Universidad de La Laguna
Escuela Superior de Ingeniería y Tecnología
Sección de Ingeniería Informática

Trabajo de Fin de Grado

Biblioteca JAVA para clasificación de datos: JDataMining

JAVA library for data classification: JDataMining

Ivan Steven Ramirez Ladino

La Laguna, 8 de septiembre de 2015

D. **Jesus Manuel Jorge Santiso**, con N.I.F. 42.097.398-S, profesor Titular de Universidad, adscrito al Departamento de Ingeniería Informática y de Sistemas de la Universidad de La Laguna, como tutor

C E R T I F I C A

Que la presente memoria titulada:

"Biblioteca JAVA para clasificación de datos: JDataMining."

ha sido realizada bajo mi dirección por D. **Ivan Steven Ramirez Ladino**, con N.I.F. 49.513.062-L.

Y para que así conste, en cumplimiento de la legislación vigente y a los efectos oportunos, firmo la presente en La Laguna a 8 de septiembre de 2015.

Agradecimientos

En primer lugar quiero declarar mi más profundo agradecimiento a mis padres, quienes con su apoyo incondicional, han hecho posible que haya llegado a esta instancia, que manifiesta la culminación y obtención del Grado en Ingeniería Informática.

"Papá y mamá, gracias, sin ustedes todo esto no hubiese sido posible".

En segundo lugar quiero agradecer a mi esposa, quien me ha ayudado en la redacción y corrección de este trabajo, además me ha brindado la fuerza necesaria en momentos de desasosiego.

"Amor, gracias por apoyarme y colaborarme tanto".

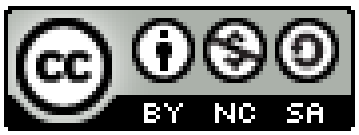
En tercer lugar quiero agradecer a toda mi familia, aunque estén lejos, me han apoyado durante todo el camino para llegar a ser profesional.

"Familia, los quiero mucho".

Y por último pero no menos importante, agradezco al director de este Trabajo de Fin de Grado, D. Jesús Manuel Jorge Santiso, quien ha cumplido con creces su rol para lograr los objetivos planteados.

"Jesús, gracias por desempeñar tan bien su labor".

Licencia



Resumen

El objetivo principal de este trabajo es hacer una biblioteca de clases llamada JDataMining, usando el lenguaje de programación JAVA, para realizar tareas de minería de datos (Data mining) tales como: lectura/escritura de datos, preprocesado de datos, clasificación de datos y realización de experimentos con conjuntos de entrenamiento y pruebas de los datos. Además se proporcionará una interfaz gráfica (GUI) simple, desarrollada en SWING, para comprobar el buen funcionamiento de esta biblioteca.

Lo anteriormente mencionado se llevará a cabo basándose en buenas prácticas de desarrollo, tomando como base el paradigma de la programación orientada a objetos, haciendo uso de patrones de diseño software y empleando sus cinco principios básicos (SOLID): única responsabilidad (Single responsibility – SRP), abierto/cerrado (Open-closed – OCP), sustitución de Liskov (Liskov substitution – LSP), segregación de interface (Interface segregation – ISP) e inversión de dependencia (Dependency inversión – DIP).

Para el desarrollo de la GUI, se seguirá el patrón arquitectónico de software modelo-vista-controlador (MVC).

Palabras clave: Técnicas de clasificación de datos, Minería de datos, Knn, Naive Bayes, Java.

Abstract

The principal purpose of this work is to create a class library called JDataMining, using the JAVA programming language, to do Data mining tasks such as data management, data preprocessing, data classification and generation of experiments with test and training sets of data. A very simple graphic user interface (GUI), developed in SWING, will ensure good functionality of this library.

This process will be carried out based on good development practices, following the object-oriented programming paradigm (OOP) and using software design patterns by employing the "first five principles" (SOLID): Single responsibility (SRP), Open-closed (OCP), Liskov substitution (LSP), Interface segregation (ISP) and Dependency inversion (DIP).

Development of the GUI will follow the pattern of architectural software, model–view–controller (MVC).

Keywords: Data classification techniques, Data Mininig, Knn, Naive Bayes, Java.

Índice General

Capítulo 1. Introducción **1**

1.1 Motivación 1

1.2 Minería de Datos (*Data mining*). 1

1.2.1 Preprocesamiento de los datos. 2

1.2.2 Tareas de la minería de datos. 3

1.2.3 Técnicas de la minería de datos. 4

1.2.4 Interpretación y evaluación de resultados. 5

1.3 Clasificador: K-vecinos más cercanos. 6

1.4 Clasificador: Naive Bayes. 9

Capítulo 2. Tecnologías utilizadas. **12**

2.1 Lenguaje de programación: Java SE 7[4]. 12

2.2 Biblioteca de entorno gráfico: Swing[5] 13

2.3 Biblioteca auxiliar: Java CSV[7] 14

2.4 Lenguaje de programación estadístico: R[8] 14

2.4.1 Intérprete del lenguaje R: Renjin[9]. 15

2.5 Gestor de bibliotecas externas: Maven[10]. 15

2.6 Javadoc[11]. 16

2.7 Sistema de control de versiones: Git[12]. 16

2.7.1 Servicio de *hosting* de Git: Github[13]. 17

2.8 Entorno integrado de desarrollo: Eclipse[14] 18

2.8.1 *Plug-in* Swing de Eclipse: WindowBuilder[15] 18

2.8.2 *Plug-in* UML de Eclipse: UML Lab[16] 19

2.8.3 *Plug-in* Git de Eclipse: EGit[17] 21

2.8.4 *Plug-in* Maven de Eclipse: M2Eclipse[18] 21

2.9 Resumen. 22

Capítulo 3. Descripción de la biblioteca JDataMining. **24**

3.1 Metodología de desarrollo. 24

3.2 Requisitos funcionales. 25
3.3 Modelado de la biblioteca 27
3.4 Proceso de desarrollo 28
3.4.1 Núcleo de la biblioteca. 28
3.4.2 Preprocesado de datos 28
3.4.3 Clasificadores. 30
3.4.4 Fase de experimentación. 32
3.5 Despliegue: API de la biblioteca 32
3.5.1 Paquete: org.ull.jdatamining.core. 32
3.5.2 Paquete: org.ull.jdatamining.filters. 36
3.5.3 Paquete: org.ull.jdatamining.classifiers. 37
3.5.4 Paquete: org.ull.jdatamining.experiments. 40
3.5.5 Paquete: org.ull.jdatamining.util. 42

Capítulo 4. Descripción de la aplicación de escritorio. 46
4.1 Requisitos funcionales del entorno gráfico. 46
4.2 Diagramas de casos de uso 49
4.2.1 Caso de uso 1 49
4.2.2 Caso de uso 2 49
4.2.3 Caso de uso 3 50
4.2.4 Caso de uso 4 50
4.2.5 Caso de uso 5 51
4.2.6 Caso de uso 6 52
4.3 Presentación de la interfaz gráfica. 53
4.3.1 Pantalla de inicio 53
4.3.2 Pantalla análisis de datos: Exploración 53
4.3.3 Pantalla análisis de datos: Clasificadores. 54
4.3.4 Pantalla análisis de datos: Resultados 55
4.3.5 Pantalla experimentación: Configuración 56
4.3.6 Pestaña de configuración: Cargar conjunto de entrenamiento y pruebas 57

Capítulo 5. Conclusiones y líneas futuras **59**
5.1 Líneas futuras..... 59

Capítulo 6. Summary and Conclusions **61**
6.1 Future woks..... 61

Apéndice A. Ficheros de configuración. **63**
A.1. Fichero de configuración pom.xml de Maven..... 63

Apéndice B. Enlaces de interés. **64**
B.1. Repositorio público de la biblioteca JDataMining. 64
B.2. Documentación de la biblioteca JDataMining..... 64

Bibliografía **65**

Índice de figuras

Figura 1.1. Diagrama de flujo algoritmo Knn..... 7
Figura 1.2. Modelo grafico Naive Bayes..... 10
Figura 2.1. Descripción conceptual de Java SE 7[4]..... 13
Figura 2.2. Diagrama de clases de la biblioteca Swing[6]..... 14
Figura 2.3. Diseñador visual WYSIWYG, WindowBuilder. 18
Figura 2.4. Editor de propiedades, WindowBuilder. 19
Figura 2.5. Añadir eventos a controles, WindowBuilder..... 19
Figura 2.6. Generación de código a partir de diagramas de clases, Uml Lab. 20
Figura 2.7. Vista de la línea de trabajo, EGit. 21
Figura 2.8. Menú contextual, EGit. 22
Figura 3.1. Metodología de desarrollo usada en el trabajo..... 24
Figura 3.2. Diagrama de clases, JDataMining. 28
Figura 3.3. Diagrama de clases, núcleo principal..... 29
Figura 3.4. Diagrama de clases, filtros. 30
Figura 3.5. Diagrama de clases general de clasificadores..... 31
Figura 3.6. Diagrama especifico clasificador Knn. 32
Figura 3.7. Diagrama de clases del paquete de experimentación. 33
Figura 4.1. Diagrama de caso de uso 1: Pantalla principal. 49
Figura 4.2. Diagrama de caso de uso 2: Pantalla de análisis. 49
Figura 4.3. Diagrama de caso de uso 3: Pestaña de exploración..... 50
Figura 4.4. Diagrama de caso de uso 4: Pestaña de clasificación..... 51
Figura 4.5. Diagrama de caso de uso 5: Pantalla de experimentación..... 51
Figura 4.6. Diagrama de caso de uso 6: Configuración del experimento. 52
Figura 4.7. Pantalla de inicio..... 53
Figura 4.8. Pantalla análisis de datos: Exploración. 54
Figura 4.9. Pantalla análisis de datos: Clasificadores. 55

Figura 4.10. Pantalla análisis de datos: Resultados..... 56
Figura 4.11. Pantalla experimentación: Configuración..... 57
Figura 4.12. Pantalla experimentación: Cargar conjunto de entrenamiento y pruebas..... 58

Índice de tablas

Tabla 2.1. Resumen de las tecnologías usadas..... 23

Capítulo 1. Introducción

1.1 Motivación

Este trabajo surgió como una continuación de las tareas realizadas dentro de la asignatura de Modelado de Sistemas Software, la cual se da en el itinerario de Ingeniería de Software del grado. En ella se expuso la idea de que antes del desarrollo del *software*, se debe realizar un buen análisis y diseño previos para obtener un producto de calidad. En la asignatura se planteó como caso práctico, un primer diseño de la biblioteca que se ha abordado en este trabajo.

Por otro lado, se interiorizaron las buenas prácticas del desarrollo *software*, vistas en la asignatura de cuarto año del itinerario ya mencionado; Diseño Arquitectónico y Patrones, donde se trabajó con temas importantes para este trabajo tales como los cinco principios básicos de la programación orientada a objetos (SOLID) y los patrones de diseño.

Por último, destacar el interés que ha despertado el tema de la minería de datos en el autor de este TFG, pues se necesitan conocimientos adquiridos durante todo el transcurso del Grado en Ingeniería Informática, y la madurez y experiencia adquiridos con el lenguaje de programación JAVA.

