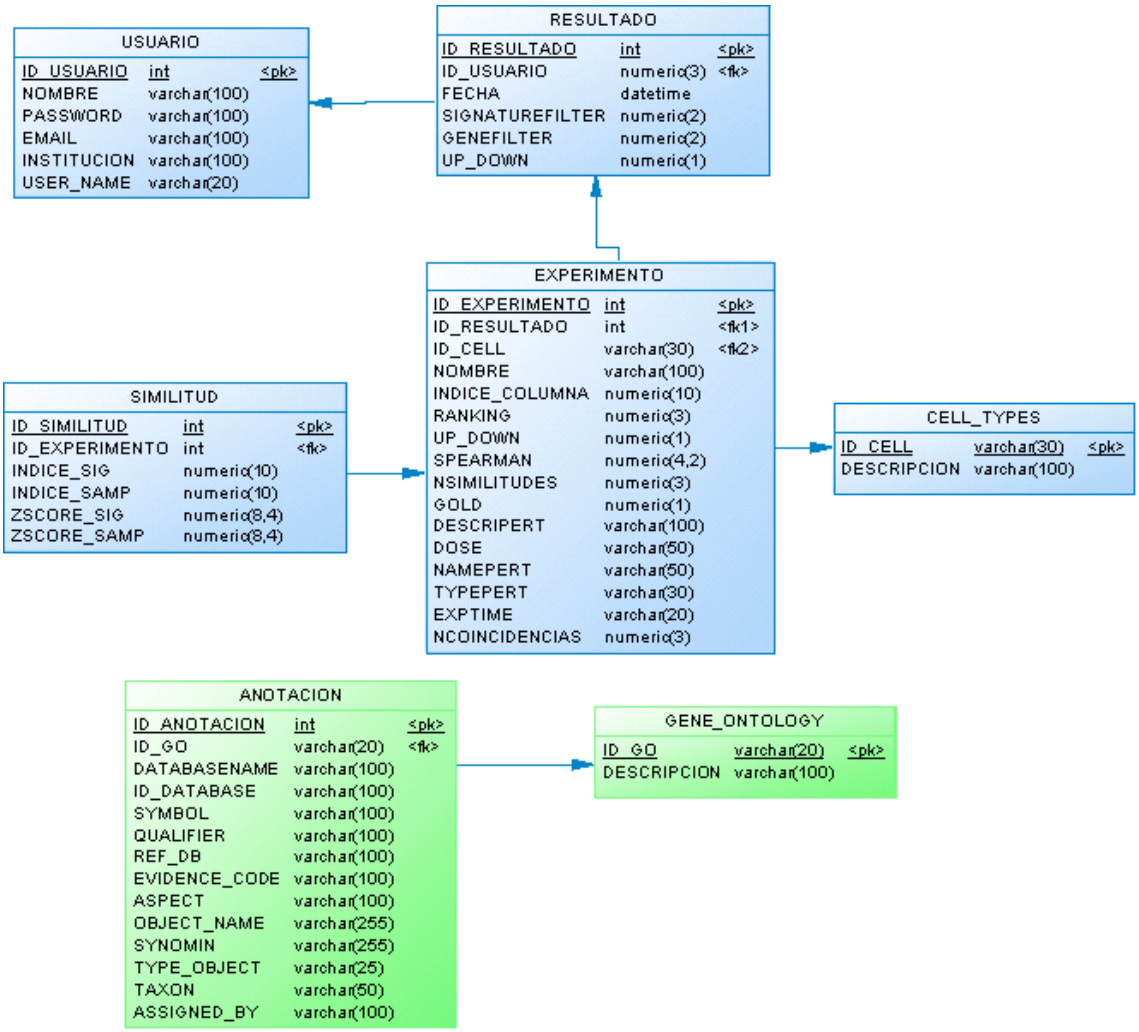


Figura 10. Diseño de la base de datos Signature.

- USUARIO: recoge los datos personales de los usuarios que se registren en la aplicación, que además le dan acceso a ello. Su clave primaria se asigna automáticamente al autoincrementase al registrarse el usuario.
- RESULTADO: en esta tabla se almacenan los datos referentes a las consultas, con los parámetros correspondientes, realizadas por usuario con anterioridad, esta relación viene dada por el campo ID_USUARIO, que es la clave foránea de usuario en esta tabla. Al guardar el experimento su clave primaria aumenta en un número al ser autoincremental.
- EXPERIMENTO: Cada comparación realizada arrojará las firmas más parecidas a la firma que el usuario trata de comparar. Esos datos, cuyos metadatos se han nombrado en el apartado anterior serán almacenados en esta tabla, y estarán relacionados con la consulta realizada por el usuario a través de la clave foránea ID_RESULTADO. La clave primaria es un número que se le asigna de forma programática.
- SIMILITUD: en esta tabla se recogerán aquellos genes que han sido señalados por similitud con la consulta realizada en cada firma recogida en Experimento, por lo que la relación es 1 a n, varias similitudes pueden pertenecer a un mismo experimento, siendo la clave

foránea ID_EXPERIMENTO. La clave primaria es un número que se le asigna de forma programática.

- CELL_TYPES recoge la clave del tipo celular utilizado en los experimentos del fichero .gtcx y la descripción correspondiente a la línea celular. Esta tabla se asocia con la de EXPERIMENTO a través de una clave foránea por la columna ID_CELL.
- GENE_ONTOLOGY y ANOTACIONES. Estas tablas no están asociadas a ninguna de las tablas diseñadas, aunque GENE_ONTOLOGY es clave foránea en ANOTACIONES, a través de la columna ID_GO. Estas tablas recogen la información de las anotaciones que se encuentra disponible en http://geneontology.org. En cuanto a ANOTACIONES, está cargada con los datos procedentes del fichero goa_human.gaf, que contiene las anotaciones para la especie humana.

En un primer momento se pensó unir esta tabla a SIMILITUD, ya que la columna GENE_NAME recoge la misma información que SYMBOL en ANOTACIONES. Sin embargo, SYMBOL no podía ser clave primaria en ANOTACIONES ya que un mismo gen puede tener varias anotaciones, y por tanto no podía ser clave foránea en SIMILITUD. La tabla GENE_ONTOLOGY se pobló a partir del fichero de datos geneontology.dat proporcionado por el director Guerau Fernández, compuesto por un listado de identificadores de la ontología y otra columna con la descripción.

Inicialmente la base de datos estaba compuesta únicamente por las tablas USUARIO, RESULTADO, EXPERIMENTO y SIMILITUD. Sin embargo, a medida que avanzó el desarrollo se decidieron incluir las tablas maestras nombradas anteriormente para ofrecer más información al usuario sobre el resultado obtenido y ampliar, de esta manera, la funcionalidad de la aplicación. En el caso de

También se introdujeron campos nuevos en las tablas RESULTADO, SIMILITUD y EXPERIMENTO para poder recoger información de los metadatos de cada experimento del fichero .gctx.

Por último, también se tuvo que cambiar el tipo de clave primaria en EXPERIMENTO y SIMILITUD y que dejara de ser autoincremental, como se había planificado anteriormente, ya que al intentar guardar los elementos desde una lista en java se obtenía un error de base de datos, al no poder recuperar correctamente la última clave almacenada, de forma que esta clave se autoincrementa desde el código y se asigna.

2.4 El algoritmo comparador.

El objetivo principal que persigue el prototipo es el de ofrecer al usuario que lo utilice la posibilidad de comparar los resultados de expresión diferencial de genes que han sido obtenidos en su experimentación con los registrados dentro del proyecto L1000 de LINCS, es decir, determinar las firmas más similares de este proyecto con la firma que ha obtenido él. De esta manera, el usuario dispondrá de la información experimental

que llevó a la obtención de dichas firmas y comparar así con sus propias condiciones experimentales.

Aunque ya se ha descrito en la introducción que el algoritmo utilizado por el proyecto L1000 es el Connectivity Score [17], en esta aplicación, al tratarse de un prototipo de carácter sencillo, la aplicación de este algoritmo resultaba demasiado complicada, pudiendo ralentizar la aplicación al necesitar elevado tiempo para realizar los análisis.

Ante esta dificultad se plantearon dos alternativas con distintos niveles de complejidad:

- Llamar a un script de R por línea de comandos que invocara al paquete de R, GSEA, el cual es capaz de calcular los valores del Enrichment Score, aplicando Kolmogorov-Smirnov con ponderaciones, como se realiza en el artículo [17].
- Llevar a cabo una ordenación en Java, basada en otro criterio diferente de determinación de la similitud, aunque menos potente que el descrito anteriormente.

Finalmente se optó por el segundo criterio por la dificultad a nivel de programación que consistía en invocar a otro lenguaje desde la máquina virtual de Java.

Este criterio nuevo de determinación de similitud, es un criterio cualitativo, mediante el cual la aplicación devuelve una serie de firmas contenidas en el fichero respecto a los siguientes parámetros:

- El usuario podrá escoger cuantos genes sobreexpresados/inhibidos quiere comparar.
- El usuario podrá escoger el número de firmas más similares a la que el aporta en la comparación.

El código correspondiente a este algoritmo se encuentra descrito en el Anexo 3. El primer paso es cargar el grupo de genes aportado por el usuario, que se ha realizado a través de la subida de los datos desde un fichero .xls. Estos datos se encuentran guardados en una estructura de Java conocida como mapa, la cual es similar a los diccionarios en otros lenguajes como Python o Perl. En este mapa clave recoge el nombre del gen, y el valor el Z-score correspondiente para ese gen. A continuación, a través de un método del paquete Utils se ordenan de mayor a menor Z-score los elementos del mapa. Finalmente, se realiza un grupo de menor tamaño de genes y se almacena en un nuevo mapa según el parámetro de genes a comparar por el usuario y el tipo de regulación, si sobreexpresión o inhibición del gen.

El siguiente paso consiste en cargar todos los datos de todas las firmas y sus metadatos en un objeto de la clase Dataset utilizando la clase GctxReader descrita anteriormente. Para ello deben recorrerse todas las columnas y filas del objeto y extraer la información deseada. La forma de realizarlo es a través de dos bucles *for* anidados desde 0 hasta el número de columnas/filas que tenga el dataset, siendo el más exterior el

de las columnas, ya que cada columna se corresponde a una firma distinta. Cada vez que se sale del bucle de filas, tres variables se reinician, la del objeto Experimento, la del objeto Similitud y el arraylist de Similitudes.

Dentro de los bucles anidados el objetivo es recuperar, en primera instancia, únicamente los datos relativos al símbolo del gen y a su Z-score, y guardarlos en un mapa. De la misma manera que se realizó para el mapa con los datos a comparar por el usuario, este mapa también es ordenado de mayor a menor valor de Z-score y dividido en grupos más pequeños en función de los parámetros escogidos.

En este punto se dispone de dos mapas filtrados, uno con datos del usuario y otro con datos obtenidos del fichero .gctx. Ambos ordenados según mayor Z-score. Es entonces cuando se procede a comparar el mapa del usuario con el mapa correspondiente para esa columna de datos determinada. Se recorre el mapa de la muestra gen a gen, y si el mapa de la firma contiene el gen, se suma un entero a una variable. Finalmente, si el valor de la variable es mayor de 0 se cumple la condición para pasar al siguiente paso de la comparación.

Si se cumple la condición anterior se van añadiendo propiedades del experimento al objeto Experimento: el objeto Resultado al que pertenecen, si se trata de un dato Upregulated o Downregulated, el nombre identificador del experimento, y la columna a la que pertenece. Estos dos últimos datos se extraen de los metadatos de columna del fichero .gctx en la que el bucle se haya en ese momento. Además, se pueden añadir otros metadatos complementarios si se han escogido como opciones por el usuario. Estos datos pueden ser la descripción del pertubagen, el tiempo de exposición, el nombre del pertubagen, la dosis, si contiene oro, el tipo del que se trata o el identificador de la célula sobre la que se realizó en ensayo.

A continuación, y por medio de dos bucles anidados que recorren los dos mapas que contienen las firmas que han sido filtradas, las del usuario y las del fichero, se seleccionan aquellos genes que están en ambas. El mapa exterior se trata del mapa del fichero, y cada vez que se vuelve a entrar se crea un objeto nuevo Similitud y se reinicia a cero una variable contador. Si existe coincidencia de genes, se procede a asignarle los valores a las siguientes propiedades del objeto Similitud: nombre de gen, posición que ocupa en la ordenación realizada en la muestra, los valores de Z-score en la muestra y en el fichero, y el valor de la clave primaria que va aumentando a medida que se entra al bucle. Fuera del bucle más interno se aumenta el valor del contador y se añaden las propiedades del número de posición que ocupa dicho gen en el mapa del fichero y el experimento al que está asociada esa similitud. Se comprueba si el nombre del gen se ha añadido al objeto Similitud, si es así, se añadirá ese objeto a un arrayList de objetos de tipo Similitud.