1.2 Minería de Datos (*Data mining*).

Se puede definir como el proceso de extracción de conocimiento a partir de grandes conjuntos de datos [1]

Aunque la minería de datos también es la etapa de análisis en el descubrimiento de conocimiento en bases de datos (KDD[1]); donde su objetivo principal es extraer la información de un conjunto de datos y transformarlos

[1] Knowledge Discovery in Databases

en una estructura comprensible para su uso posterior. Para llevar a cabo esta transformación, la minería de datos se apoya en métodos de inteligencia artificial, aprendizaje automático, estadística, computación paralela y sistemas de toma de decisiones.

Los distintos campos donde se aplica la minería de datos son[1]:

- Aplicaciones financieras y banca
- Análisis de mercado
- Seguros y salud privada
- Educación
- Procesos industriales
- Medicina
- Biología y biotecnología
- Telecomunicaciones
- Otros (Correo electrónico, recursos humanos, web, turismo, hacienda, deportes, política, etc.)

En la minería de datos se trabaja principalmente dos tipos de modelos: predictivos y descriptivos.

- Los **modelos predictivos**, estiman valores futuros o desconocidos de variables de interés, denominadas variables objetivo, tomando como referencia otras variables predictivas. Este tipo de modelo es el que vamos a abordar en este trabajo.
- Los **modelos descriptivos**, identifican patrones que explican o resumen los datos. A diferencia de los modelos predictivos, sirven para explorar las propiedades de los datos examinados.

1.2.1 Preprocesamiento de los datos.

Para que los modelos de los datos mantengan uniformidad a la hora de aplicar procedimientos para su posterior análisis, es fundamental realizar limpieza y transformaciones. El éxito de un proceso de minería de datos depende no sólo de tener todos los datos necesarios, sino que sean íntegros, completos y consistentes[1].

El preprocesado de los datos se lleva a cabo bajo el apartado de filtros donde podemos añadir, modificar y eliminar los datos y donde podemos aplicar las siguientes transformaciones: normalización y estandarización.

a) **Normalización:** Se aplica la normalización lineal uniforme en la que se genera números entre 0 y 1 utilizando la siguiente fórmula[1], donde es necesario conocer el valor máximo (max) y mínimo (min) del atributo:

$$v' = \frac{v - \min}{max - min}$$

Aunque también podemos modificar el rango de valores aplicando una traslación (t) y un escalado (s) de la siguiente forma:

$$v' = \frac{v - \min}{max - min} * s + t$$

b) **Estandarización:** El objetivo es hacer que los datos se comporten como una distribución normal. Para lograr esto, usamos la fórmula[2]:

$$v' = \frac{v - \mu}{\sigma}$$

Donde (μ) corresponde a la media estadística y (σ) corresponde a la desviación típica de los datos del atributo.

1.2.2 Tareas de la minería de datos.

Hay múltiples tareas que se pueden realizar en minería de datos. Cada una de ellas se puede identificar como un tipo de problema a ser resuelto por un algoritmo de minería de datos. Esto nos lleva a considerar que cada tarea tiene sus propios requisitos y el tipo de información de una terea a otra puede ser variable.

Como se hizo referencia anteriormente, las tareas pueden ser predictivas o descriptivas. Entre las tareas predictivas se encuentra la clasificación y la regresión. Las tareas descriptivas pueden ser: el agrupamiento (*clustering*), las reglas de asociación, asociaciones secuenciales y correlación.

Como las tareas que se van a desarrollar en este trabajo son predictivas, nos centraremos en la clasificación.

La **clasificación** es la tarea más usada en la minería de datos. Consiste en determinar la clase a la que pertenece una instancia dada. Cada instancia pertenece a una sola clase, lo cual se indica mediante el valor de un atributo

denominado, *clase de la instancia.* Este atributo puede tomar diferentes valores discretos, cada uno perteneciente a una clase distinta. El resto de los atributos se usan para predecir la clase.

Existen dos tipos de clasificación: la supervisada y la no supervisada.

- La **clasificación supervisada**, cuenta con un conocimiento a priori. Es decir, para la tarea de clasificar un valor dentro de una clase, contamos con instancias ya clasificadas. Este tipo de clasificación es la que trataremos en el actual trabajo.
- En la **clasificación no supervisada**, el algoritmo clasificador no necesita de más información que la instancia a clasificar y algunos parámetros que limitan el número de clases.

1.2.3 Técnicas de la minería de datos.

Existe un gran número de técnicas empleadas en la minería de datos, desde la estadística, hasta los algoritmos evolutivos. Algunas de esas técnicas se listan a continuación:

- Modelización estadística paramétrica y no paramétrica
- Reglas de asociación y dependencia
- Arboles de decisión y dependencia
- Métodos relacionales y estructurales
- Redes neuronales artificiales
- Máquinas de vectores de soporte
- Extracción de conocimiento con algoritmos evolutivos y reglas difusas
- Métodos basados en casos y en vecindad
- Métodos Bayesianos

Los dos últimos ítems de la lista serán abordados en este trabajo. El primero con el algoritmo Knn[2] y el segundo con el algoritmo Naive Bayes.

1.2.4 Interpretación y evaluación de resultados.

Como el objetivo de la minería de datos es descubrir patrones que deben gozar de cierta calidad, se persiguen tres cualidades: que el patrón sea preciso,

[2] K-Nearest Neighbors

comprensible e interesante. Sin embargo, las tres cualidades no se suelen cumplir al mismo tiempo[1].

Evaluación de resultados: Para realizar esta tarea es necesario dividir el conjunto de datos en dos subconjuntos: el conjunto de entrenamiento (*Training Set*) y el conjunto de pruebas (*Test Set*). Esta división nos garantiza que la validación de la precisión del modelo es una medida independiente.

- **Validación simple:** Consiste en reservar un porcentaje de la base del conjunto de datos como conjunto de prueba, el cual no se usa para construir el modelo. Este porcentaje varía entre 5% y 50%. Para una realización correcta, la división de los grupos debe ser aleatoria.
- **Validación cruzada (*cross validation*)**: Consiste en dividir aleatoriamente el *dataset* en dos conjuntos de datos equitativos con los que se estima la precisión predictiva del modelo. Para esto, se crea un modelo con el primer conjunto y se usa para predecir el resultado en el segundo conjunto, calculando así un ratio de error. Luego se construye un modelo con el segundo conjunto y se usa para predecir los resultados del primer conjunto, obteniendo el segundo ratio de error. Por último, se construye un modelo con todos los datos, se calcula el promedio de los ratios y se usa para estimar su precisión.
- **Validación cruzada con n pliegues (*n-folds cross validation*)**: Es el método más usado. En este caso los datos se dividen aleatoriamente en n grupos. Un grupo se reserva para el conjunto de prueba y con los otros $n-1$ restantes, se construye un modelo que se usa para predecir el resultado de los datos del grupo reservado. Así se calculan n ratios de error independientes. Por último, se construye un modelo con todos los datos y se obtienen sus ratios de error y precisión, promediando los ratios de error.

Interpretación de resultados. En las tareas de clasificación es importante usar la precisión como medida de calidad. Esto tiene ciertas desventajas, una de ellas es tener distribuciones de clases no equilibradas, es decir, muchas instancias de una clase y muy pocas de otras. Para solucionar la necesidad de conocer mejor el tipo de error y su coste asociado, se usa la matriz de confusión.

La **matriz de confusión** muestra el recuento de los casos de las clases predichas y sus valores reales. Cada columna de la matriz representa el número de predicciones de cada clase, mientras que cada fila representa a las instancias en la clase real.

Con la matriz de confusión es muy fácil conocer la **precisión predictiva** de la clasificación (relación entre el número de aciertos y el número de predicciones realizadas), simplemente basta con sumar los valores de la diagonal de la matriz de confusión y dividirlo por la suma de todos sus coeficientes.

1.3 Clasificador: K-vecinos más cercanos.

Knn (*K nearest neighbors*), es un método basado en casos y en vecindad. Recibe este nombre, porque la predicción se basa en la utilización del conjunto de instancias "vecinas" a la instancia por clasificar[1]. La clase a asignar será la más frecuente.

El algoritmo calcula todas las distancias que hay entre las instancias del conjunto de instancias y la instancia a clasificar. Se seleccionan las k instancias con menor distancia. Después se procede a la votación, donde la instancia a clasificar obtiene su clase de las k instancias del *dataset* con el mayor número de votos.

Aunque se tiene en cuenta los k vecinos más cercanos, no sólo se depende de este parámetro para aumentar la precisión de la predicción, también entran en juego parámetros como la elección de la distancia a usar, los pesos asignados a cada caso de la vecindad o las reglas de selección de votación utilizadas.

Para tener una vista global del comportamiento del algoritmo, se muestra en la Figura 1.1 el diagrama de flujo del mismo.

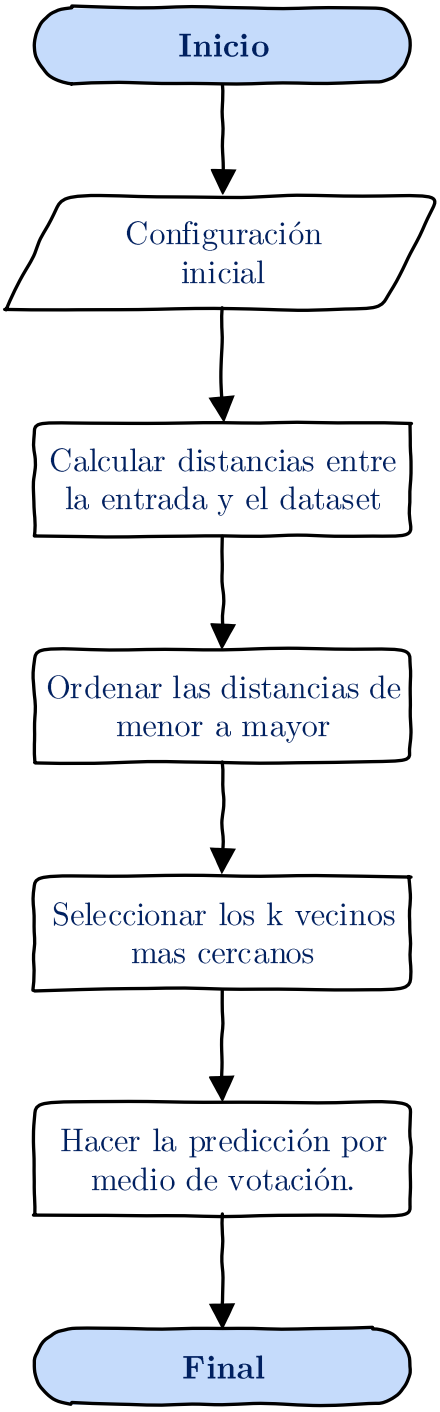


Figura 1.1. Diagrama de flujo algoritmo Knn.

1. **Configuración inicial.** Se precisa el número de vecinos a tener en cuenta (se asigna el valor k). Se establece la función de distancia. Se definen los pesos de las k instancias más cercanas. También se selecciona la regla de votación para efectuar la clasificación.
2. **Calculo de distancias.** Se aplica la función distancia seleccionada en el paso anterior. Las medidas de distancia pueden ser: Euclídea, Manhattan, Chebychef, del Coseno, Mahalanobis, entre otras. En este

trabajo se han implementado las tres primeras mencionadas. Considerando p y q como instancias arbitrarias, tenemos las siguientes métricas:

- **Distancia Euclídea**. Es la distancia clásica, como la longitud de la recta que une dos puntos en el espacio euclídeo:

$$d(p,q) = \sqrt{\sum_{i=1}^{n}(q_i - p_i)^2}$$

- **Distancia Manhattan.** También conocida como la distancia por cuadras (*city-block*). Hace referencia a recorrer un camino zigzagueando. Su origen se debe al recorrido que debe hacer un coche en la ciudad de Manhattan:

$$d(p,q) = \sum_{i=1}^{n}|p_i - q_i|$$

- **Distancia Chebychef.** Calcula la discrepancia más grande en alguna de las dimensiones:

$$d(p,q) = \max(|p_i - q_i|)$$

3. **Ordenar distancias.** Se ordena todo el conjunto de datos en función de la distancia resultante. En este trabajo se optó por crear una estructura de datos que mejora considerablemente a este paso.
4. **Seleccionar los k vecinos más cercanos.** Se selecciona el número de vecinos que van a intervenir en el proceso de clasificación.
5. **Predicción por votación.** Se fija el peso de la instancia a clasificar, pudiendo optar por las siguientes opciones: i) todos los casos tienen el mismo peso, ii) se asigna un peso fijo a cada caso o iii) el peso es inversamente proporcional a la distancia.

Con lo anterior, se procede a hacer la predicción de la clase por medio de la regla de votación seleccionada: mayoría simple o mayoría con umbral (*threshold majority*).

- **Mayoría simple:** La clase más votada es la seleccionada.
- **Mayoría con umbral:** La clase que supere el umbral establecido por el usuario es candidata para ser seleccionada. En caso que sea más de

una, se selecciona la más votada. Cuando el umbral es del 50% la regla se denomina: mayoría absoluta.

1.4 Clasificador: Naive Bayes.

Los **métodos bayesianos** y la teoría de la probabilidad son técnicas que se han usado frecuentemente para resolver problemas de inteligencia artificial, de ahí su estrecha relación con la minería de datos. Según Mitchell[3], estas son las razones por las cuales los métodos bayesianos son relevantes:

1. Son prácticos a la hora de realizar inferencia de datos, induciendo modelos probabilísticos que serán usados para razonar sobre nuevos valores observados. También, son capaces de calcular de forma explícita la probabilidad de cada una de las hipótesis posibles.
2. Son útiles para la comprensión y análisis de varias técnicas de aprendizaje y minería de datos que no trabajan explícitamente con probabilidades.

El **teorema de Bayes** es la regla básica para realizar inferencias en este algoritmo, permitiéndonos pasar de la probabilidad a priori $P(suceso)$ a la probabilidad a posteriori $P(suceso|observaciones)$.

- **Probabilidad a priori**: Es la probabilidad inicial, la que obtenemos sin saber nada más.
- **Probabilidad a posteriori**: Nos indica qué obtendríamos tras conocer cierta información.

La representación matemática del teorema de Bayes viene dada por la siguiente expresión[1]:

$$P(h|O) = \frac{P(O|h)P(h)}{P(O)}$$

$P(h)$: Probabilidad a priori de la hipótesis.
$P(O)$: Probabilidad a priori de las observaciones.
$P(O|h)$: Probabilidad condicionada conocida como la verosimilitud de que la hipótesis h haya producido el conjunto de observaciones O.

Si aplicamos el teorema de Bayes al problema de la clasificación, con una variable de clase (C) y un conjunto de atributos $\{A_1 \dots A_n\}$, nuestra representación matemática quedaría de esta forma[1]:

$$P(C|A_1 \dots A_n) = \frac{P(A_1 \dots A_n|C)P(C)}{P(A_1 \dots A_n)}$$

Hipótesis MAP[3]: Se trata de seleccionar la hipótesis que tiene máxima probabilidad a posteriori dados los valores de los atributos. Todo esto teniendo en cuenta que C tiene k posibles valores $\Omega_C = \{c_1 \dots c_k\}$. Entonces la hipótesis MAP se expresa de la siguiente forma[1]:

$$C_{MAP} = \arg_{c\in\Omega_c} max\, P(C|A_1 \dots A_n) = \arg_{c\in\Omega_c} max \frac{P(A_1 \dots A_n|C)P(C)}{P(A_1 \dots A_n)}$$
$$= \arg_{c\in\Omega_c} max\, P(A_1 \dots A_n|C)P(C)$$

Ω_c: Representa el conjunto de valores que puede tomar la variable C.

Naive Bayes. El fundamento de este clasificador bayesiano es la suposición de que todos los atributos son independientes.

La hipótesis de independencia asumida por el clasificador Naive Bayes da lugar a un modelo grafico probabilístico en el que existe un nodo raíz (la clase), y en el que todos los atributos son hojas que tienen como único padre a la variable de la clase (ver Figura 1.2).

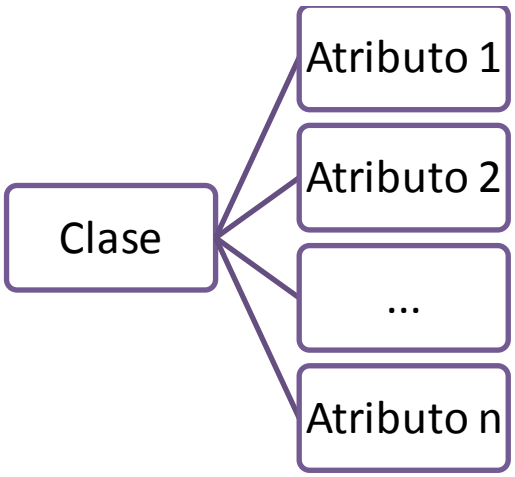


Figura 1.2. Modelo grafico Naive Bayes.

La expresión para obtener la hipótesis MAP queda de la siguiente forma[1]:

$$C_{MAP} = \arg_{c\in\Omega_c} max\, P(A_1 \dots A_n|C)P(C) = \arg_{c\in\Omega_c} max\, P(C) \prod_{i=1} P(A_i|C)$$

Por tanto los parámetros que tenemos que estimar son $P(A_i|C)$ para cada atributo y la probabilidad a priori de la variable de la clase $P(C)$.

[3] Maximum a posteriori

Como los atributos que tratamos pueden ser tanto discretos como continuos, cada uno de ellos debe ser tratado de distinta forma.

a) **Atributos discretos**. Para calcular la probabilidad condicionada cuando los valores son discretos, nos basamos en la frecuencia de aparición que tenemos en el conjunto de datos.

Una de las técnicas que se usan en este caso es la estimación por máxima verosimilitud[1], la cual consiste dividir el número de casos favorable por el número de casos totales:

$$P(x_i|Pa(x_i)) = \frac{n(x_i, Pa(x_i))}{n(Pa(x_i))}$$

Donde $n(x_i, Pa(x_i))$ es el número de valores del conjunto de datos en que la variable Xi toma el valor x_i y los padres de X_i $Pa(X_i)$ toman la configuración denotada por $Pa(x_i)$.

Esta técnica tiene como desventaja necesitar una muestra muy grande y que sobreajusta los datos. Para solucionar esta situación, se empleará el estimador basado en la ley de sucesión de Laplace[1], este describe el número de casos favorables más uno, dividido por el número de casos totales, más el número de valores posibles:

$$P(x_i|Pa(x_i)) = \frac{n(x_i, Pa(x_i)) + 1}{n(Pa(x_i)) + |\Omega_{X_i}|}$$

Con esta estimación logramos que todas las configuraciones posibles tengan una mínima probabilidad, si con el estimador de máxima verosimilitud encontramos un valor que no esté en la base, tendríamos probabilidad cero.

b) **Atributos continuos**. En este caso el clasificador Naive Bayes supone que el atributo sigue una distribución normal[1]; por tanto lo único que se calcula es la media μ y la desviación típica σ condicionadas a cada valor de la variable de la clase:

$$P(A_i|c) \propto N(\mu, \sigma) = \frac{1}{\sqrt{2\pi} \cdot \sigma} e^{-\frac{(X-\mu^2)}{2\sigma^2}}$$

El único inconveniente con esta hipótesis, es que no todos los datos siguen una distribución normal.

Capítulo 2.
Tecnologías utilizadas.

2.1 Lenguaje de programación: Java SE 7[4].

Java es un lenguaje de programación creado en 1991 por Sun Microsystems (actualmente gestionado por Oracle) con la intención de mantener una estructura y sintaxis parecida a C++ pero eliminando las herramientas a bajo nivel.

Los cinco pilares básicos en que se sustenta Java son: la programación orientada a objetos, la posibilidad de ejecutar un mismo programa en diversos sistemas operativos, la inclusión por defecto de soporte para trabajo en red, la opción de ejecutar el código en sistemas remotos de manera segura y la facilidad de uso.

Las aplicaciones de Java se ejecutan en una máquina virtual conocida como Java Virtual Machine (JVM) y esto permite que las aplicaciones funcionen en cualquier entorno sin necesidad de recompilarse.

La realización de este trabajo se pudo hacer gracias a Java Development Kit (JDK), el cual proporciona herramientas de desarrollo para la creación de programas en Java. Dentro de las herramientas se puede contar con: herramientas de programación, Java Runtime Environment (JRE), bibliotecas adicionales, bases de datos de Java, código fuente de la biblioteca estándar, etc.

JRE contiene una máquina virtual de java privada, bibliotecas de clases en tiempo de ejecución y todos los ficheros necesarios para realizar una ejecución de un programa hecho en el lenguaje de programación Java.

Hay gran variedad de plataformas en Java para el desarrollo y despliegue de aplicaciones, las más conocidas son: Java Standard Edition (Java SE), destinada para aplicaciones de estaciones de trabajo aisladas y servidores; Java Enterprise Edition (Java EE), destinada para el ámbito corporativo y

aplicaciones como servicios para la web; Java Micro Edition (Java ME), especialmente usado en sistemas empotrados. Para nuestro propósito usaremos Java SE.

Se ha decidido usar Java SE 7 por motivos de estabilidad y compatibilidad con el entorno de desarrollo elegido. En la Figura 2.1 se puede ver cómo está compuesta la plataforma Java SE 7.