Cuando se han terminado de comparar los dos mapas, el objeto Experimento al que se le han añadido anteriormente algunas propiedades recibe también como propiedad el arrayList de objetos del tipo Similitud y el número de genes coincidentes que ha habido entre el mapa con los resultados del usuario y el del fichero, esté número no es otro más que el número de elementos que hay dentro del array. Finalmente, ese objeto Experimento se añadirá a un arrayList de objetos de tipo Experimento, y entonces, se sale el bucle que recorre esa columna en concreto del dataset, para pasar a valorar el siguiente y repetir el proceso que se ha descrito en este apartado.

Una vez que se dispone de un arrayList que contiene todos los experimentos que cumplen las condiciones determinadas por los parámetros es preciso realizar un filtrado más sobre esos experimentos que depende de otro parámetro que se solicita al usuario, el número de firmas similares a la suya que quiere visualizar. Por tanto hay que realizar una ordenación de los experimentos en el arrayList de Experimentos.

Para realizar esta ordenación se recorren los elementos del arrayList de experimentos con un bucle en el que se añade el a un nuevo mapa, el cual almacenará un arrayList en la parte del valor. En la clave el número de genes coincidentes en el arrayList de Similitudes de ese Experimento, y en el valor se irá añadiendo al arrayList el número de columna que ese Experimento ocupaba en el fichero .gtcx. Este nuevo mapa se ordena, pero esta vez por los valores de la clave de mayor a menor, de forma que, si hay un experimento que presenta 6 genes coincidentes le corresponderá una posición en el mapa con un número más bajo (se recuerda que las posiciones de los arrays en Java comienzan en el 0) que otro que tenga 2.

A continuación se recuperan en un arrayList nuevo de objetos Experimentos los mismos que se utilizaron para confeccionar el mapa, pero esta vez ordenados, es decir, con el que objeto Experimento que tiene más coincidencias en posiciones más bajas del arrayList. Pero esta ordenación resulta insuficiente para determinar si una determinada firmas es más similar a la del usuario que otra firma del fichero. Por ejemplo, una firma obtenida del fichero .gtcx pueden tener 3 genes coincidentes en los 15 genes más expresados y otra únicamente 2, pero en la firma del usuario esos 3 genes son los que tienen más expresión de los 15 mientras que en la firma del fichero son los que menos expresión de los 15, y sin embargo los 2 genes coincidentes en la segunda firma ser también los de máxima expresión, de manera que se podría decir que la segunda firma es más similar que la primera a pesar de tener menos coincidencias.

Así pues, para determinar cual es el orden de las coincidencias encontradas se utilizó el coeficiente de correlación de Spearman, ρ, que permite medir la correlación entre dos variables aleatorias basándose en

el orden de aparición, y que varía entre -1 y 1. El valor del estadístico se calcula de la siguiente manera [19]:

$$\rho = 1 - \frac{6 \sum D^2}{N(N^2 - 1)}$$

Siendo D la diferencia entre los correspondientes estadísticos y N el número de parejas de datos.

Éste estadístico únicamente podrá ser aplicable si el número de firmas a filtrar es mayor de uno y si el número de coincidencias entre las firmas también es mayor de uno. En esos casos, los objetos Experimentos se añadirán a las últimas posiciones del arrayList. Si se cumplen las condiciones para que se pueda aplicar el estadístico se calculará el coeficiente de Spearman gracias a la librería de java commons.Math3 que tiene una clase, SpearmansCorrelation, para realizar este cálculo. Una vez creado el objeto de esta clase es posible acceder al método **correlation()** que recibe como parámetros un array con las posiciones de los genes en la firma del Experimento y otro con las posiciones de los genes en la firma proporcionada por el usuario. Tras aplicar este método se obtiene el valor de la correlación que se añade como propiedad al objeto Experimento que se está evaluando en ese momento.

De esta manera los objetos Experimento del arrayList de experimentos tienen dos propiedades por las que pueden ser ordenados: coeficiente de correlación de Spearman y el número de genes coincidentes. De forma que la ordenación final del arrayList se lleva a cabo en base a estos dos criterios, siendo prioritaria la correlación de Spearman.

Como paso final en el código del algoritmo se añade el identificador del experimento que será la clave primaria al guardar el objeto en la base de datos.

2.5 Desarrollo de la arquitectura Modelo-Vista-Controlador

El prototipo de aplicación web está desarrollado con la arquitectura Modelo-Vista-Controlador, que es la más habitual para este tipo de aplicaciones. Los datos de la aplicación, la interfaz de usuario, y la lógica de control está separada en tres componentes distintos. [20]

El Modelo contiene una representación de los datos que maneja el sistema, su lógica de negocio, y sus mecanismos de persistencia.

La Vista, o interfaz de usuario, que compone la información que se envía al cliente y los mecanismos interacción con éste.
El Controlador, que actúa como intermediario entre el Modelo y la Vista, gestionando el flujo de información entre ellos y las transformaciones para adaptar los datos a las necesidades de cada uno.

En esta aplicación se han utilizado las librerías de EclipseLink que son una implementación del framework JPA. Por medio de estas librerías se

manejan los datos relacionales de la aplicación creando unas clases de Java, denominadas entidades, que son la representación de la base de datos en el proyecto Java. Cada tabla es una entidad, y cada columna de esa tabla será una propiedad distintas de esa clase. Cada propiedad estará acompañada por una o más anotaciones de Java para indicar cual es su columna correspondiente en la base de datos y si se trata de la clave primaria, @Id, que implica que no pueden contener valores nulos, y las claves foráneas aparecen en las tablas padre como listas de ese tipo de objeto, mientras que en las entidades correspondientes a las tablas hijas esas claves foráneas aparecen definidas como objetos.

Para que estas tablas pudieran ser útiles en la aplicación, fueron convertidas en clases, denominadas entidades, gracias a las librerías de persistencia de EclipseLink. Cada tabla es una entidad y cada columna es una propiedad de esa clase. Las claves primarias, aparecen con la anotación @Id, para indicar que no pueden contener valores nulos, y las claves foráneas aparecen en las tablas padre como listas de ese tipo de objeto, mientras que en las entidades correspondientes a las tablas hijas esas claves foráneas aparecen definidas como objetos (Figura 11).

```
@Id
@GeneratedValue(strategy = GenerationType.IDENTITY)
@Basic(optional = false)
@Column(name = "ID_EXPERIMENTO")
private Integer idExperimento;
@Column(name = "NOMBRE")
private String nombre;
@Column(name = "INDICE_COLUMNA")
private Integer indiceColumna;
@Column(name = "RANKING")
private Integer ranking;
@Column(name = "SPEARMAN")
private Double spearman;
@Column(name = "UP_DOWN")
private Integer upDown;
@Column(name = "NCOINCIDENCIAS")
private Integer ncoincidencias;
@OneToMany(mappedBy = "experimento")
private List<Similitud> similitudList;
@JoinColumn(name = "ID_RESULTADO", referencedColumnName = "ID_RESULTADO")
@ManyToOne
private Resultado resultado;
```

Figura 11: esquema de una entidad de Java, se muestra la de Experimento.

Experimento actúa como clave foránea en la tabla Similitud, por ello aparece la anotación @OneToMany sobre la lista de Similitud, ya que la relación es 1 a n. Mientras que resultado es clave foránea en la tabla Experimento, por lo que la relación es de n a 1, y aparece definida como un objeto de la clase Resultado.

En resumen, en el árbol de ficheros hay siete entidades, una por cada tabla de la base de datos.

Las vistas se tratan de ficheros con extensión xhtml que son páginas JSF por la utilización del framework JavaServer Faces 2.2 para la

creación de las distintas interfaces que va a utilizar el usuario. Para la utilización de esta tecnología se añadieron las librerías JSF al proyecto, así como las implementaciones MyFaces y Facelets y la extensión PrimeFaces.

La aplicación contiene dos plantillas que sirven de base para que las vistas tengan una apariencia similar sin necesidad de repetir código. Éstas se hallan dentro del directorio WEB-INF/plantillas, y son base y baseUsuario. Todas las vistas utilizarán estas plantillas, de hecho, baseUsuario, ya utiliza base como plantilla. En total la aplicación tiene seis vistas principales alojadas en el directorio pages:

- Comparison: es la vista más importante de la aplicación, desde ella se lleva a cabo la introducción de datos para la comparación de las firmas. Tiene el componente “steps” que permite separar en distintas fases la elección de los parámetros y la firma así como la muestra del resultado obtenido. Cada uno de estos pasos es a su vez otro fichero .xhml que se encuentra alojado en WEB-INF/componentes.
- Index: es la vista del menú principal donde se muestran los tres módulos que tiene la aplicación.
- Login: es la vista que da acceso a la aplicación, desde ella el usuario se autentifica.
- Profile: en esta vista el usuario puede cambiar los datos con los que se registró.
- Registrar: si el usuario no está almacenado en la base de datos desde esta vista puede darse de alta como un usuario nuevo.
- Results: es un módulo que muestra un histórico de las comparaciones llevadas a cabo con anterioridad por el usuario.

Los controladores son también clases de Java pero que tienen la anotación @ManagedBean, para hacer referencia a JSF que se puede acceder a las propiedades y métodos de esa clase de java desde la vista. En los controladores se procesa la información introducida por el usuario para luego mostrar los resultados que han sido solicitados por el usuario. En la aplicación hay un total de seis controladores, cada uno asociado a una de las vistas que se ha descrito anteriormente:

- ComparisonController: en él está el código correspondiente al algoritmo comparador. Tras realizar la comparación se almacena en base de datos el resultado obtenido.
- LoginController: busca en la base de datos el usuario introducido y coteja la contraseña con la almacenada, previa encriptación con el algoritmo SHA256.
- PrincipalController: se encarga de redirigir a los módulos correspondientes según el botón pulsado.
- ProfileController: administra las propiedades de la entidad Usuario para actualizar sus datos en base de datos.

- RegistrationController: genera nuevos usuarios y codifica sus contraseñas con el algoritmo SHA256, para después almacenarlos en la base de datos.
- ResultsController: carga los listados correspondientes al usuario para las entidades Resultado, Experimento y Similitud.

La integración de estos tres tipos de elementos permite el funcionamiento de la aplicación realizando un movimiento de informaciones desde la entrada de parámetros y datos del usuario por medio de las vistas que generan la interfaz hasta el almacenamiento en la base de datos por medio del almacenamiento de datos en los objetos de las entidades, los cuales son generados en los controladores asociados a las vistas. Cabe señalar que existen clases fuera de la arquitectura MVC que se utilizan de forma transversal a todos los controladores y contienen métodos indispensables para el correcto funcionamiento de la aplicación.

2.6 Descripción funcionamiento prototipo.

Explicada la arquitectura de la aplicación, así como el algoritmo que funciona de motor para la realización de la comparación de firmas entre el usuario y el contenido del fichero .gtcx, únicamente resta, en este apartado, mostrar el funcionamiento de la aplicación con una consulta representativa.

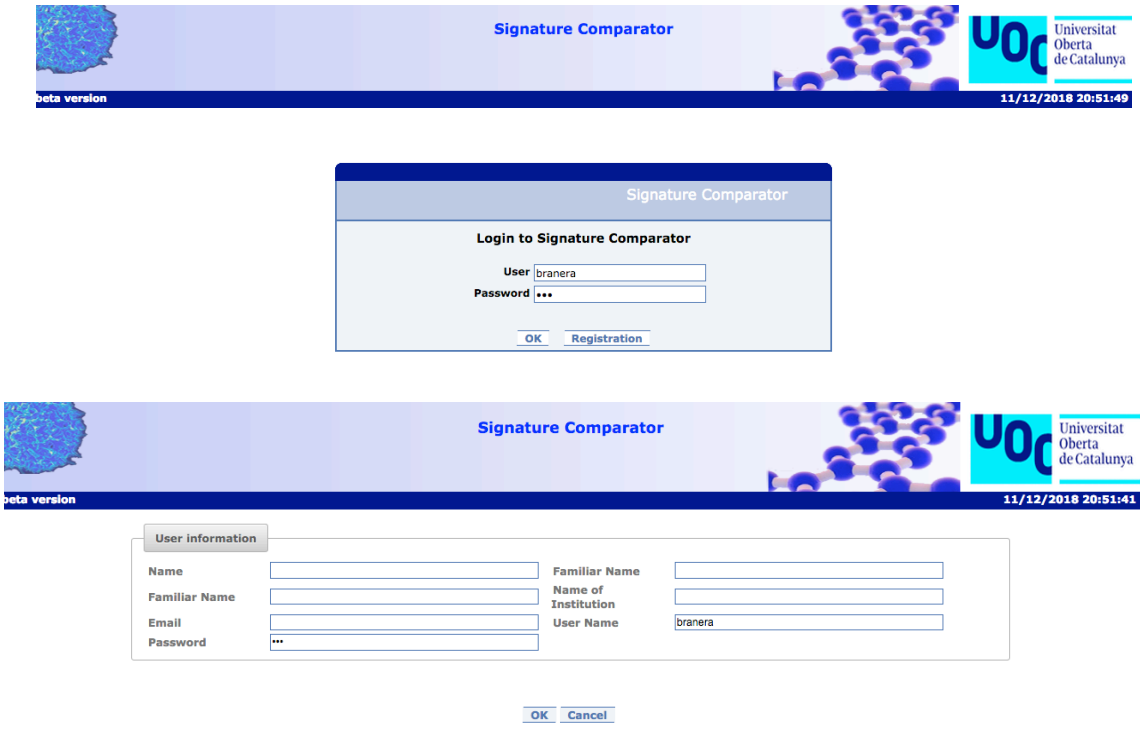


Figura 12. Formularios de acceso y registro

La aplicación está contemplada para que el navegador cargue como vista inicial la de login, por la cual el usuario valida su acceso

comprobando su existencia en la base de datos y la correspondencia con la contraseña, la cual se encuentra codificada por medio del algoritmo SHA256. A través del botón Registration el usuario accede a la vista correspondiente para darse de alta como nuevo usuario (registro) (Figura 12).