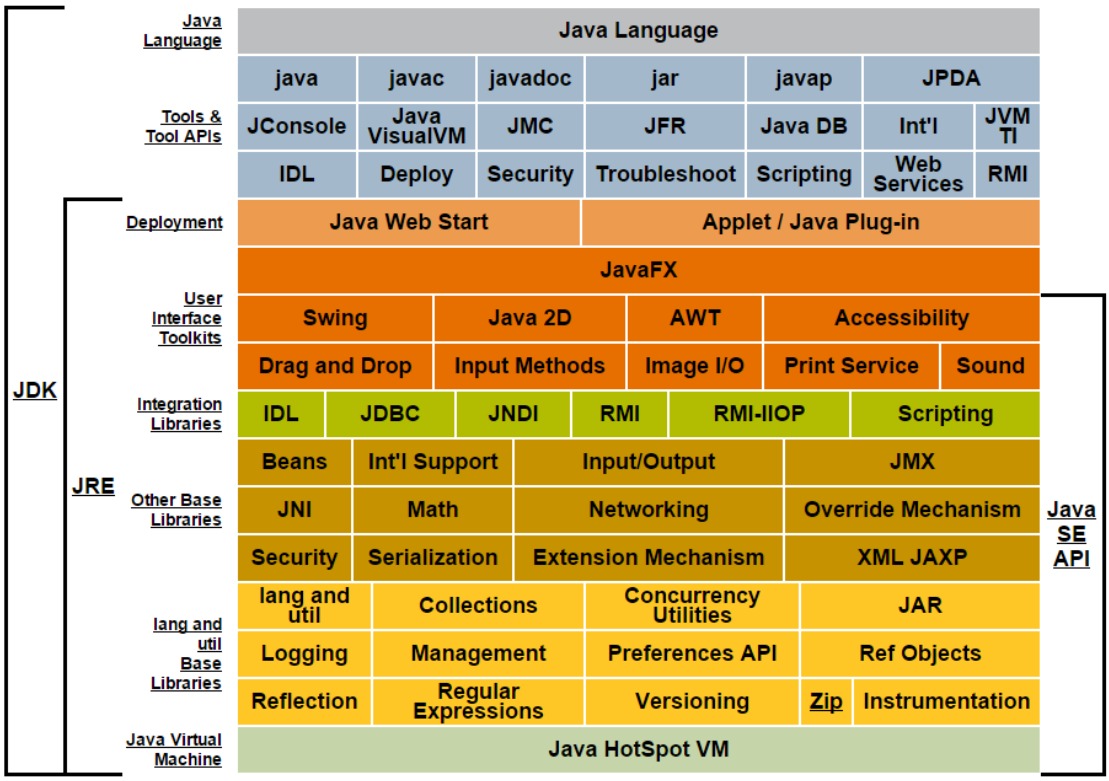


Figura 2.1. Descripción conceptual de Java SE 7[4].

2.2 Biblioteca de entorno gráfico: Swing[5].

Forma parte del *framework* Java Foundation Classes (JFC), junto con AWT y Java2D. Su principal labor es construir interfaces gráficas para cualquier tipo sistema operativo.

Se ha elegido Swing para la elaboración de este trabajo, por la sencillez que otorga en la construcción de la interfaz gráfica. Con el *plug-in* de Eclipse, WindowBuilder (del cual se hablará más adelante), se acelera el proceso de desarrollo de manera sustancial.

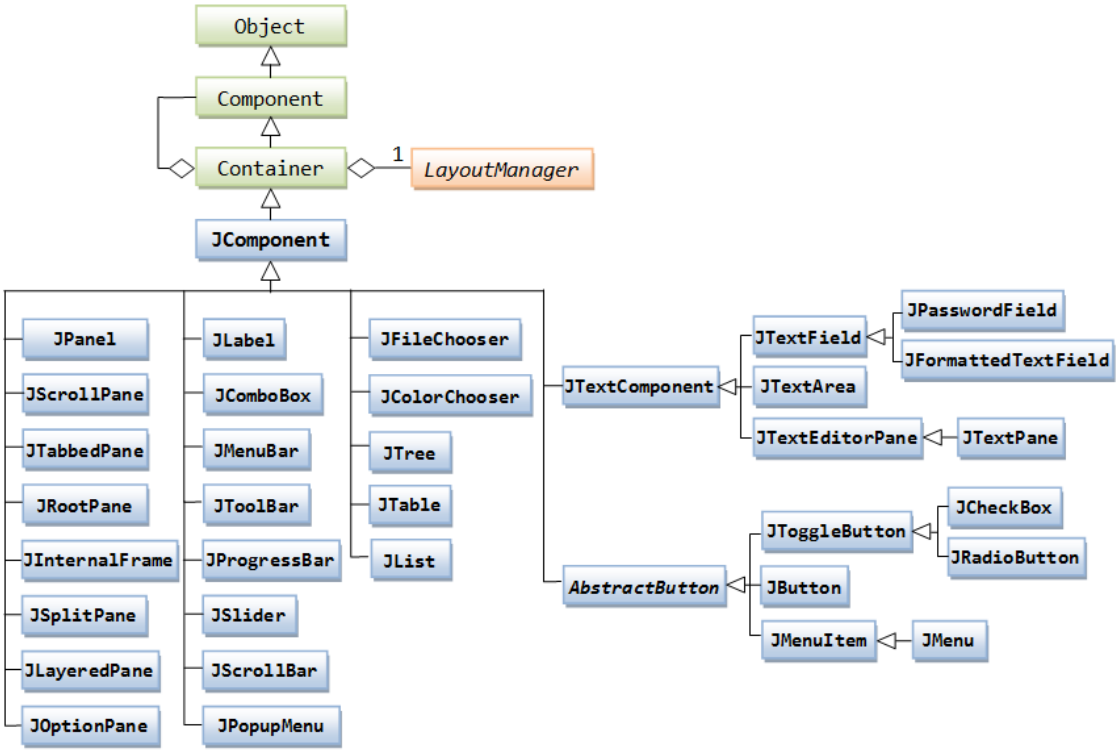


Figura 2.2. Diagrama de clases de la biblioteca Swing[6].

2.3 Biblioteca auxiliar: Java CSV[7].

Es una biblioteca de código abierto para leer y escribir ficheros de texto delimitado por comas, mejor conocido como CSV (*Comma-separated values*).

Sus principales características son:

- Manejar delimitadores de registro y columna dentro de los datos utilizando calificadores de texto citado
- Compatibilidad con la versión de Java 1.4 y superiores
- Licencia: *Lesser General Public License version 2.0* (LGPLv2)

Se eligió esta biblioteca por ser un proyecto de software libre y por su sencillez de uso.

2.4 Lenguaje de programación estadístico: R[8].

R es un lenguaje y un entorno para computación y gráficos estadísticos. Es un proyecto GNU, similar al lenguaje S. Desarrollado en los Laboratorios Bell por John Chambers.

Ofrece una amplia variedad de técnicas gráficas estadísticas (análisis de series de tiempo lineal, pruebas estadísticas clásicas, clasificación, agrupación,...) y es altamente extensible.

Está disponible como software libre, bajo los términos de la Licencia Pública General GNU de Free Software Foundation, en forma de código fuente.

Este lenguaje puede ayudar de forma significativa para los cálculos estadísticos que son altamente frecuentes en las tareas de clasificación. Asimismo nos vale para hacer análisis gráfico de los datos.

2.4.1 Intérprete del lenguaje R: Renjin[9].

Renjin es un intérprete Java del lenguaje de programación R. Al igual que JRuby y Jython lo son para los lenguajes de programación Ruby y Python.

El objetivo principal es ser compatible con R, de forma que la mayoría de programas existentes en el lenguaje R, se puedan ejecutar en Renjin sin necesidad de realizar ningún cambio en el código fuente. Vale la pena aclarar que Renjin actualmente no es 100% compatible con R, aunque está en constante desarrollo para lograrlo.

La mayor ventaja de Renjin es que tiene su propio intérprete de R, el cual es un módulo de Java que se puede integrar perfectamente en cualquier aplicación. Por lo tanto, se evita la carga de bibliotecas dinámicas.

No requiere ninguna fuente adicional para habilitar la interfaz Java/R. Renjin proporciona un acceso directo a los métodos de Java, usando una interfaz completamente discreta.

2.5 Gestor de bibliotecas externas: Maven[10].

Maven es un software para la gestión y construcción de proyectos Java basado en el término *project object model* (POM).

El objetivo principal de Maven, es permitir al desarrollador comprender el estado completo de un proyecto en un periodo de tiempo muy breve. Para lograr esto Maven lleva a cabo las siguientes tareas:

- Hace el proceso de construcción más simple

- Proporcionar un sistema de construcción uniforme
- Proporciona una información de calidad del proyecto
- Proporciona directrices para mejorar las prácticas de desarrollo
- Permite una migración transparente a nuevas características

La principal razón de usar Maven, ha sido la facilidad que ofrece para integrar tecnologías externas. En este caso se ha usado para añadir Renjin y Java CSV.

En el apéndice A.1. Fichero de configuración pom.xml de Maven se puede visualizar el archivo de configuración POM, usado en el desarrollo de la biblioteca JDataMinig.

2.6 Javadoc[11].

Javadoc es una herramienta que genera documentación de archivos de código fuente Java y produce un conjunto de páginas HTML que describen las clases, interfaces, constructores, métodos y campos.

Javadoc proporciona una API para la creación de *doclets* y *taglets*.

D*oclets:* Es un programa que especifica el contenido y el formato de una salida generada por la herramienta Javadoc. Se puede desarrollar un *doclet* para generar todo tipo de archivos de texto como lo son: HTML, SGML, XML, RTF, y MIF.

T*aglest:* Es un programa que permite crear y utilizar etiquetas personalizadas siendo más flexibles que las etiquetas creadas con la opción "`-tag`".

Se ha usado Javadoc para realizar toda la documentación de la biblioteca. Así, se hace más sencilla la ampliación de las funcionalidades de la misma.

2.7 Sistema de control de versiones: Git[12].

Git es un sistema de control de versiones (VCS[4]) gratuito y de código abierto, diseñado para manejar desde pequeños a grandes proyectos con

[4] Version Control System

rapidez y eficiencia. Su funcionamiento es similar a herramientas como Subversion, CVS, Perforce y ClearCase aunque es más eficiente. Las principales características de Git son:

- Ramificación y Fusión (*Branching and Merging*): Permite tener múltiples ramas locales que pueden ser totalmente independientes entre sí. La creación, fusión y supresión de esas líneas de desarrollo se hacen rápidamente.
- Velocidad y tamaño: Como ha sido construido para trabajar en el *kernel* de Linux, ha tenido que gestionar con eficacia grandes repositorios desde el primer día. Git está escrito en C, lo que reduce la sobrecarga de los tiempos de ejecución asociados con lenguajes de alto nivel.
- Distribución: Este VCS es distribuido. Esto significa que no hace una verificación o *checkout* de la rama actual sino que hace un "clon" de todo el repositorio.
- Seguridad de los datos: El modelo de datos que utiliza Git garantiza la integridad criptográfica de cada bit del código fuente.
- *Staging Area*: Es una zona intermedia donde cada *commit* puede ser formateado y revisado antes de completar la confirmación.
- Gratis y de código abierto: Esta liberado bajo la Licencia Pública General GNU versión 2.0. Esto da la garantía y la libertad de compartir y modificar el software libremente.

Para facilitar aún más el trabajo se ha usado el servicio hosting de repositorios Git basado en Web, Github.

2.7.1 Servicio de *hosting* de Git: Github[13].

Github no solo facilita el trabajo en Git empleando una interfaz web y de escritorio, además proporciona controles de acceso y varias funciones de colaboración como: seguimiento de errores, peticiones de características, gestión de tareas y wikis para cada proyecto.

Para mantener una integración completa del trabajo realizado, se decidió hacer todo el control de versiones de la biblioteca con la ayuda del *plug-in* de Eclipse para Git que será comentado más adelante.

2.8 Entorno integrado de desarrollo: Eclipse[14].

Eclipse es un entorno integrado de desarrollo (IDE[5]) que provee múltiples herramientas el manejo de espacios de trabajo (*workspace*); construye, ejecuta y depura aplicaciones; comparte artefactos con un equipo de trabajo y controla las versiones del código.