Una vez que se ha validado el acceso al usuario, éste accede al menú principal de la aplicación (Figura 13), a partir del cual puede acceder a los tres módulos distintos que tiene la aplicación.

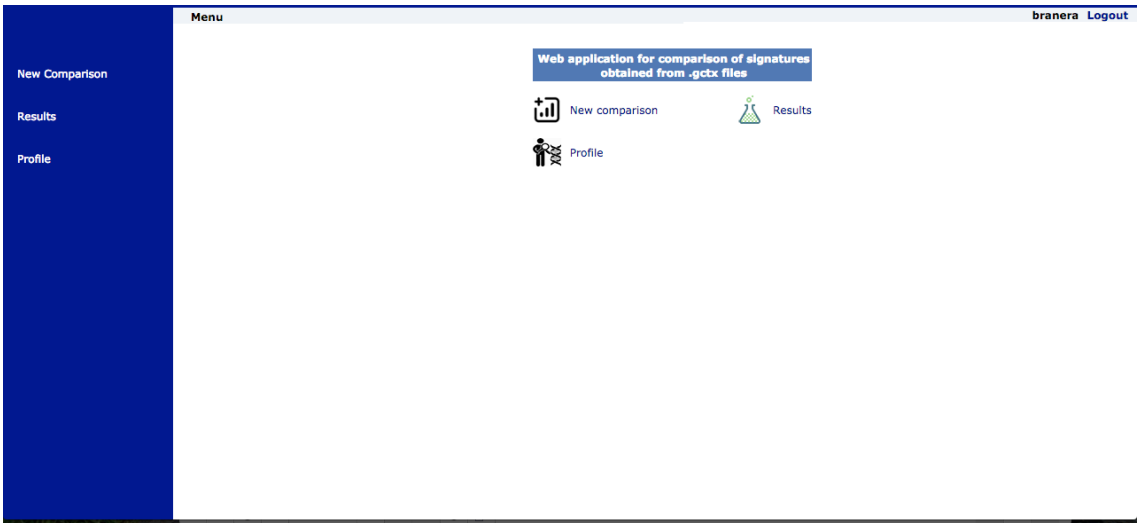


Figura 13. Menú principal al que se accede tras la autentificación.

En el módulo del menú inicial llamado “New Comparison se ideó un diseño tipo “wizard” para que el usuario pudiera introducir la información paso a paso. Al usuario se le va guiando a que introduzca los parámetros necesarios para el análisis comparativo de las firmas sin los cuáles no puede continuar el análisis ya que son obligatorios para el algoritmo descrito anteriormente.

En la figura 14 se puede apreciar un componente de la librería Primefaces, para posibilitar la subida de ficheros al servidor y que estos puedan ser leídos por la aplicación. Es preciso señalar, que en la pantalla inicial se muestra una estructura muy concreta que debe tener el fichero. Éste debe ser un archivo .xls con dos columnas, la primera conteniendo el nombre de los genes y la segunda el valor Z-score resultado de la experimentación realizada por el usuario. La primera fila del fichero sería el encabezado. Cabe destacar que el formulario utilizado en la vista asociada a este módulo debe ser del tipo de codificación “multipart/form-data” que permite la subida de ficheros.

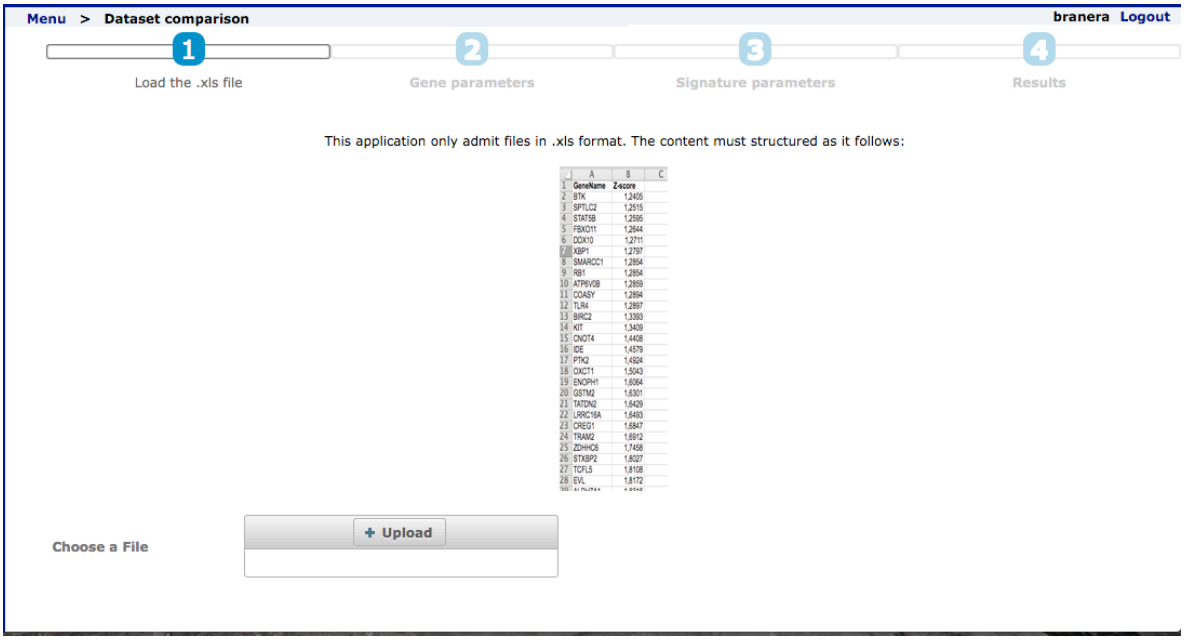


Figura 14. Primer paso del módulo de análisis comparativo de las firmas.

El siguiente paso del módulo, se muestra en la Figura 15, y se corresponde con la introducción de parámetros referentes a la selección de genes que se desean comparar entre el fichero de resultados del usuario y los contenidos en el fichero .gctx, que como ya se describió en la PEC anterior está integrado dentro del árbol de ficheros de la aplicación.

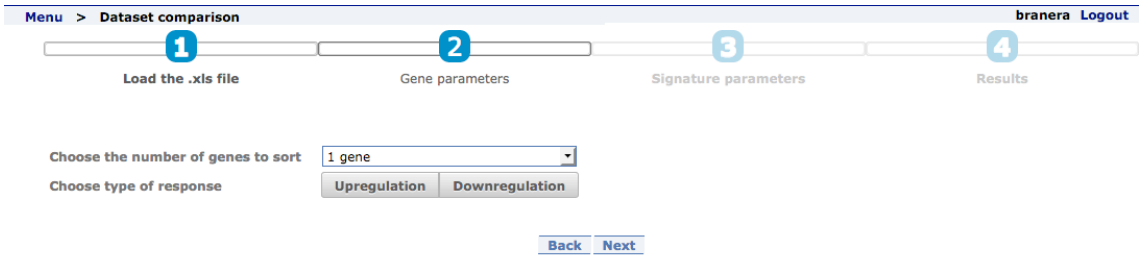


Figura 15. Segundo paso en la interfaz de análisis comparativo de firmas.

En este paso, el usuario debe escoger en un listado desplegable entre 1, 5, 10, 15 y 25 genes que estén o bien sobreexpresados, o bien inhibidos. Este número de genes se seleccionará en ambos ficheros tras ordenar de forma descendente (de mayor expresión a menor) y realizando un

subset con ese número de genes. Por defecto se selecciona 1 gen y si no se escoge el tipo de regulación que se observa no permite avanzar.
El siguiente paso, mostrado en la Figura 16, permite escoger el número de firmas más similares a la proporcionada por el usuario, así como escoger entre los metadatos más informativos, como anteriormente se ha señalado. El número de firmas a escoger se realiza a través de un menú desplegable que ofrece las posibilidades entre 1, 5, 10, 15 y 25 firmas.

Al pasar del paso 3 al 4 se realizaría el análisis comparativo de la firma del usuario y las contenidas en el fichero .gtcx, es decir, se aplicaría el algoritmo descrito anteriormente.

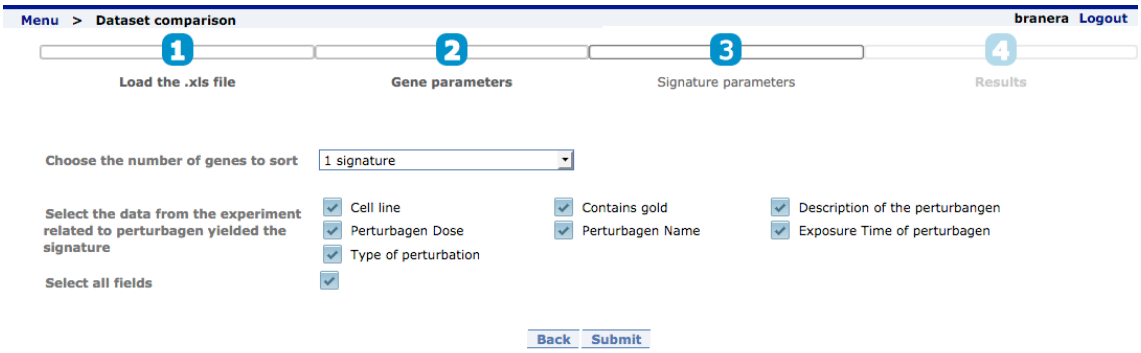


Figura 16. Paso relacionado con la elección de número de firmas más similares a la aportada y de datos que se desean mostrar.

Finalmente, en la Figura 17 se recogen los resultados que se obtendrían tras el análisis, mostrando por un lado una caja de resumen con los parámetros introducidos, así como el usuario.

Estos datos están almacenados en un objeto correspondiente a la entidad Resultado, y que por tanto, al finalizar se guardarán en dicha tabla.

Así mismo se muestra una tabla con las firmas más similares según los parámetros introducidos, mostrando los campos seleccionados en el paso 3.

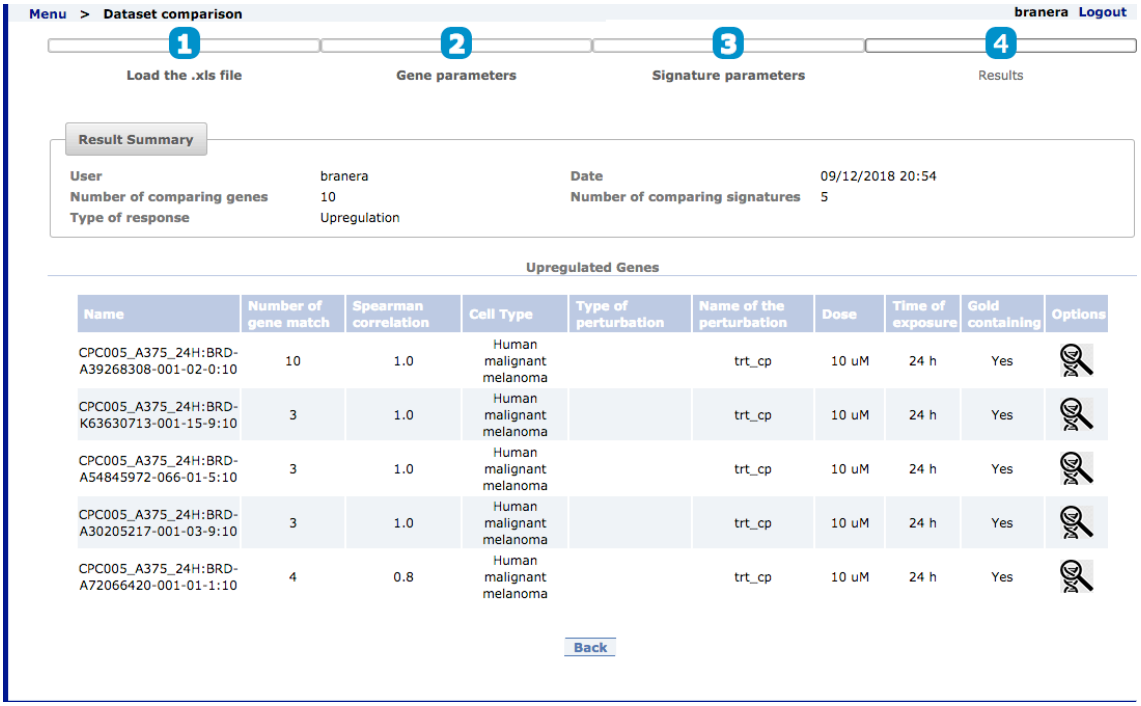


Figura 17. Paso final mostrando los resultados del análisis comparativo.