Este IDE es un mecanismo para el descubrimiento, integración y ejecución de módulos llamados *plug-ins*. Múltiples productos, no necesariamente relacionados, pueden ser instalados en una instancia de eclipse y fácilmente cooperar entre sí.

Debido al gran catálogo con el que cuenta Eclipse respecto a *plug-ins* y a que es el entorno de desarrollo más popular y fácil de usar para el lenguaje Java, lo hemos seleccionado para aumentar la productividad en el proceso de desarrollo.

A continuación se nombrarán los *plug-ins* utilizados:

2.8.1 *Plug-in* Swing de Eclipse: WindowBuilder[15]

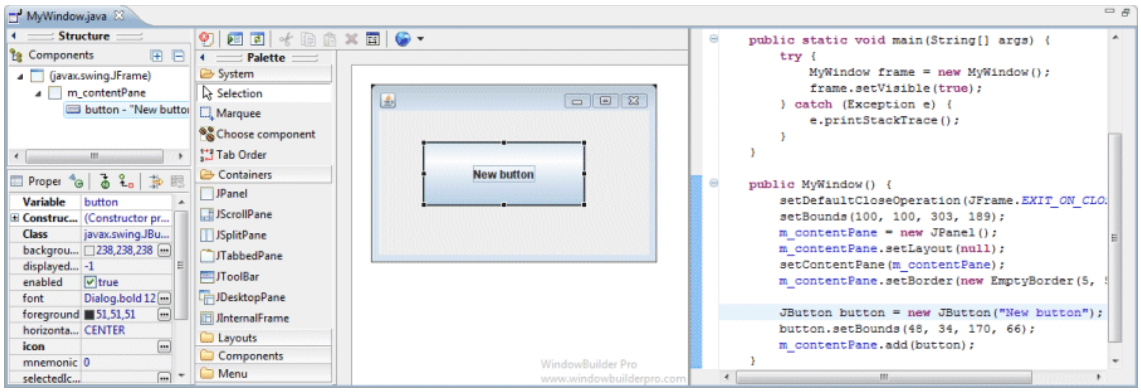


Figura 2.3. Diseñador visual WYSIWYG, WindowBuilder.

Facilita la creación de interfaces gráficas de aplicaciones en Java sin gastar mucho tiempo escribiendo código. Usa el diseñador visual WYSIWYG (Figura 2.3) y herramientas de diseño para crear desde formularios simples a complejas ventanas; generando código en Java automáticamente. Permite añadir controles usando "*drag-and-drop*", cambiar varias propiedades de los controles con el editor de propiedades (Figura 2.4) y agregar manejadores de eventos a los controles (Figura 2.5).

[5] Integrated Development Environment

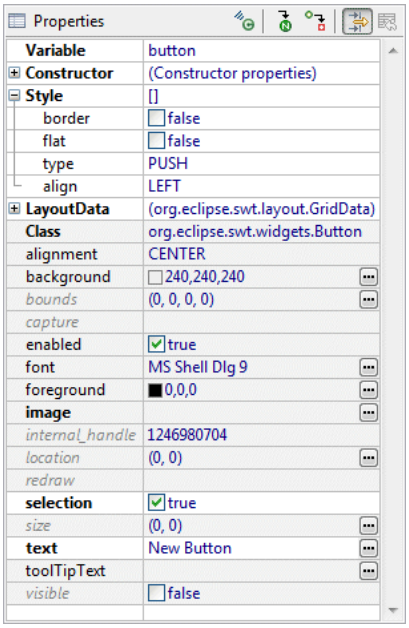


Figura 2.4. Editor de propiedades, WindowBuilder.

WindowBuilder ha sido de gran ayuda para lograr la construcción de la interfaz gráfica que muestra el uso de la biblioteca desarrollada. Gracias a esta herramienta se ha ahorrado tiempo al codificar esta parte del proyecto. También ha contribuido a lograr una interfaz gráfica más ordenada y de simple manejo.

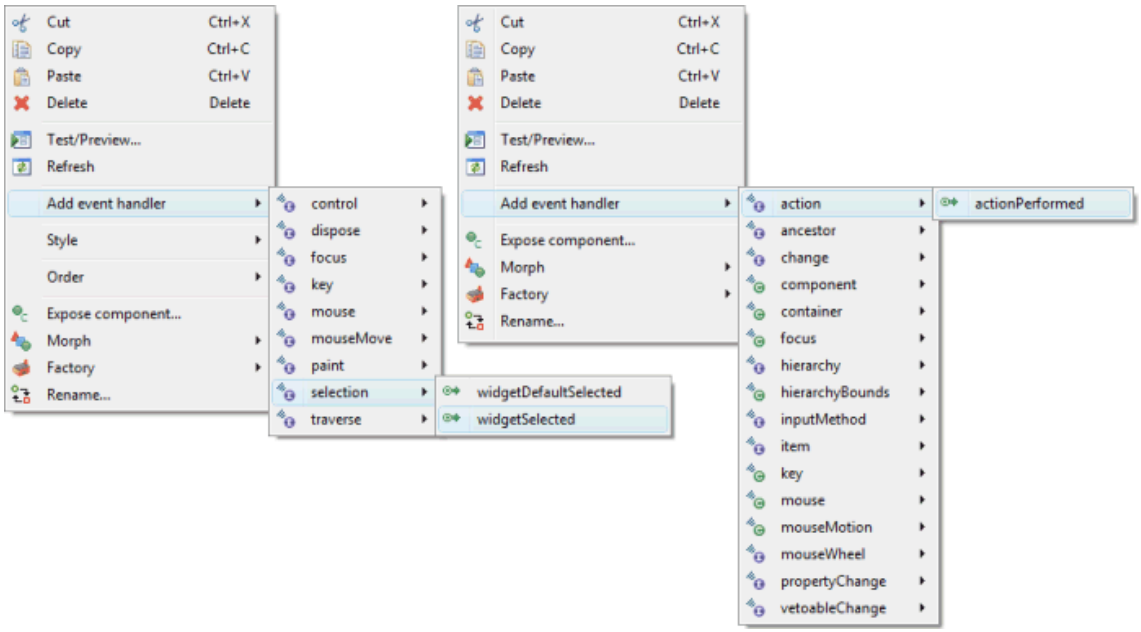


Figura 2.5. Añadir eventos a controles, WindowBuilder.

2.8.2 *Plug-in* UML de Eclipse: UML Lab[16]

Esta herramienta también la encontramos como un IDE, pero se ha optado por integrarlo a Eclipse como *plug-in*. La finalidad de este artefacto es

combinar el modelado y la programación con un editor de diagramas UML intuitivos y con *Round-Trip-Engineering* (conocido como ingeniería inversa y generación de código en tiempo real).

Las principales características de esta herramienta son:

- Ingeniería tanto directa como inversa en tiempo real (Figura 2.6)
- Editor de diagramas de clase UML 2.5 con integración Xpand/Xtend
- Validación de código basado en modelos (OCL[6])
- Navegación y exploración del modelo
- Importación y exportación de XMI[7]
- Exportación de los modelos en varios formatos (SVG, PNG, JPG, BMP)

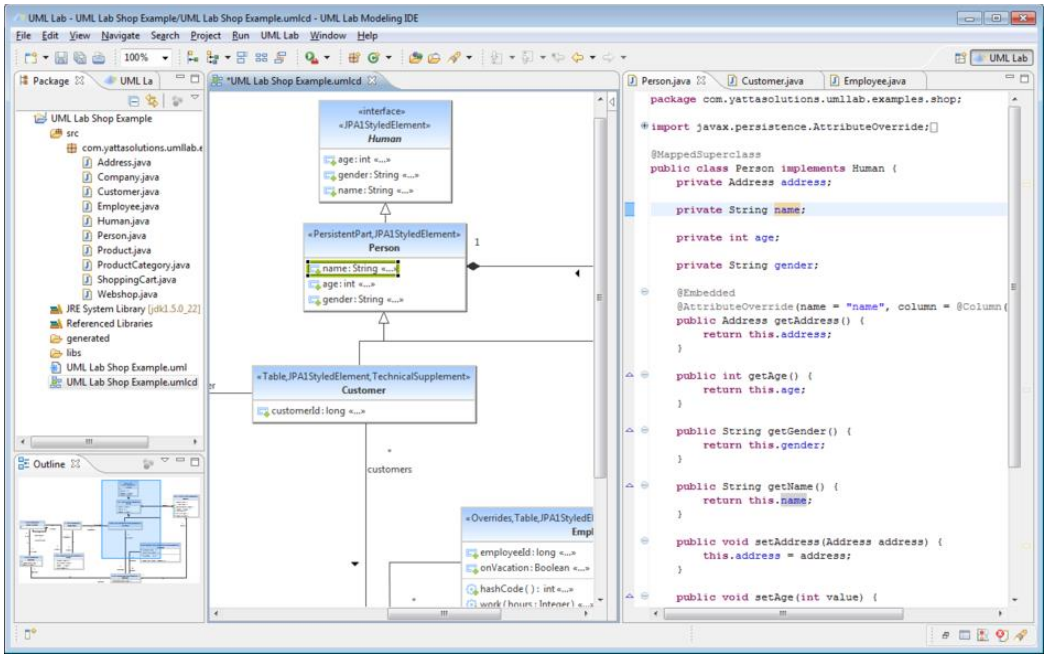


Figura 2.6. Generación de código a partir de diagramas de clases, Uml Lab.

Esta herramienta no cuenta con una licencia de código libre, se trabajó con ella gracias a que posee una licencia para estudiantes. En el inicio del trabajo fue de gran ayuda, debido a que las primeras iteraciones consistían en construir la biblioteca usando modelos sólidos de software y esta herramienta ahorró tiempo, generando el código pertinente. En las iteraciones finales se ha suprimido el uso de este software, debido a que el procesamiento de todas las clases generadas, para la actualización de los modelos, implicaba un gasto de tiempo innecesario.

[6] Object Constraint Language

[7] XML Metadata Interchange

2.8.3 *Plug-in* Git de Eclipse: EGit[17]

Las principales características son:

- Provee asistentes para la realización de acciones comunes en Git de forma intuitiva
- Muestra de forma gráfica las acciones realizadas al proyecto (Figura 2.7)
- Enseña las acciones más comunes en Git desde el menú contextual (Figura 2.8)
- Emite mensajes de dialogo cuando hay un punto de enganche perdido
- Hace plantillas de Java para las variables de configuración de Git

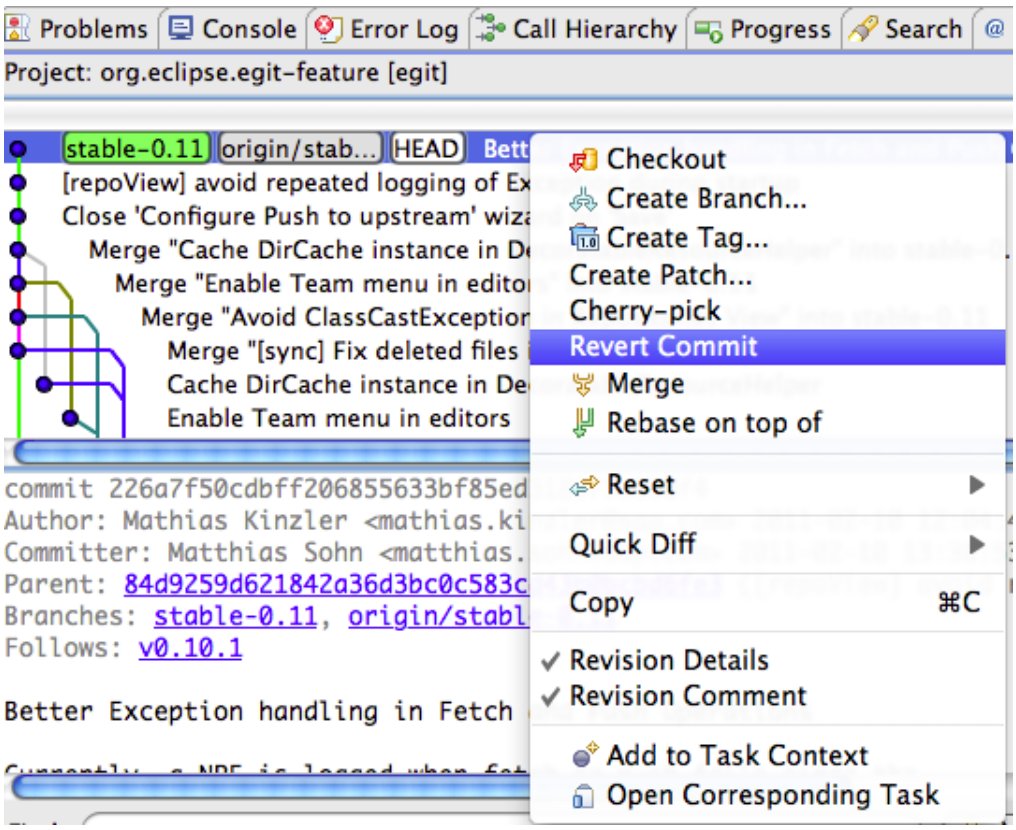


Figura 2.7. Vista de la línea de trabajo, EGit.

En este trabajo se usó EGit para facilitar la labor del mantenimiento del sistema de control de versiones con Git.

2.8.4 *Plug-in* Maven de Eclipse: M2Eclipse[18]

M2eclipse es un *plug-in* que ofrece una estrecha integración de Maven con Eclipse, estas son las características más relevantes:

- Lanzamiento de *Maven builds* desde Eclipse

- Gestión de la dependencia para la ruta de construcción del proyecto en Eclipse, basado en pom.xml de Maven
- Resuelve dependencias de Maven desde el *workspace* de Eclipse, sin necesidad de instalar repositorios locales
- Descarga las dependencias requeridas desde los repositorios Maven remotos
- Provee asistentes para la creación de nuevos proyectos Maven y habilita el soporte en proyectos simples de Java
- Búsqueda rápida de dependencias en repositorios remotos Maven

Aunque el IDE ya hace una integración general de las herramientas anteriormente mencionadas y Maven persigue el mismo objetivo, el uso de este *plug-in* ayuda básicamente en la configuración de dependencias y facilita el mantenimiento del fichero pom.xml.

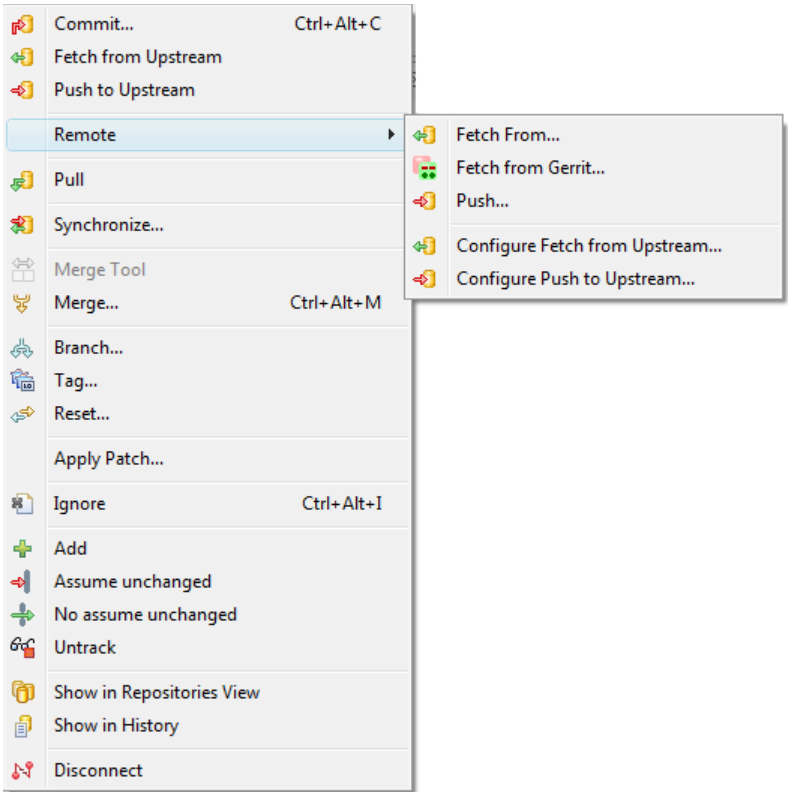


Figura 2.8. Menú contextual, EGit.

2.9 Resumen.

En la Tabla 2.1 se presentan todas las tecnologías usadas en el desarrollo de este trabajo. Se pueden comparar las versiones, las licencias de uso, las plataformas de trabajo, el estado de desarrollo, el tipo y la plataforma donde se ejecuta cada una de las herramientas.

Nombre	Versión usada	Licencia	Última actualización	Tipo	Plataforma
Java SE	7.0	Licencia Pública General de GNU	18/03/2014	Lenguaje de programación	Multiplataforma
Swing	7.0	Licencia Pública General de GNU	18/03/2014	Librería de clases	Multiplataforma
JavaCSV	2.0	GNU GPL v2	10/12/2014	Librería de clases	Multiplataforma
R Project for Statistical Computing	3.2.1	GNU R v2.14.2.	Junio 2015	Lenguaje de programación	Multiplataforma
Renjin	0.7.0-RC7	GNU R v2.14.2.	24/06/2015	Lenguaje interpretado	JVM
Javadoc	1.5.0	Licencia Pública General de GNU	Febrero 2004	Herramienta para generar documentación	Java
Git	2.4.8	GNU GPL v2	03/08/2015	Sistema de control de versiones	Multiplataforma
GitHub	2.3.0	Comercial	19/08/2015	Servicio de alojamiento repositorio Git	Multiplataforma
Eclipse	Luna 4.4.2	Eclipse Public License v1.0	23/07/2015	Entorno integrado de desarrollo	Multiplataforma
WindowBuilder	1.7.0	Eclipse Public License v1.0	10/06/2015	Plug-in	Eclipse
Uml Lab	Student 1.7.1	Comercial	22/06/2015	Plug-in	Eclipse
Egit	3.4.2	Eclipse Public License v1.0	24/06/2015	Plug-in	Eclipse
M2Eclipse	1.5.1	Eclipse Public License v1.0	18/06/2015	Plug-in	Eclipse

Tabla 2.1. Resumen de las tecnologías usadas

Capítulo 3. Descripción de la biblioteca JDataMining.

3.1 Metodología de desarrollo.

Este trabajo se realizó siguiendo una metodología iterativa incremental. Este tipo de metodologías ordenan las etapas del proceso de desarrollo donde la descripción del sistema es importante para especificar cada uno de los incrementos, hasta llegar al producto final. De esta forma el sistema se elabora poco a poco, disfrutando de un *feedback* continuo.

Se usó esta metodología porque existen iteraciones que no tienen por qué realizarse de manera secuencial, pudiendo haber varias tareas ejecutándose simultáneamente siempre que no existan incompatibilidades entre ellas.

Al ser una metodología del tipo evolutivo, donde se permiten y esperan probables cambios en los requisitos en tiempo de desarrollo, se ha considerado la elección más adecuada.

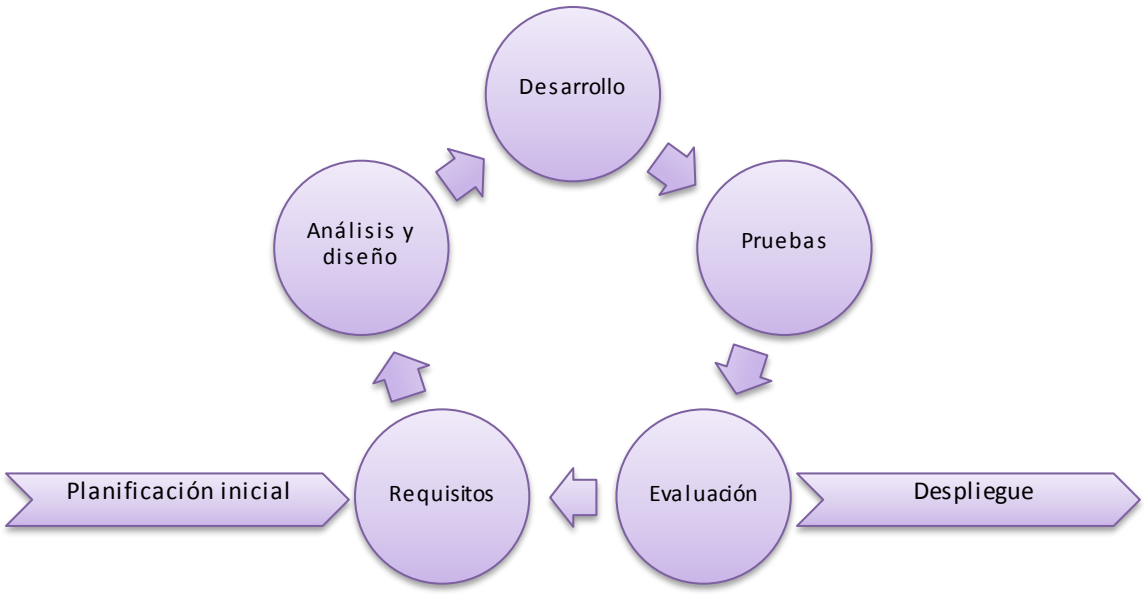


Figura 3.1. Metodología de desarrollo usada en el trabajo.

3.2 Requisitos funcionales.

Estos son los requisitos funcionales que debe cumplir la biblioteca. Vale la pena destacar que esta lista es el resultado final, después de pasar por varias iteraciones donde se fueron refinando cada uno de ellos.