Para cada experimento se abre una ventana modal, Figura 18, que va a mostrar los detalles correspondientes a los genes localizados en ambas firmas coincidentes entre los parámetros de genes introducidos en el paso 2. Se muestra el nombre del gen, así como el valor del Z-score que tenía la muestra y en la firma seleccionada. Además, como se ha nombrado anteriormente se ha ampliado la funcionalidad de la aplicación, mostrando también las funcionalidades descritas para ese gen.

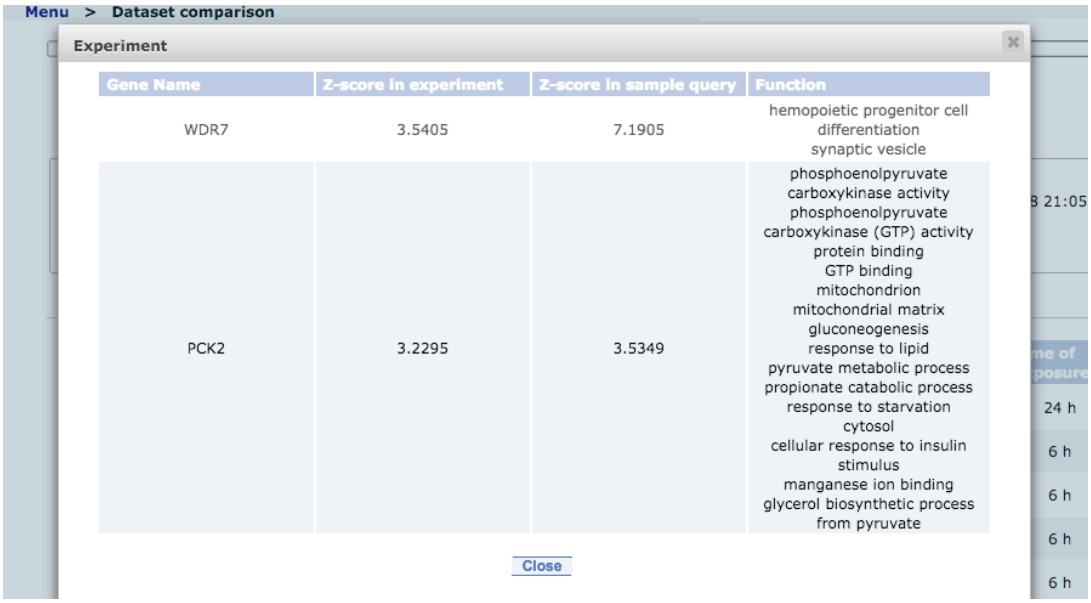


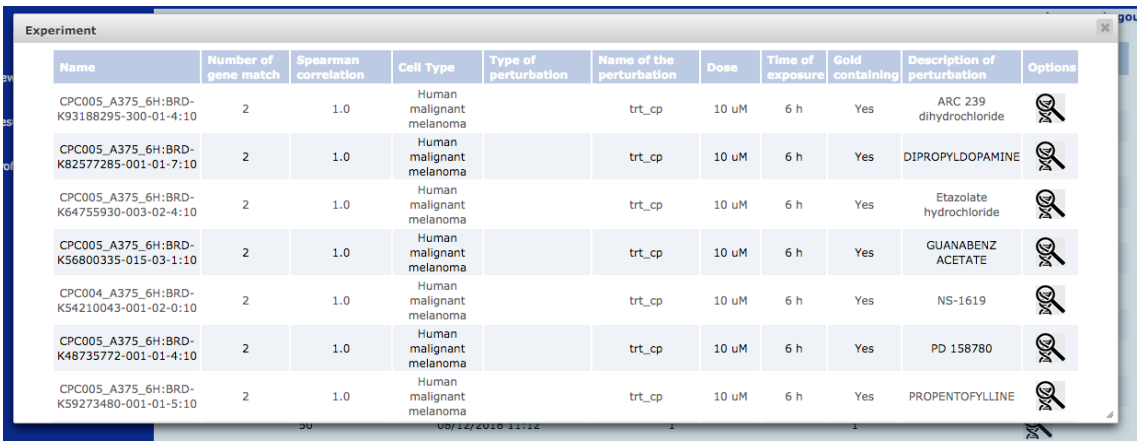
Figura 18. Ventana modal mostrando los genes asociados al experimento seleccionado, así como la funcionalidad descrita en geneontology.

El módulo llamado Results al que se puede acceder desde la barra lateral o el menú principal, muestra un listado que recoge los análisis comparativos previos realizados. El contenido inicial de la vista es un listado de los resultados obtenidos con anterioridad (Figura 19).

El listado muestra el número que se le asignó de forma automática al guardar en la base de datos al registro en la tabla de resultados, la fecha con la hora, y la información relativa sobre el número de genes que se comparaban y el número de firmas donde se realizaba el corte para mostrar al usuario. Se planea añadir una columna más que muestre si se realizó una comparación entre genes sobreexpresados o inhibidos.

Figura 19. Módulo de visualización de histórico de resultados.

La figura 20 muestra el resumen de resultados de las firmas más similares a la firma problema, primero la relación de experimentos-firma mostrando la misma información que el listado mostrado en el módulo de comparación y pulsando sobre el botón de opciones la información relativa sobre los genes con mayor cambio de respuesta en dichos experimentos y coincidentes con la firma problema.



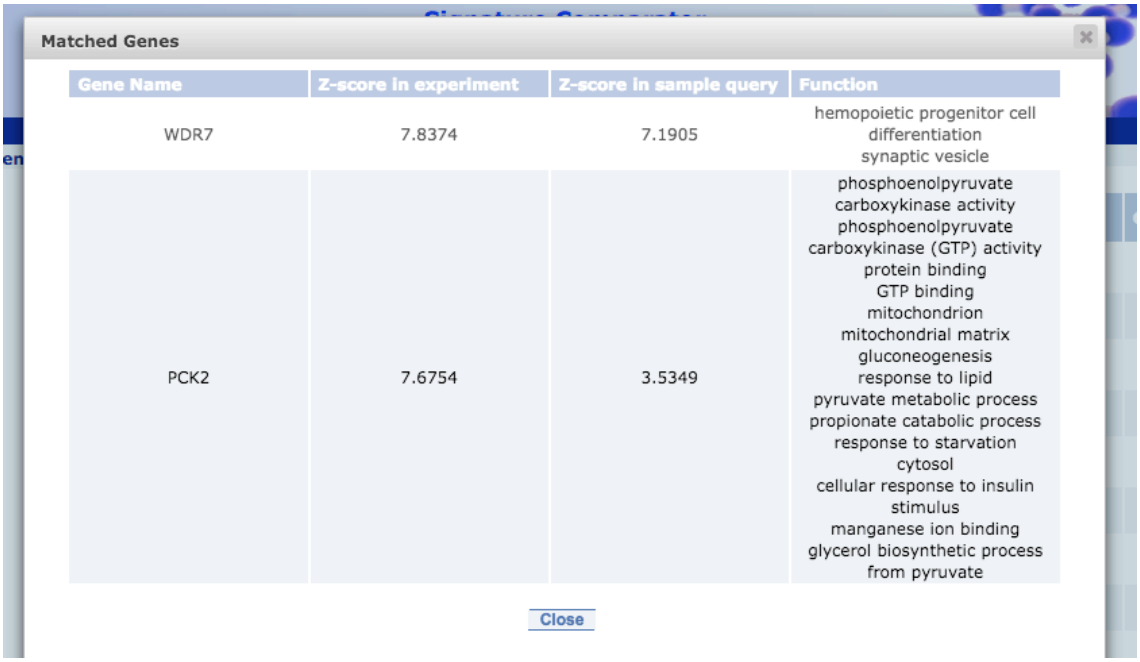


Figura 20. Experimentos y Simililtudes asociadas a un resultado.

Finalmente, el último módulo corresponde al mantenimiento del perfil del usuario, llamado Profile donde el usuario podrá cambiar sus datos de acceso a la aplicación (Figura 21), los cuales quedarán reflejados en la base de datos.

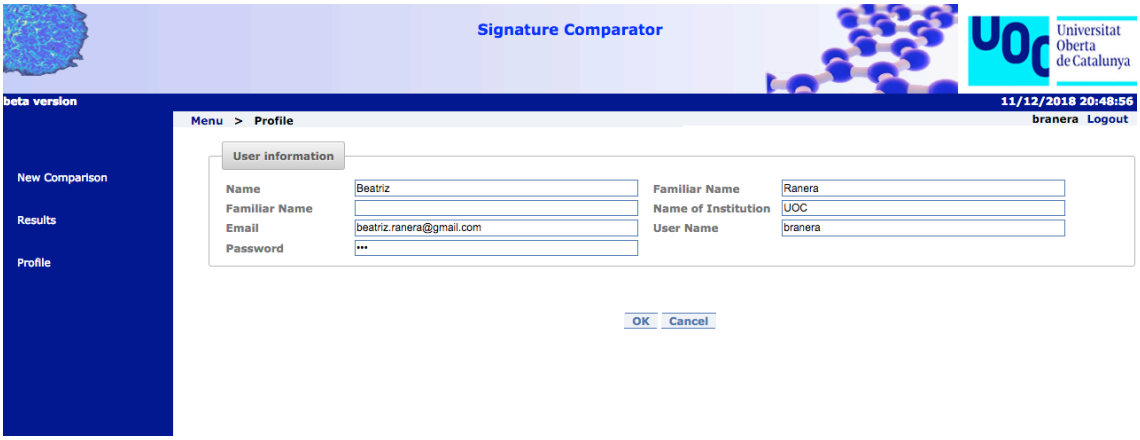


Figura 21. Módulo actualización de datos del perfil del usuario.

2.7 Validación con datos reales

Para validar la funcionalidad de la aplicación se utilizaron tres datasets distintos al original utilizado durante el desarrollo. Recordar que la firma query empleada se trataba de una firma aleatoria escogida de las que había en el fichero que se toma para referencia para consultar en él las firmas.

Los tres datasets se obtuvieron del análisis de microarrays del artículo de Shi y colaboradores [22] en el que se examinan los cambios en la expresión de genes de muestras obtenidas de las células

mononucleares de sangre periférica. Los ficheros .CEL se obtuvieron de la serie GSE49515 de GEO y su expresión diferencial fue determinada mediante los paquetes affy, Biobase, oligo, limma, annotate y hgu133plus2.db de R para el análisis de microarrays, según el protocolo establecido para la realización de este tipo de análisis de la asignatura Análisis de datos ómicos.

Estas muestras se extrajeron de 4 tipos distintos de pacientes humanos: normales (10 muestras), con cáncer de páncreas (3 muestras), con cáncer hepatocelular (10 muestras) y cáncer gástrico (3 muestras). Por tanto, se podrá comparar la expresión diferencial los genes alterados en tres tipos de cáncer distinto y compararlos con las firmas de los experimentos del ensayo L1000. De esta manera se podrá determinar si alguno de los genes expresados diferencialmente se encuentrá entre los más afectados por algún perturbagen, aportando así, información de posibles tratamientos para los tumores que sirve de query.

Ejemplo carcinoma pancreático:

Se obtuvieron 67 genes con expresión diferencial aplicando el protocolo anteriormente señalado. Aplicando los filtros de comparación: Upregulation – 25 genes más expresados – 25 firmas más parecidas se obtuvieron los resultados que se muestran en la figura 22.