R1. La biblioteca permite manipular datos

- R1.1. La biblioteca permite cargar datos desde un fichero CSV en una estructura de datos definida (*dataset*)
  - R1.1.1. La biblioteca hace la lectura de datos en ficheros CSV
  - R1.1.2. La biblioteca hace la escritura de datos en ficheros CSV
- R1.2. La biblioteca permite manipular los datos como columnas (*Attributes*)
  - R1.2.1. Se define el nombre del atributo
  - R1.2.2. Se define el tipo del atributo
    - R1.2.2.1. Se define un atributo numérico (*Numeric Attribute*)
      - R1.2.2.1.1. Se calcula el valor mínimo del atributo numérico
      - R1.2.2.1.2. Se calcula el valor máximo del atributo numérico
      - R1.2.2.1.3. Se calcula el valor de la media del atributo numérico
      - R1.2.2.1.4. Se calcula el valor de la desviación típica del atributo numérico
    - R1.2.2.2. Se define un atributo nominal (*Nominal Attribute*)
      - R1.2.2.2.1. Se calcula el número de clases del atributo nominal
      - R1.2.2.2.2. Se calcula las frecuencias relativas del atributo nominal
  - R1.2.3. Se define si el atributo es una clase (*Attribute Class*) o no
- R1.3. La biblioteca permite manipular atributos del *dataset*
  - R1.3.1. Cantidad de atributos del *dataset*
  - R1.3.2. *Dataset* como un conjunto de atributos
- R1.4. La biblioteca permite manipular los datos como filas de instancias (*Instances*)
  - R1.4.1. Se implementa una interfaz de instancias para crear una asociación de dependencia al *dataset*

R2. **La biblioteca permite hacer preprocesado a los datos**

R2.1. La biblioteca permite normalizar los datos

R2.2. La biblioteca permite estandarizar los datos

R2.3. La biblioteca proporciona una interfaz para añadir nuevos filtros

R3. La biblioteca permite aplicar algoritmos de clasificación a los datos

R3.1. La biblioteca permite aplicar el algoritmo de clasificación Knn a los datos

R3.1.1. Se puede alterar la configuración de Knn

R3.1.1.1. Se puede modificar el valor de k

R3.1.1.2. Se puede seleccionar diferentes métricas para el cálculo de distancias

R3.1.1.2.1. Aplica la distancia Euclídea para el cálculo de distancias

R3.1.1.2.2. Aplica la distancia Manhattan para el cálculo de distancias

R3.1.1.2.3. Aplica la distancia Chebychef para el cálculo de distancias

R3.1.1.2.4. La biblioteca proporciona una interfaz para añadir nuevas métricas para el cálculo de distancias

R3.1.1.3. Se asignan pesos entre las k instancias más cercanos

R3.1.1.3.1. La biblioteca proporciona una interfaz para añadir nuevos modos de pesado

R3.1.1.4. Se asignan reglas de clasificación al algoritmo

R3.1.1.4.1. Mayoría simple.

R3.1.1.4.2. Votación con umbral.

R3.1.1.4.3. La biblioteca proporciona una interfaz para añadir nuevas reglas de clasificación

R3.2. La biblioteca permite aplicar el algoritmo de clasificación Naive Bayes a los datos

R3.3. La biblioteca proporciona una interfaz para añadir nuevos algoritmos de clasificación de datos

R4. La biblioteca permite generar experimentos

R4.1. La biblioteca permite generar métodos de evaluación simple

R4.1.1. Se definen conjuntos de entrenamiento (*Trainset*)

R4.1.1.1. Conjuntos de entrenamiento aleatorios

R4.1.1.2. Conjuntos de entrenamiento secuenciales

R4.1.2. Se definen conjuntos de pruebas (*Testset*)

R4.1.2.1. Conjuntos de pruebas aleatorios

R4.1.2.2. Conjuntos de pruebas secuenciales

R4.1.3. Se guardan conjuntos de entrenamiento y pruebas en formato CSV

R4.1.4. Se leen conjuntos de entrenamiento y pruebas en formato CSV

R4.2. La biblioteca permite generar métodos de validación cruzada (*cross validation*)

R4.2.1. Validación cruzada secuencial

R4.2.2. Validación cruzada aleatoria

R4.3. La biblioteca genera métodos de validación cruzada con n-pliegues (*n-folds cross validation*)

R4.3.1. Validación cruzada con n-pliegues aleatoria.

R4.4. La biblioteca muestra resultados de experimentos

R4.4.1. Se genera una matriz de confusión

R4.4.2. Se genera una precisión predictiva

R4.4.3. La biblioteca proporciona una interfaz para añadir nuevas métricas de evaluación de resultados

3.3 Modelado de la biblioteca

Para llevar a cabo esta tarea, se ha elaborado un diagrama de clases de alto nivel. En la Figura 3.2 se puede apreciar el resultado del mismo. Vale la pena aclarar que este diagrama es un primer modelo y no tiene por qué corresponderse con el resultado final.

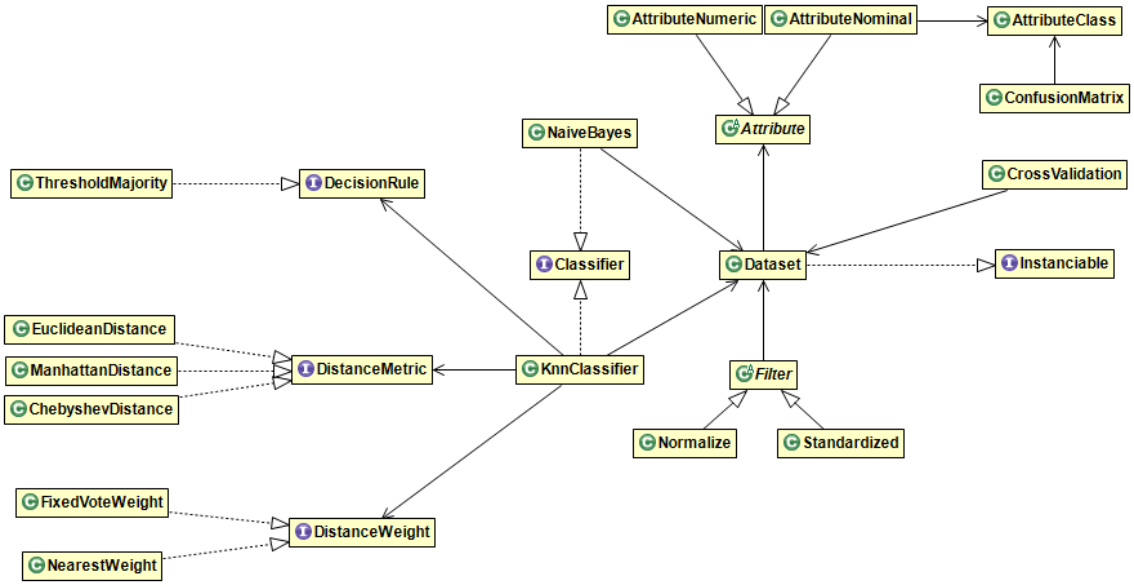


Figura 3.2. Diagrama de clases, JDataMining.

3.4 Proceso de desarrollo.

El proceso de desarrollo siguió las pautas sugeridas por ciclo de vida de software seleccionado. Se han tomado cada uno de los requisitos mencionados anteriormente, pasando por las etapas de análisis, desarrollo, pruebas y evaluación.

La mejor forma de ilustrar el trabajo realizado en la etapa de desarrollo, es siguiendo los diagramas de clases UML presentados a continuación.

3.4.1 Núcleo de la biblioteca.

En este paquete se encuentran todas las clases e interfaces necesarias para hacer un correcto tratamiento de los datos (Figura 3.3), cargando en memoria principal el contenido de ficheros externos, en formato CSV.

Las clases que están encapsuladas en este paquete corresponden con el cumplimiento del primer requisito funcional R1.

3.4.2 Preprocesado de datos.

Empaqueta todas las clases e interfaces para aplicar los respectivos filtros nombrados en el segundo requisito R2. Se contempla el uso de una clase

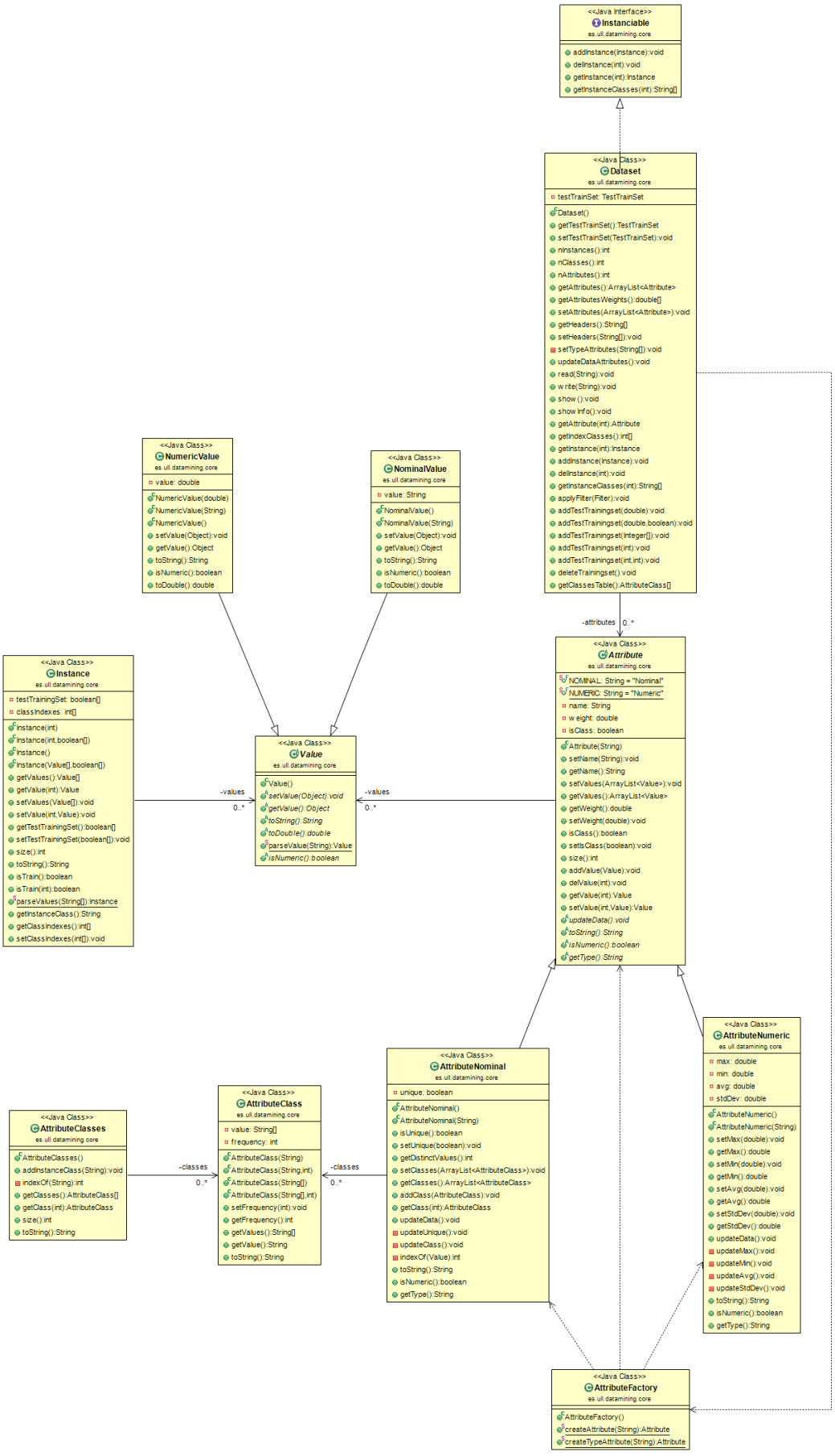


Figura 3.3. Diagrama de clases, núcleo principal

abstracta que sirve de interfaz para la ampliación de la biblioteca (Figura 3.4), de esta forma se pueden añadir otros filtros cumpliendo con el principio abierto/cerrado de SOLID[19].