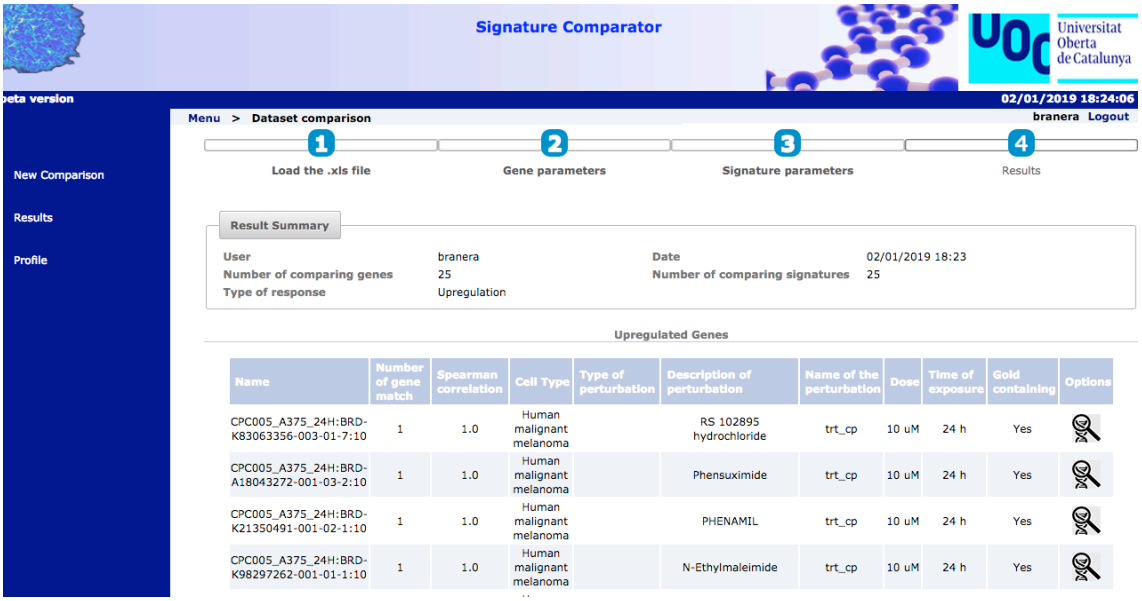


Figura 22. Resultado del análisis de las muestras de carcinoma pancreático mostrando las condiciones experimentales más similares.

En total se obtuvieron una coincidencia en un total de 25 firmas que contenían en sus 25 genes más expresados alguno de los 25 más expresados en la firma query. Estas condiciones fueron encontradas para la misma línea celular, la del melanoma humano maligno a 10 uM de concentración del perturbagen y con una exposición de 6h o 24h. Los perturbagenes, y por tanto sustancias suceptibles de ser ensayadas

como tratamiento antitumoral en páncreas podrían ser: RS102895 hidroclorhídro, fensuximida, PHENAMIL, N-etilmaleimida, MARMESIN, ácido iocetámico, maleato hidroxitacrino (R,S), gelsamina, difemanil metilsulfato, daidzen, DIPRPYLDOPAMINA, benzamil hidroclorhírdro, amilsulprido, Amino-1,8-naphtlalamida[4-amino1-8-naphtalamida], ATPA, purvalanol B, Paxiline, propentofylina, nocodazole, nitrofurazone, isoxsuprine hidroclorhidro, fluorextine hidroclorhídro, dihidroergocristina, CGP 54626 hidroclorhídro y ácido asiático.

En todos los casos, el gen que aparecía sobreexpresado se trataba del gen IL13RA1 una proteína de membrana implicada en rutas de señalización, unión de proteínas, receptor…
(Figura 23).

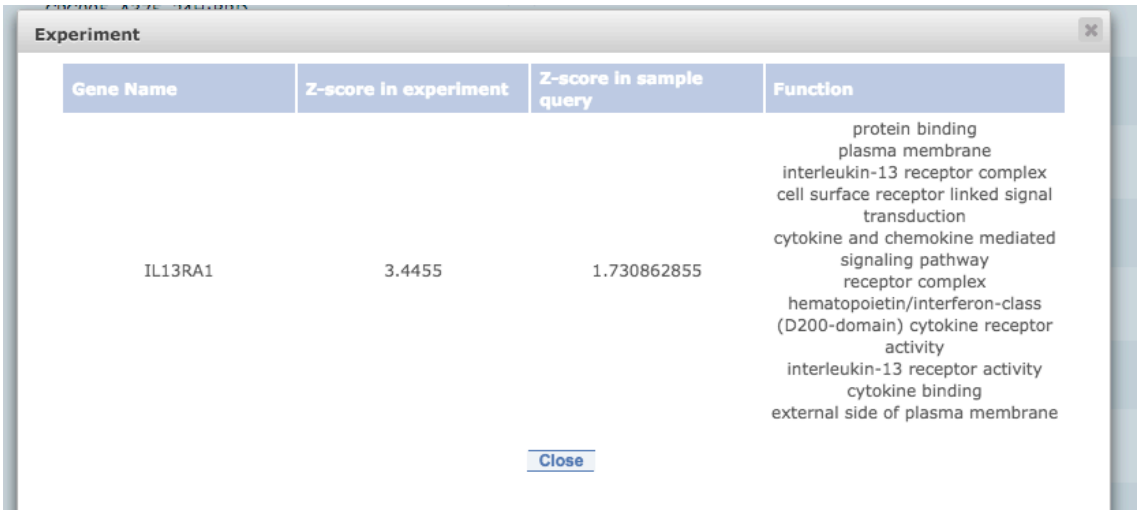


Figura 23. Gen expresado diferencialmente en las firmas.

Ejemplo carcinoma hepatocelular:
Se obtuvieron 32 genes expresados diferencialmente y los filtros para realizar las comparaciones de las firmas fueron Upregulation – 15 genes más expresados – 25 firmas más parecidas. Los resultados se muestran en la figura 24.

Se obtuvieron 8 condiciones experimentales que mostraban un gen en común con el de la firma aportada. De estas 8, 3 tenían como línea celular tumoral al melanoma humano maligno con condiciones de exposición de 6 y 24 horas a 10 uM de concentración para los perturbagenes: dihidroergocristina, U-54494A, y CW 5074, estando afectado, para todos los casos, el gen: NIT1. Para las otras 5 firmas, correspondientes a la línea celular de epitelio humana inmortalizada los perturbagenes fueron AZ 10417808, ciglitazona, CP93129 dihidroclorhídro, dihidrosamidina y doxepina hidroclorhídro expuestas durante 6 horas a una concretación de 10 uM. El gen en común también resultó ser el NIT1.

Upregulated Genes

Name	Number of gene match	Spearman correlation	Cell Type	Type of perturbation	Description of perturbation	Name of the perturbation	Dose	Time of exposure	Gold containing	Options
CPC005_A375_6H:BRD-A65076780-001-01-3:10	1		Human malignant melanoma		DIHYDROERGOCRISTINE	trt_cp	10 uM	6 h	Yes	
CPC005_A375_6H:BRD-K20995441-001-01-7:10	1		Human malignant melanoma		U-54494A	trt_cp	10 uM	6 h	Yes	
CPC005_A375_24H:BRD-K39520573-001-01-2:10	1		Human malignant melanoma		GW 5074	trt_cp	10 uM	24 h	Yes	
CPC002_HA1E_6H:BRD-K36258877-001-01-5:10	1		Human kidney epithelial immortalized cell line		AZ 10417808	trt_cp	10 uM	6 h	Yes	
CPC002_HA1E_6H:BRD-A93000692-001-02-4:10	1		Human kidney epithelial immortalized cell line		CIGLITAZONE	trt_cp	10 uM	6 h	Yes	
CPC001_HA1E_6H:BRD-K81876028-300-01-2:10	1		Human kidney epithelial immortalized cell line		CP 93129 dihydrochloride	trt_cp	10 uM	6 h	Yes	
CPC004_HA1E_6H:BRD-K63945320-001-03-2:10	1		Human kidney epithelial immortalized cell line		DIHYDROSAMIDIN	trt_cp	10 uM	6 h	Yes	
CPC004_HA1E_6H:BRD-K36616567-003-01-5:10	1		Human kidney epithelial immortalized cell line		Doxepin hydrochloride	trt_cp	10 uM	6 h	Yes	

Figura 24. Resultado del análisis de las muestras de carcinoma hepatocelular mostrando las condiciones experimentales más similares.

El gen NIT1, está ubicado en el núcleo y mitocondria y sus actividades están relacionadas con el metabolismo del nitrógeno. (Figura 25)

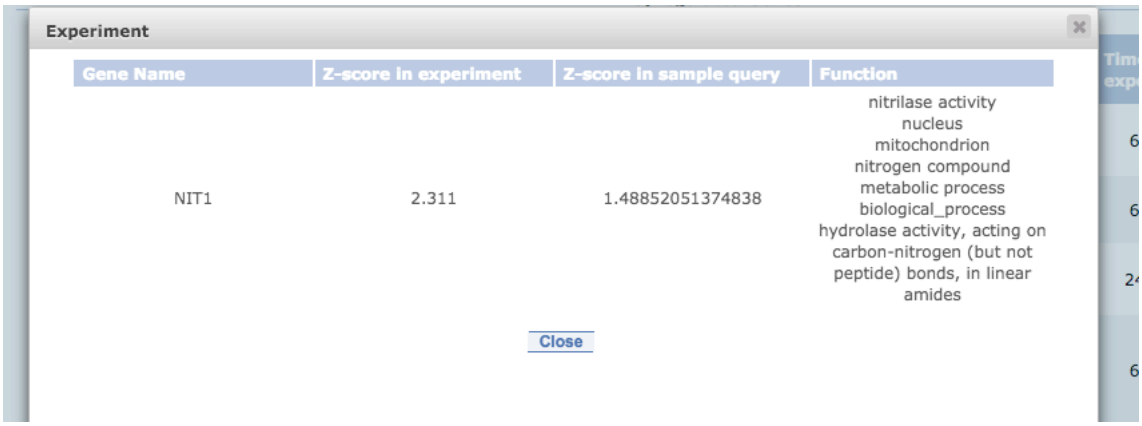


Figura 25. Gen expresado diferencialmente en las firmas.

De esta manera se podría pensar que los perturbagenes anteriormente descritos podrían pasar a considerarse como potenciales antitumorales para el tratamiento del carcinoma hepatocelular, del mismo modo que han demostrado ya tener esta actividad en las líneas celulares de riñón y melanoma maligno.

Ejemplo carcinoma gástrico:
Sin embargo en el caso del carcinoma gástrico no se encontró ningún gen coincidente entre los más/menos expresados dentro del fichero .gtcx, lo que demuestra también la limitación del prototipo al realizar la comparación entre un límite máximo de 25 genes y 25 firmas, así como de utilizar el fichero test_data que únicamente contiene los 1000 primeros experimentos realizados en 3 líneas tumorales únicamente.

3. Conclusiones

- El prototipo desarrollado en este proyecto ofrece una interfaz de usuario sencilla y ergonómica para que un usuario investigador sin conocimientos previos de bioinformática pueda comparar los resultados de diferencia de expresión génica con experimentos llevados a cabo por el consorcio LINCS dentro del proyecto L1000. Para poder lograr el objetivo de la creación del prototipo se ha tenido que realizar un análisis exhaustivo de los trabajos realizados por el consorcio LINCS (objetivos, desarrollos, datos, resultados, plataformas…) para poder comprender las posibilidades que ofrece el trabajo llevado hasta la fecha.

  Del mismo modo la realización de este prototipo también me ha permitido descubrir los trabajos realizados con el grupo HDF. El trabajo llevado a cabo por este grupo creando ficheros que faciliten el desarrollo y la labor de investigadores para que a través del manejo de ficheros .gctx se permita la consulta de datos dentro de un gran volumen de ellos de una manera rápida y eficiente por medio de sus paquetes CMap.

  Además, a la hora de crear el algoritmo comparador para ofrecer resultados cualitativos en la aplicación sobre la comparación de firmas se llevó a cabo un estudio en profundidad de los algoritmos bioinformáticos utilizados para el cálculo realizado por los desarrolladores de los paquetes CMap. Esto a su vez implicó estudiar el Gene Set Enrichment Analysis desarrollado por el Broad Institute, y en el que se basan actualmente toda las comparaciones de grupos de genes.

  A parte de los conocimientos relacionados con el campo de la bioinformática propiamente dichos, la consecución de este proyecto también ha requerido de la utilización de manuales y documentación para la creación de una base de datos relacional MySQL y el funcionamiento de una aplicación web con el lenguaje Java: configuración de ficheros, distribución de elementos en directorios, implementación de frameworks, coordinación de librerías…

- El trabajo estaba ideado originalmente para poder leer cualquier tipo de fichero que tuviera un extensión .gctx. Sin embargo las excepciones obtenidas a la hora de cargar los datos en el objeto de la clase Dataset dentro del GctxReader.java truncaron esta idea debido a que se trataban de excepciones más relacionadas con la programación en Java que el verdadero propósito de generar una aplicación que pudiera comparar un grupo de genes aportado por el usuario con las firmas contenidas en el fichero. Una dedicación más extensa a buscar la forma de subsanar estas excepciones podría haber afectado al desarrollo del prototipo y haber alterado de forma sustancial el seguimiento de la planificación establecido.

Por otra parte, el trabajo ofrece una comparación entre firmas, aunque únicamente se pensó para destacar los genes más expresados entre lo aportado por el usuario y lo contenido en el fichero. Sin embargo, reorientar la funcionalidad de la aplicación a la posibilidad de mostrar las firmas más similares con información orientativa sobre la función de esos genes y las perturbaciones que las provocaron se constituyó como un objetivo primordial. Sin embargo, la complejidad que suponía aplicar el algoritmo de cálculo del Connectivity Score y la posible ralentización del prototipo al implementarlo hizo que se descartara la idea de presentar al usuario un resultado cuantitativo sino uno más cualitativo que le sirviera de orientación.

- El objetivo original de crear una aplicación web que permitiera comparar un conjunto de genes obtenido por un usuario investigador con los ensayos recogidos en un fichero .gctx ha sido cumplido. Es más, se amplió la idea de original de únicamente mostrar los genes más o menos expresados, para mostrar las firmas más similares en función del número de genes que tuvieran en común la sobreexpresión o la inhibición. Se siguió la planificación prevista, aunque para ello se tuvieron que cambiar de orden algunas de las tareas a cómo se pensaron inicialmente para poder optimizar el tiempo de trabajo, como por ejemplo, crear una base de datos inicial para poder importar las entidades al proyecto, y así utilizarlas para almacenar información en el algoritmo comparador.

- Línea de trabajo futuro: sacar un mayor rendimiento a la información contenida en el fichero, creando, por ejemplo, un módulo en el que se introduzca el perturbagen y la aplicación devuelva aquellas condiciones experimentales, así como los genes, en las que el perturbagen ha sido utilizado.

Además, la aplicación debería ser más compatible a la hora de subir ficheros, de modo que pudieran subirse archivos con otras extensiones además de la .xls.

4. Glosario

CMap: Connectivity Map.
CMapJ: Connevitivity Map en Java.
CSS: Cascading Style Sheet.
DAO: Data Access Object.
ES: Enrichment Score.
GEO: Gene Expression Omnibus.
GSEA: Gene Set Enrichment Analysis.
HDF: Hierarchical Data Format.
HDF5: Versión 5 del Hierarchical Data Format.
IDE: Integrated Development Environment
J JPA: Java Persistence API.
JSF: Java Server Faces.
DK: Java Development Kit.
LINCS: Library of Integrated Network-Based Cellular Signatures.
Método constructor: conjunto de órdenes en código por las cuales se crea un objeto de una clase. A partir de este objeto se pueden acceder a los atributos y método que la clase contiene.
MVC: arquitectura modelo-vista-controlador.
siRNA: Small interference Ribonucleic Acid.
Perturbagen: Compuesto de carácter biológico o químico que se utiliza en el laboratorio para tratar células y medir la respuesta biológica resultante.
Servlet: clase en el lenguaje de programación Java, utilizada para generar páginas web de forma dinámica a partir de los parámetros de la petición que envíe el navegador web.

5. Bibliografía

[1] Amesty, A., Estevez-Braun, A., & Hortelano, S. (2018). Metal complexes of natural product like-compounds with antitumoral activity. Anti-cancer agents in medicinal chemistry. Epub ahead of print

[2] Ramos, M. C., Boulaiz, H., Griñan-Lison, C., Marchal, J. A., & Vicente, F. (2017). What's new in treatment of pancreatic cancer: a patent review (2010–2017). Expert opinion on therapeutic patents, 27(11), 1251-1266.

[3] Mouhid, L., Corzo-Martínez, M., Torres, C., Vázquez, L., Reglero, G., Fornari, T., & Ramírez de Molina, A. (2017). Improving In Vivo Efficacy of Bioactive Molecules: An Overview of Potentially Antitumor Phytochemicals and Currently Available Lipid-Based Delivery Systems. Journal of oncology, 2017.

[4] C Sobrinho, J., Simoes-Silva, R., J Holanda, R., Alfonso, J., F Gomez, A., B Zanchi, F., ... & M Soares, A. (2016). Antitumoral potential of snake venom phospholipases A2 and synthetic peptides. Current pharmaceutical biotechnology, 17(14), 1201-1212.

[5] Massi, P., Valenti, M., Solinas, M., & Parolaro, D. (2010). Molecular mechanisms involved in the antitumor activity of cannabinoids on gliomas: role for oxidative stress. Cancers, 2(2), 1013-1026.

[6] Hazim, A., & Prasad, V. (2018). A pooled analysis of published, basket trials in cancer medicine. European Journal of Cancer, 101, 244-250. Epub 2018 Aug 7

[7] Duan, Q., Reid, S. P., Clark, N. R., Wang, Z., Fernandez, N. F., Rouillard, A. D., ... & Niepel, M. (2016). L1000cds2: Lincs l1000 characteristic direction signatures search engine. NPJ Syst Biol Appl. 2016; 2: 16015..

[8] http://www.lincsproject.org/ Visitado por primera vez: 29/09/2018

[9] Subramanian, A., Narayan, R., Corsello, S. M., Peck, D. D., Natoli, T. E., Lu, X., ... & Lahr, D. L. (2017). A next generation connectivity map: L1000 platform and the first 1,000,000 profiles. Cell, 171(6), 1437-1452.

[10] Enache, O. M., Lahr, D. L., Natoli, T. E., Litichevskiy, L., Wadden, D., Flynn, C., ... & Subramanian, A. (2018). The GCTx format and cmap {Py, R, M, J} packages: resources for optimized storage and integrated traversal of annotated dense matrices. Bioinformatics.

[11 Duan, Q., Flynn, C., Niepel, M., Hafner, M., Muhlich, J. L., Fernandez, N. F., ... & Sorger, P. K. (2014). LINCS Canvas Browser: interactive web app to query, browse and interrogate LINCS L1000 gene expression signatures. Nucleic acids research, 42(W1), W449-W460.

[12] https://github.com/cmap/cmapJ/. Visitado por primera vez 1/10/2018

[13] https://www.hdfgroup.org/ Visitado por primera vez 14/10/2018

[14] https://clue.io/connectopedia/data_levels Visitado por primera vez 30/09/2018

[15] Capitulo 3. Lantz, B. (2013). Machine learning with R. Packt Publishing Ltd.

[16] Enache, O. M., Lahr, D. L., Natoli, T. E., Litichevskiy, L., Wadden, D., Flynn, C., ... & Subramanian, A. (2017). The GCTx format and cmap

{Py, R, M} packages: resources for the optimized storage and integrated traversal of dense matrices of data and annotations. bioRxiv, 227041.
[17] Subramanian, A., Narayan, R., Corsello, S. M., Peck, D. D., Natoli, T. E., Lu, X., ... & Lahr, D. L. (2017). A next generation connectivity map: L1000 platform and the first 1,000,000 profiles. Cell, 171(6), 1437-1452.
[18] Subramanian, A., Tamayo, P., Mootha, V. K., Mukherjee, S., Ebert, B. L., Gillette, M. A., ... & Mesirov, J. P. (2005). Gene set enrichment analysis: a knowledge-based approach for interpreting genome-wide expression profiles. Proceedings of the National Academy of Sciences, 102(43), 15545-15550.
[19] Crawley, M. J. (2012). The R book. John Wiley & Sons.
[20] https://si.ua.es/es/documentacion/asp-net-mvc-3/1-dia/modelo-vista-controlador-mvc.html Visitado por primera vez 15/11/2018
[21] http://www.jtech.ua.es/j2ee/publico/jsf-2012-13/sesion01-apuntes.html Visitado por primera vez 14/11/2018.
[22] Shi M, Chen MS, Sekar K, Tan CK et al. A blood-based three-gene signature for the non-invasive detection of early human hepatocellular carcinoma. Eur J Cancer 2014 Mar;50(5):928-36.

6. Anexos

6.1 ANEXO 1

LISTADO DE LIBRERÍAS UTILIZADAS EN EL PROYECTO

- activation.
- commons-beanutils-1.7.0
- commos-codec.1.5
- commons-collections-3.2.1
- commons-digester-1-8
- commons-el-1.0
- commons-fileupload-1.2.1
- commons-io-1.3.2
- commons-lang-2.4
- commons-logging-1.1.1
- commons-math3-3.6.1
- commons-validator-1.3.1
- org-netbeans-modules-java
- eclipselink-jpa
- eclipselink-jpa
- el-api
- el-ri-1.0
- guava-r09
- javax.persistence_1.0.0
- jhdf (específica de CMapJ)
- jhdf5 (específica de CMapJ)
- jhdf5obj (específica de CMapJ)
- jhdfobj (específica de CMapJ)
- njhdf (específica de CMapJ)
- javaee-doc-api
- javax.faces
- log4j-1.2.15
- mysql-connector-java-8.0.13
- poi-4.0.0
- primefaces-6.2
- tomahawk 1.1.14
- xml-apis.1.0b2
- xmlParserAPIs-2.0.2

6.2 ANEXO 2

SQL disparado para la creación de la base de datos MySQL

```
drop table if exists ANOTACION;

drop table if exists CELL_TYPES;

drop table if exists EXPERIMENTO;

drop table if exists GENE_ONTOLOGY;

drop table if exists RESULTADO;

drop table if exists SIMILITUD;

drop table if exists USUARIO;

/*==============================================================
===*/
/* Table: ANOTACION                                             */
/*==============================================================
===*/
create table ANOTACION
(
   ID_ANOTACION         int not null,
   ID_GO                varchar(20),
   DATABASENAME         varchar(100),
   ID_DATABASE          varchar(100),
   SYMBOL               varchar(100),
   QUALIFIER            varchar(100),
   REF_DB               varchar(100),
   EVIDENCE_CODE        varchar(100),
   ASPECT               varchar(100),
   OBJECT_NAME          varchar(255),
   SYNOMIN              varchar(255),
   TYPE_OBJECT          varchar(25),
   TAXON                varchar(50),
   ASSIGNED_BY          varchar(100),
   primary key (ID_ANOTACION)
);

/*==============================================================
===*/
/* Table: CELL_TYPES                                            */
/*==============================================================
===*/
create table CELL_TYPES
(
   ID_CELL              varchar(30) not null,
   DESCRIPCION          varchar(100),
```

```
   primary key (ID_CELL)
);

/*==============================================================
===*/
/* Table: EXPERIMENTO                                           */
/*==============================================================
===*/
create table EXPERIMENTO
(
   ID_EXPERIMENTO       int not null auto_increment comment 'PK
autoincremental del experimento',
   ID_RESULTADO         int comment 'FK del resultado donde se integrará.',
   ID_CELL              varchar(30),
   NOMBRE               varchar(100) comment 'Id extraído de los metadatos',
   INDICE_COLUMNA       numeric(10) comment 'Posición de columna que
ocupa en el dataset.',
   RANKING              numeric(3) comment 'Cómo queda situado el experimento
en función de las similitudes que presente',
   UP_DOWN              numeric(1) comment '0: Upregulated; 1: Downregulated',
   SPEARMAN             numeric(4,2),
   NSIMILITUDES         numeric(3),
   GOLD                 numeric(1),
   DESCRIPERT           varchar(100),
   DOSE                 varchar(50),
   NAMEPERT             varchar(50),
   TYPEPERT             varchar(30),
   EXPTIME              varchar(20),
   NCOINCIDENCIAS       numeric(3),
   primary key (ID_EXPERIMENTO)
);

/*==============================================================
===*/
/* Table: GENE_ONTOLOGY                                         */
/*==============================================================
===*/
create table GENE_ONTOLOGY
(
   ID_GO                varchar(20) not null,
   DESCRIPCION          varchar(100),
   primary key (ID_GO)
);

/*==============================================================
===*/
/* Table: RESULTADO                                             */
/*==============================================================
===*/
create table RESULTADO
```

```
(
   ID_RESULTADO         int not null auto_increment comment 'PK
autoincremental',
   ID_USUARIO           int comment 'FK del usuario Conectado',
   FECHA                datetime comment 'Fecha en la que se generó',
   SIGNATUREFILTER      numeric(2),
   GENEFILTER           numeric(2),
   UP_DOWN              numeric(1),
   primary key (ID_RESULTADO)
);

/*==============================================================
===*/
/* Table: SIMILITUD                                             */
/*==============================================================
===*/
create table SIMILITUD
(
   ID_SIMILITUD         int not null auto_increment comment 'PK
autoincremental',
   ID_EXPERIMENTO       int comment 'FK del experimento al que pertenece',
   INDICE_SIG           numeric(10) comment 'Posición que ocupa en el Array de
la firma el gen',
   INDICE_SAMP          numeric(10) comment 'Posición que ocupa en el Array
de la muestra el gen',
   ZSCORE_SIG           numeric(8,4) comment 'Valor Zscore del gen en la
firma',
   ZSCORE_SAMP          numeric(8,4) comment 'Valor Zscore del gen en la
muestra',
   primary key (ID_SIMILITUD)
);

/*==============================================================
===*/
/* Table: USUARIO                                               */
/*==============================================================
===*/
create table USUARIO
(
   ID_USUARIO           int not null auto_increment comment 'Autoincremental de
Usuario',
   NOMBRE               varchar(100) comment 'Nombre y Apellidos del Usuario',
   PASSWORD             varchar(100) comment 'Contraseña',
   EMAIL                varchar(100) comment 'Email',
   INSTITUCION          varchar(100) comment 'Nombre de la Institución a la que
pertenece',
   USER_NAME            varchar(20) comment 'Nombre de Usuario',
   primary key (ID_USUARIO)
);
```

```
alter table ANOTACION add constraint FK_ANOTACION_GENEONTOLOGY
foreign key (ID_GO)
    references GENE_ONTOLOGY (ID_GO) on delete restrict on update
restrict;

alter table EXPERIMENTO add constraint FK_EXPERIMENTO_CELLTYPE
foreign key (ID_CELL)
    references CELL_TYPES (ID_CELL) on delete restrict on update restrict;

alter table EXPERIMENTO add constraint FK_EXPERIMENTO_RESULTADO
foreign key (ID_RESULTADO)
    references RESULTADO (ID_RESULTADO) on delete cascade on update
restrict;

alter table RESULTADO add constraint FK_RESULTADO_USUARIOS foreign
key (ID_USUARIO)
    references USUARIO (ID_USUARIO) on delete restrict on update restrict;

alter table SIMILITUD add constraint FK_SIMILITUD_EXPERIMENTO foreign
key (ID_EXPERIMENTO)
    references EXPERIMENTO (ID_EXPERIMENTO) on delete restrict on
update restrict;
```

6.3 ANEXO 3

ALGORITMO PARA COMPARAR Y OBTENER LAS FIRMAS MÁS SIMILARES

```
Map<String,Double> sampleUp = Maps.newHashMap();
      Map<String,Double> sampleDown = Maps.newHashMap();

      try {
      reader                                    =                                    new
GctxReader(JsfUtil.getRealPath("/gctx/testdata_n1000x978.gctx"));
     //reader                                    =                                    new
GctxReader("/Volumes/SeagateBac/GSE70138_Broad_LINCS_Level5_COMP
Z_n118050x12328_2017-03-06.gctx");
      // read the full dataset
     dataset = reader.read();
     sampleSignature = MapUtil.sortByValue2(sampleSignature);

     //Ordenación de los genes upregulados o downregulados según elección
en la muestra.

     //Genes upregulated
     if (typesRegulation.equals("1")) {
         sampleUp              =              MapUtil.subMapaUp(sampleSignature,
resultado.getGenefilter());
     } else { //Genes Downregulated
        sampleDown            =            MapUtil.subMapaDown(sampleSignature,
resultado.getGenefilter());
     }


      //Algoritmo para buscar los genes match entre los elegidos por el número
marcado por geneSelected.
      for (int i=0; i<dataset.getColumnCount(); i++) {
         exp = null;
         sim = null;
         List<Similitud> similitudesInExp = Lists.newArrayList();

         //Se introduce en un mapa las firmas encontradas y se ordenan, de
forma individual.
         for (int j=0; j<dataset.getRowCount(); j++) {

signatureDataSet.put((String)dataset.getRowMetadata().getValue(j,"pr_gene_s
ymbol"),new Double(Float.toString(dataset.getValue(j, i))));
         }
         signatureDataSet = MapUtil.sortByValue2(signatureDataSet);

         Map<String,Double> signatureDataSetUp = Maps.newHashMap();
         Map<String,Double> signatureDataSetDown = Maps.newHashMap();
         int coincidenciasUp = 0;
         int coincidenciasDown = 0;
```

```
        //Se hace un subset de los dos mapas y se determinan el número de
coincidencias existentes en la up o downregulation.
        //Genes upregulated
        if (typesRegulation.equals("1")) {
            signatureDataSetUp = MapUtil.subMapaUp(signatureDataSet,
resultado.getGenefilter());
            for (Map.Entry<String, Double> entry : sampleUp.entrySet()) {
                if (signatureDataSetUp.containsKey(entry.getKey())) {
                    coincidenciasUp++;
                    //nExperimento++;
                }

            }
        } else { //Genes Downregulated
            signatureDataSetDown = MapUtil.subMapaDown(signatureDataSet,
resultado.getGenefilter());
            for (Map.Entry<String, Double> entry : sampleDown.entrySet()) {
                if (signatureDataSetDown.containsKey(entry.getKey())) {
                    coincidenciasDown++;
                    //nExperimento++;
                }
            }
        }


        //En el caso que se hayan determinado que hay coincidencias en
upregulation se guardan los datos
        //de los experimentos donde haya coincidencia y cuáles son.
        if (coincidenciasUp > 0) {
            similitudesInExp = Lists.newArrayList();
            exp = new Experimento();
            exp.setResultado(resultado);
            exp.setUpDown(1);
            exp.setNombre((String)dataset.getColumnMetadata().getValue(i,
"id"));
            exp.setIndiceColumna(i);
            if (experimentData.contains("pert_desc")) {
                exp.setDescripert((String)dataset.getColumnMetadata().getValue(i,
"pert_desc"));
            }
            if (experimentData.contains("cell_id")) {
                String cell_id = (String)dataset.getColumnMetadata().getValue(i,
"cell_id");
                exp.setCell(JpaUtil.getDAO().find(CellTypes.class, cell_id));
            }
            if (experimentData.contains("is_gold")) {
                if (((String)dataset.getColumnMetadata().getValue(i,
"is_gold")).equals("True")) {
```

```
                exp.setGold(1);
            } else {
                exp.setGold(0);
            }

        }
        if (experimentData.contains("pert_idose")) {
            exp.setDose((String)dataset.getColumnMetadata().getValue(i,
"pert_idose"));
        }
        if (experimentData.contains("pert_iname")) {
            exp.setNamepert((String)dataset.getColumnMetadata().getValue(i,
"pert_iname"));
        }
        if (experimentData.contains("pert_type")) {
            exp.setNamepert((String)dataset.getColumnMetadata().getValue(i,
"pert_type"));
        }
        if (experimentData.contains("pert_itime")) {
            exp.setExptime((String)dataset.getColumnMetadata().getValue(i,
"pert_itime"));
        }
        int contadorSig = 0;

        for (Map.Entry<String, Double> entrySig :
signatureDataSetUp.entrySet()) {

            sim = new Similitud();
            int contadorSampl = 0;
            for (Map.Entry<String, Double> entry : sampleUp.entrySet()) {
                if (entrySig.getKey().equals(entry.getKey())) {
                    sim.setGeneName(entrySig.getKey());
                    sim.setIndiceSamp(contadorSampl);
                    sim.setZscoreSamp(entry.getValue());
                    sim.setZscoreSig(entrySig.getValue());
                    numSimil++;
                    sim.setIdSimilitud(numSimil);
                }
                contadorSampl++;
                if (!Strings.isNullOrEmpty(sim.getGeneName())) {
                    sim.setIndiceSig(contadorSig);
                    sim.setExperimento(exp);
                }

            }
            contadorSig++;
            if (!Strings.isNullOrEmpty(sim.getGeneName())) {
                similitudesInExp.add(sim);
            }
```

```
            }
            if (!similitudesInExp.isEmpty()) {
                exp.setSimilitudList(similitudesInExp);
                exp.setNcoincidencias(similitudesInExp.size());
            }
        }
        if (exp != null) {
            experimentosUp.add(exp);
        }

        //En el caso que se hayan determinado que hay coincidencias en
downregulation se guardan los datos
        //de los experimentos donde haya coincidencia y cuáles son.
        if (coincidenciasDown > 0) {
            similitudesInExp = Lists.newArrayList();
            exp = new Experimento();
            exp.setResultado(resultado);
            exp.setUpDown(-1);
            exp.setNombre((String)dataset.getColumnMetadata().getValue(i,
"id"));
            exp.setIndiceColumna(i);
            if (experimentData.contains("pert_desc")) {
                exp.setDescripert((String)dataset.getColumnMetadata().getValue(i,
"pert_desc"));
            }
            if (experimentData.contains("cell_id")) {
                String cell_id = (String)dataset.getColumnMetadata().getValue(i,
"cell_id");
                exp.setCell(JpaUtil.getDAO().find(CellTypes.class, cell_id));
            }
            if (experimentData.contains("is_gold")) {
                if (((String)dataset.getColumnMetadata().getValue(i,
"is_gold")).equals("True")) {
                    exp.setGold(1);
                } else {
                    exp.setGold(0);
                }
            }
            if (experimentData.contains("pert_idose")) {
                exp.setDose((String)dataset.getColumnMetadata().getValue(i,
"pert_idose"));
            }
            if (experimentData.contains("pert_iname")) {
                exp.setNamepert((String)dataset.getColumnMetadata().getValue(i,
"pert_iname"));
            }
            if (experimentData.contains("pert_type")) {
                exp.setNamepert((String)dataset.getColumnMetadata().getValue(i,
"pert_type"));
            }
```

```
            if (experimentData.contains("pert_itime")) {
                exp.setExptime((String)dataset.getColumnMetadata().getValue(i,
"pert_itime"));
            }
            int contadorSig = 0;

            for (Map.Entry<String, Double> entrySig :
signatureDataSetDown.entrySet()) {

                sim = new Similitud();
                int contadorSampl = 0;
                for (Map.Entry<String, Double> entry : sampleDown.entrySet()) {
                    if (entrySig.getKey().equals(entry.getKey())) {
                        sim.setGeneName(entrySig.getKey());
                        sim.setIndiceSamp(contadorSampl);
                        sim.setZscoreSamp(entry.getValue());
                        sim.setZscoreSig(entrySig.getValue());
                        numSimil++;
                        sim.setIdSimilitud(numSimil);
                    }
                    contadorSampl++;
                    if (!Strings.isNullOrEmpty(sim.getGeneName())) {
                        sim.setIndiceSig(contadorSig);
                        sim.setExperimento(exp);
                    }

                    }
                contadorSig++;
                if (!Strings.isNullOrEmpty(sim.getGeneName())) {
                    similitudesInExp.add(sim);

                }

            }
            if (!similitudesInExp.isEmpty()) {
                exp.setSimilitudList(similitudesInExp);
                exp.setNcoincidencias(similitudesInExp.size());
            }
        }
        if (exp != null) {
            experimentosDown.add(exp);
        }
    }


    //Ordenación según similitud de los experimentos hacia la muestra.
    if (typesRegulation.equals("1")) {
        experimentosUp = orderingExperiments(experimentosUp);
        for (Experimento e : experimentosUp) {
            nExperimento++;
```

```
            e.setIdExperimento(nExperimento);
            for (Similitud simi : e.getSimilitudList()) {
                similitudes.add(simi);
            }
        }
    } else {

        experimentosDown = orderingExperiments(experimentosDown);
        for (Experimento eD : experimentosDown) {
            nExperimento++;
            eD.setIdExperimento(nExperimento);
            for (Similitud simiD : eD.getSimilitudList()) {
                similitudes.add(simiD);
            }
        }

    }

    } catch (Exception ex) {
        Logger.getLogger(ComparisonController.class.getName());
    } finally {
          reader.close();
    }
```

MÉTODO orderingExperiments

```
//Mapa donde recojo, el número de similitudes y el índice de los experimentos que las tienen.
    Map<Integer,List<Integer>> nSimilitudes = Maps.newHashMap();
    for (int cont = 1; cont < resultado.getGenefilter() + 1; cont++) {
        nSimilitudes.put(cont, Lists.newArrayList());
    }

    for (Experimento e: experiments) {
        List<Integer> indices = Lists.newArrayList();
        if (resultado.getGenefilter() == 1) {
            if (e.getSimilitudList().size() == 1) {
                indices = nSimilitudes.get(1);
                indices.add(e.getIndiceColumna());
                nSimilitudes.replace(1, indices);
            }
        } else if (resultado.getGenefilter() == 5) {
            if (e.getSimilitudList().size() == 1) {
                indices = nSimilitudes.get(1);
                indices.add(e.getIndiceColumna());
                nSimilitudes.replace(1, indices);
            } else if(e.getSimilitudList().size() == 2) {
                indices = nSimilitudes.get(2);
                indices.add(e.getIndiceColumna());
```

```
            nSimilitudes.replace(2, indices);
        } else if(e.getSimilitudList().size() == 3) {
            indices = nSimilitudes.get(3);
            indices.add(e.getIndiceColumna());
            nSimilitudes.replace(3, indices);
        } else if(e.getSimilitudList().size() == 4) {
            indices = nSimilitudes.get(4);
            indices.add(e.getIndiceColumna());
            nSimilitudes.replace(4, indices);
        } else if(e.getSimilitudList().size() == 5) {
            indices = nSimilitudes.get(5);
            indices.add(e.getIndiceColumna());
            nSimilitudes.replace(5, indices);
          }
    } else if (resultado.getGenefilter() == 10) {
        if (e.getSimilitudList().size() == 1) {
            indices = nSimilitudes.get(1);
            indices.add(e.getIndiceColumna());
            nSimilitudes.replace(1, indices);
         } else if(e.getSimilitudList().size() == 2) {
            indices = nSimilitudes.get(2);
            indices.add(e.getIndiceColumna());
            nSimilitudes.replace(2, indices);
         } else if(e.getSimilitudList().size() == 3) {
            indices = nSimilitudes.get(3);
            indices.add(e.getIndiceColumna());
            nSimilitudes.replace(3, indices);
         } else if(e.getSimilitudList().size() == 4) {
            indices = nSimilitudes.get(4);
            indices.add(e.getIndiceColumna());
            nSimilitudes.replace(4, indices);
         } else if(e.getSimilitudList().size() == 5) {
            indices = nSimilitudes.get(5);
            indices.add(e.getIndiceColumna());
            nSimilitudes.replace(5, indices);
         } else if(e.getSimilitudList().size() == 6) {
            indices = nSimilitudes.get(6);
            indices.add(e.getIndiceColumna());
            nSimilitudes.replace(6, indices);
         } else if(e.getSimilitudList().size() == 7) {
            indices = nSimilitudes.get(7);
            indices.add(e.getIndiceColumna());
            nSimilitudes.replace(7, indices);
         } else if(e.getSimilitudList().size() == 8) {
            indices = nSimilitudes.get(8);
            indices.add(e.getIndiceColumna());
            nSimilitudes.replace(8, indices);
         } else if(e.getSimilitudList().size() == 9) {
            indices = nSimilitudes.get(9);
            indices.add(e.getIndiceColumna());
```

```
            nSimilitudes.replace(9, indices);
        } else if(e.getSimilitudList().size() == 10) {
            indices = nSimilitudes.get(10);
            indices.add(e.getIndiceColumna());
            nSimilitudes.replace(10, indices);
        }
    } else if (resultado.getGenefilter() == 15) {
       if (e.getSimilitudList().size() == 1) {
            indices = nSimilitudes.get(1);
            indices.add(e.getIndiceColumna());
            nSimilitudes.replace(1, indices);
        } else if(e.getSimilitudList().size() == 2) {
            indices = nSimilitudes.get(2);
            indices.add(e.getIndiceColumna());
            nSimilitudes.replace(2, indices);
        } else if(e.getSimilitudList().size() == 3) {
            indices = nSimilitudes.get(3);
            indices.add(e.getIndiceColumna());
            nSimilitudes.replace(3, indices);
        } else if(e.getSimilitudList().size() == 4) {
            indices = nSimilitudes.get(4);
            indices.add(e.getIndiceColumna());
            nSimilitudes.replace(4, indices);
        } else if(e.getSimilitudList().size() == 5) {
            indices = nSimilitudes.get(5);
            indices.add(e.getIndiceColumna());
            nSimilitudes.replace(5, indices);
        } else if(e.getSimilitudList().size() == 6) {
            indices = nSimilitudes.get(6);
            indices.add(e.getIndiceColumna());
            nSimilitudes.replace(6, indices);
        } else if(e.getSimilitudList().size() == 7) {
            indices = nSimilitudes.get(7);
            indices.add(e.getIndiceColumna());
            nSimilitudes.replace(7, indices);
        } else if(e.getSimilitudList().size() == 8) {
            indices = nSimilitudes.get(8);
            indices.add(e.getIndiceColumna());
            nSimilitudes.replace(8, indices);
        } else if(e.getSimilitudList().size() == 9) {
            indices = nSimilitudes.get(9);
            indices.add(e.getIndiceColumna());
            nSimilitudes.replace(9, indices);
        } else if(e.getSimilitudList().size() == 10) {
            indices = nSimilitudes.get(10);
            indices.add(e.getIndiceColumna());
            nSimilitudes.replace(10, indices);
        } else if(e.getSimilitudList().size() == 11) {
            indices = nSimilitudes.get(11);
            indices.add(e.getIndiceColumna());
```

```
            nSimilitudes.replace(11, indices);
        } else if(e.getSimilitudList().size() == 12) {
            indices = nSimilitudes.get(12);
            indices.add(e.getIndiceColumna());
            nSimilitudes.replace(12, indices);
        } else if(e.getSimilitudList().size() == 13) {
            indices = nSimilitudes.get(13);
            indices.add(e.getIndiceColumna());
            nSimilitudes.replace(13, indices);
        } else if(e.getSimilitudList().size() == 14) {
            indices = nSimilitudes.get(14);
            indices.add(e.getIndiceColumna());
            nSimilitudes.replace(14, indices);
        } else if(e.getSimilitudList().size() == 15) {
            indices = nSimilitudes.get(15);
            indices.add(e.getIndiceColumna());
            nSimilitudes.replace(15, indices);
        }
    } else {
       if (e.getSimilitudList().size() == 1) {
            indices = nSimilitudes.get(1);
            indices.add(e.getIndiceColumna());
            nSimilitudes.replace(1, indices);
        } else if(e.getSimilitudList().size() == 2) {
            indices = nSimilitudes.get(2);
            indices.add(e.getIndiceColumna());
            nSimilitudes.replace(2, indices);
        } else if(e.getSimilitudList().size() == 3) {
            indices = nSimilitudes.get(3);
            indices.add(e.getIndiceColumna());
            nSimilitudes.replace(3, indices);
        } else if(e.getSimilitudList().size() == 4) {
            indices = nSimilitudes.get(4);
            indices.add(e.getIndiceColumna());
            nSimilitudes.replace(4, indices);
        } else if(e.getSimilitudList().size() == 5) {
            indices = nSimilitudes.get(5);
            indices.add(e.getIndiceColumna());
            nSimilitudes.replace(5, indices);
        } else if(e.getSimilitudList().size() == 6) {
            indices = nSimilitudes.get(6);
            indices.add(e.getIndiceColumna());
            nSimilitudes.replace(6, indices);
        } else if(e.getSimilitudList().size() == 7) {
            indices = nSimilitudes.get(7);
            indices.add(e.getIndiceColumna());
            nSimilitudes.replace(7, indices);
        } else if(e.getSimilitudList().size() == 8) {
            indices = nSimilitudes.get(8);
            indices.add(e.getIndiceColumna());
```

```
    nSimilitudes.replace(8, indices);
} else if(e.getSimilitudList().size() == 9) {
    indices = nSimilitudes.get(9);
    indices.add(e.getIndiceColumna());
    nSimilitudes.replace(9, indices);
} else if(e.getSimilitudList().size() == 10) {
    indices = nSimilitudes.get(10);
    indices.add(e.getIndiceColumna());
    nSimilitudes.replace(10, indices);
} else if(e.getSimilitudList().size() == 11) {
    indices = nSimilitudes.get(11);
    indices.add(e.getIndiceColumna());
    nSimilitudes.replace(11, indices);
} else if(e.getSimilitudList().size() == 12) {
    indices = nSimilitudes.get(12);
    indices.add(e.getIndiceColumna());
    nSimilitudes.replace(12, indices);
} else if(e.getSimilitudList().size() == 13) {
    indices = nSimilitudes.get(13);
    indices.add(e.getIndiceColumna());
    nSimilitudes.replace(13, indices);
} else if(e.getSimilitudList().size() == 14) {
    indices = nSimilitudes.get(14);
    indices.add(e.getIndiceColumna());
    nSimilitudes.replace(14, indices);
} else if(e.getSimilitudList().size() == 15) {
    indices = nSimilitudes.get(15);
    indices.add(e.getIndiceColumna());
    nSimilitudes.replace(15, indices);
} else if(e.getSimilitudList().size() == 16) {
    indices = nSimilitudes.get(16);
    indices.add(e.getIndiceColumna());
    nSimilitudes.replace(16, indices);
} else if(e.getSimilitudList().size() == 17) {
    indices = nSimilitudes.get(17);
    indices.add(e.getIndiceColumna());
    nSimilitudes.replace(17, indices);
} else if(e.getSimilitudList().size() == 18) {
    indices = nSimilitudes.get(18);
    indices.add(e.getIndiceColumna());
    nSimilitudes.replace(18, indices);
} else if(e.getSimilitudList().size() == 19) {
    indices = nSimilitudes.get(19);
    indices.add(e.getIndiceColumna());
    nSimilitudes.replace(19, indices);
} else if(e.getSimilitudList().size() == 20) {
    indices = nSimilitudes.get(20);
    indices.add(e.getIndiceColumna());
    nSimilitudes.replace(20, indices);
} else if(e.getSimilitudList().size() == 21) {
```

```
                indices = nSimilitudes.get(21);
                indices.add(e.getIndiceColumna());
                nSimilitudes.replace(21, indices);
            } else if(e.getSimilitudList().size() == 22) {
                indices = nSimilitudes.get(22);
                indices.add(e.getIndiceColumna());
                nSimilitudes.replace(22, indices);
            } else if(e.getSimilitudList().size() == 23) {
                indices = nSimilitudes.get(23);
                indices.add(e.getIndiceColumna());
                nSimilitudes.replace(23, indices);
            } else if(e.getSimilitudList().size() == 24) {
                indices = nSimilitudes.get(24);
                indices.add(e.getIndiceColumna());
                nSimilitudes.replace(24, indices);
            } else if(e.getSimilitudList().size() == 25) {
                indices = nSimilitudes.get(25);
                indices.add(e.getIndiceColumna());
                nSimilitudes.replace(25, indices);
            }
        }
    }
    //En nSimilitudes tengo recogidos los índices de los experimentos y el número de similitudes (matchings)
    // que presentan cada uno de ellos.
    //Saco el orden según la clave.
   nSimilitudes = MapUtil.sortByKey(nSimilitudes);

    //Mostrar los experimentos más similiares determinado por signatureSelected y sus genes.
    //Ordenar experimentos por los que más coincidencias tienen
    List<Experimento> experimentosOrd = Lists.newArrayList();

    for (Map.Entry<Integer,List<Integer>> entry : nSimilitudes.entrySet()) {
        if (entry.getValue().size() > 0) {
            for (Experimento expto : experiments) {
              for (Integer idx : entry.getValue()) {
                if (idx.equals(expto.getIndiceColumna())) {
                    experimentosOrd.add(expto);
                }

              }

            }
        }
    }

    //Los más coincidentes seleccionados por signature, asi los que tienen solo una coincidencia quedarán al final
    List<Experimento> experimentosOrdSelected = Lists.newArrayList();
```

```
for (int l=0; l < resultado.getSignaturefilter(); l++) {
   experimentosOrdSelected.add(experimentosOrd.get(l));
}

List<Experimento> ordenFinal = Lists.newArrayList();
   if (resultado.getSignaturefilter() > 1) {

      //Dentro de los nSignatures con más coincidencias, habría que ordenarlos en función de la posición
      //coincidente con la firma y mostrar el valor. Por la correlación de Spearman. (Entre - 1 y +1)
      //El string será el identificador del experimento
      //Solo puede evaluarse para arrays con más de dos elementos
      Map<String,Double> spearmanCorr = Maps.newHashMap();

      for (Experimento eo : experimentosOrdSelected) {

         if (eo.getSimilitudList().size() > 1) { //Solo se comparan los que tengan más de una signature

            double[] sig = new double[eo.getSimilitudList().size()];
            double[] samp = new double[eo.getSimilitudList().size()];
            int iteracion = 0;

            for (Similitud simi: eo.getSimilitudList()) {
               sig[iteracion] = simi.getIndiceSig();
               samp[iteracion] = simi.getIndiceSamp();
               iteracion++;
            }

            Double spearman = MetodosEstadisticos.correlacionSpearman(sig, samp);

            eo.setSpearman(spearman);
            spearmanCorr.put(eo.getNombre(),spearman);

         } else {//A los que vengan de 1 por defecto valor de spearman 1 y se incluyen en la lista.
            eo.setSpearman(1.0);
            ordenFinal.add(eo);
         }
      }


      //Primero por la correlación Spearman obtenida
      spearmanCorr = MapUtil.sortByValue2(spearmanCorr);
      //Transformo en Lista

      for (Map.Entry<String,Double> entry : spearmanCorr.entrySet()) {
         for (Experimento expto : experiments) {
```

```
                    if (entry.getKey().equals(expto.getNombre())) {
                        expto.setSpearman(entry.getValue());
                        ordenFinal.add(expto);
                    }
                }
            }
```

//Ordenar primero por el valor de Spearman y después por el de número de coincidencias.

```
Collections.sort(ordenFinal,Comparator.comparing(Experimento::getSpearman)
.
            thenComparing(Experimento::getNcoincidencias));


            Collections.reverse(ordenFinal);

        } else {
            ordenFinal = experimentosOrdSelected;
        }
        for (int i = 0; i < ordenFinal.size(); i++) {
            ordenFinal.get(i).setRanking(i);
        }

        return ordenFinal;
```