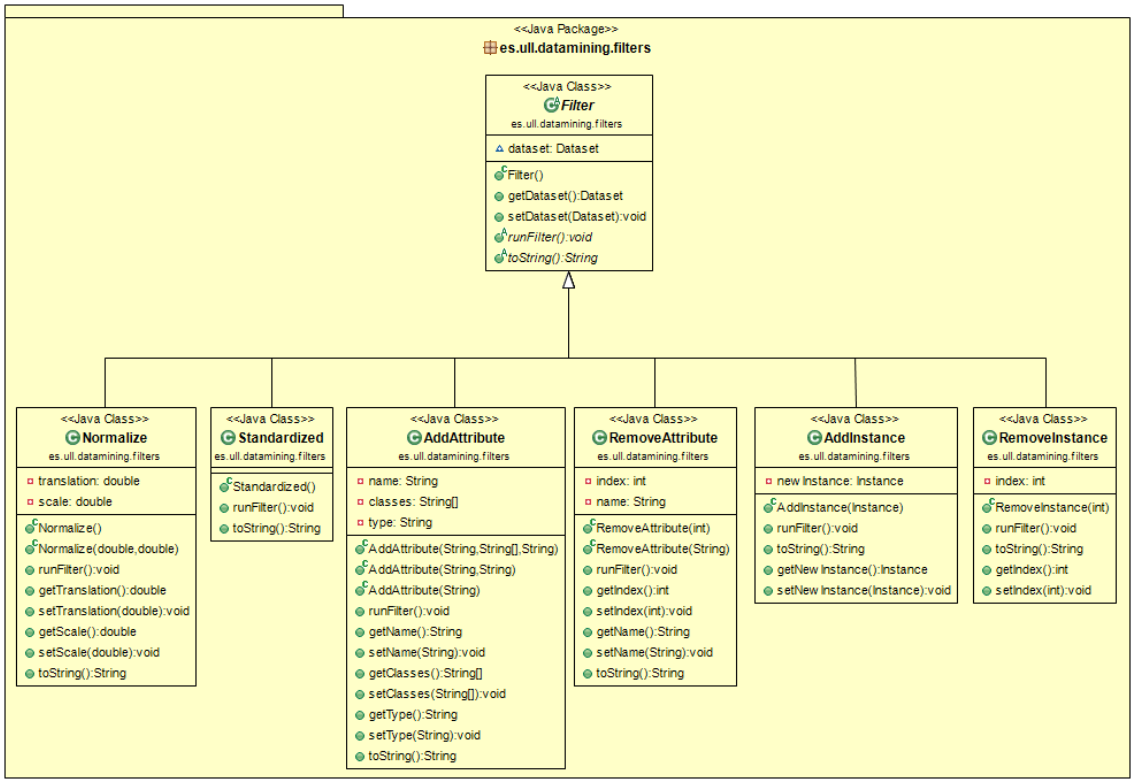


Figura 3.4. Diagrama de clases, filtros.

3.4.3 Clasificadores.

Será necesario más de un diagrama de clases. Este desarrollo supuso el mayor esfuerzo a nivel de implementación. En este caso se recogen todos los requisitos funcionales correspondientes al tercer requisito R3.

Diagrama general (Figura 3.5): Este diagrama de clases muestra las interfaces que permite ampliar la biblioteca ya sea para añadir clasificadores, como para extender nuevas opciones del clasificador Knn.

Se puede observar el uso del patrón de diseño, *Builder Pattern*[20] (`KnnBuilder`), para simplificar la instancia del clasificador Knn.

Diagrama específico (Figura 3.6), este diagrama presenta con más rigor el diseño establecido para las opciones del clasificador Knn. Se ha aplicado el patrón de diseño, *Factory Method*[20] para facilitar la creación tanto de las reglas de decisión (`DesicisionRuleFactory`), como para el pesado de votación (`DistanceWeightFactory`).

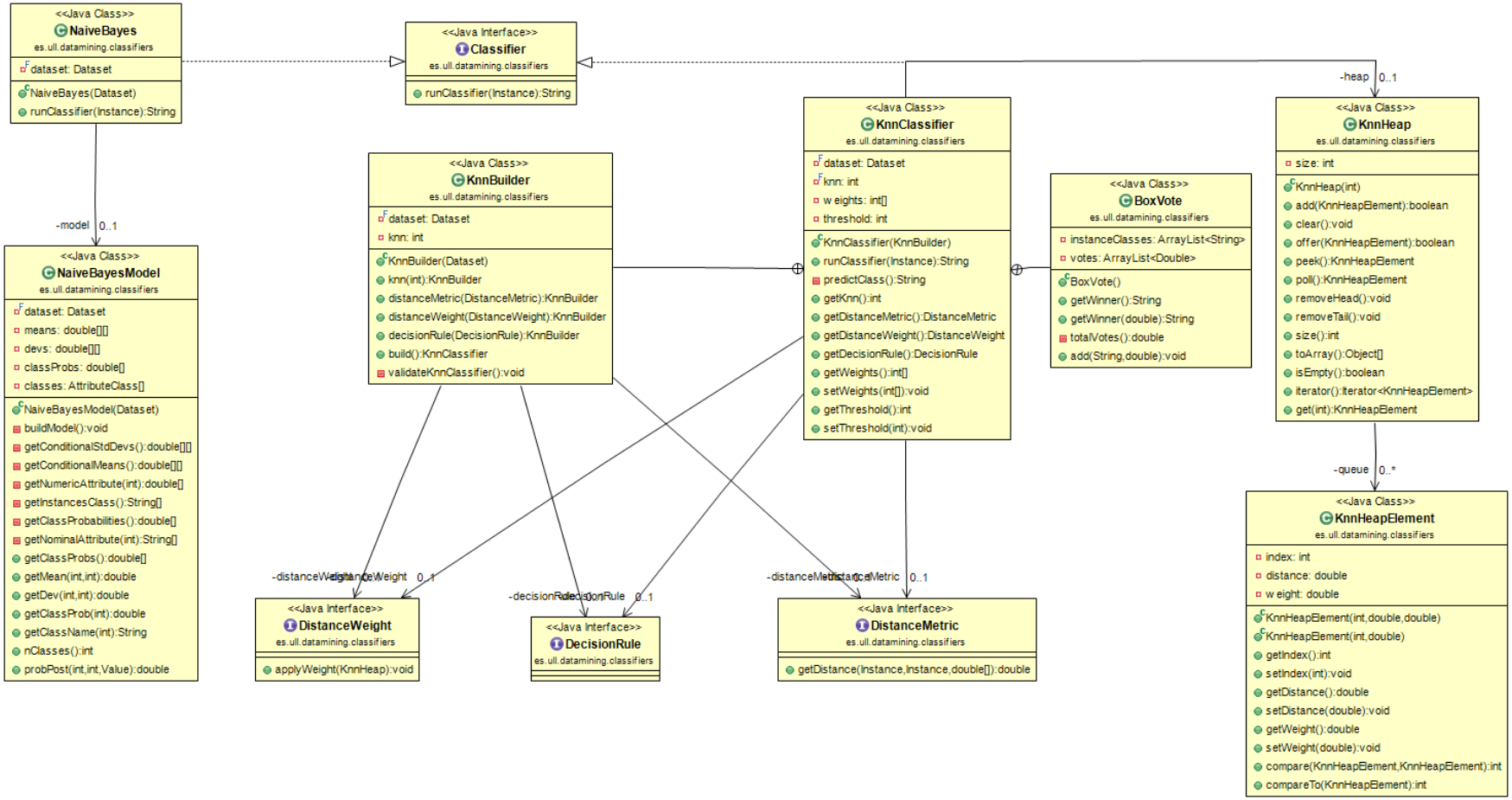


Figura 3.5. Diagrama de clases general de clasificadores.

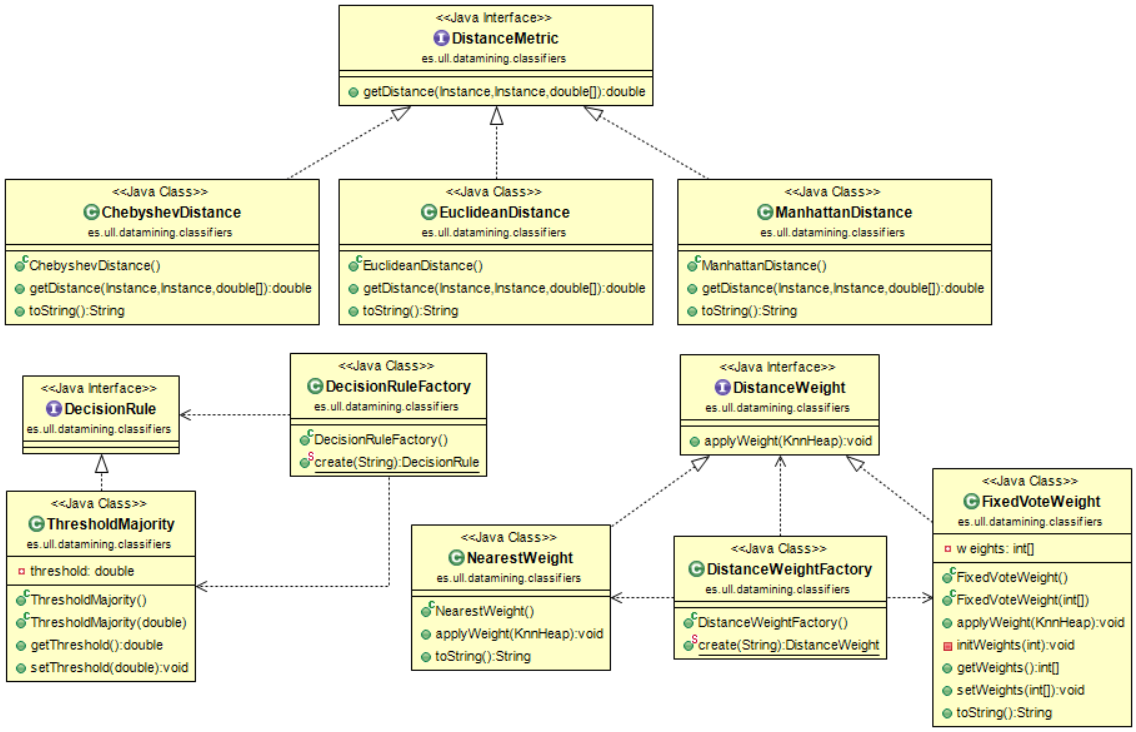


Figura 3.6. Diagrama especifico clasificador Knn.

3.4.4 Fase de experimentación.

En la Figura 3.7 se ilustra el paquete de experimentación y la asociación implícita que presenta con el núcleo principal de la biblioteca. Este paquete muestra las clases que satisfacen el requisito cuatro R4.

3.5 Despliegue: API de la biblioteca.

En esta sección se incluirá una selección de clases, métodos públicos y paquetes de JDataMining que serán los de mayor utilidad para el usuario que quiera hacer procesos de clasificación con la biblioteca.

3.5.1 Paquete: org.ull.jdatamining.core.

El paquete *org.ull.jdatamining.core* contiene todas las clases necesarias para que el usuario pueda cargar, guardar y gestionar los datos en memoria principal.

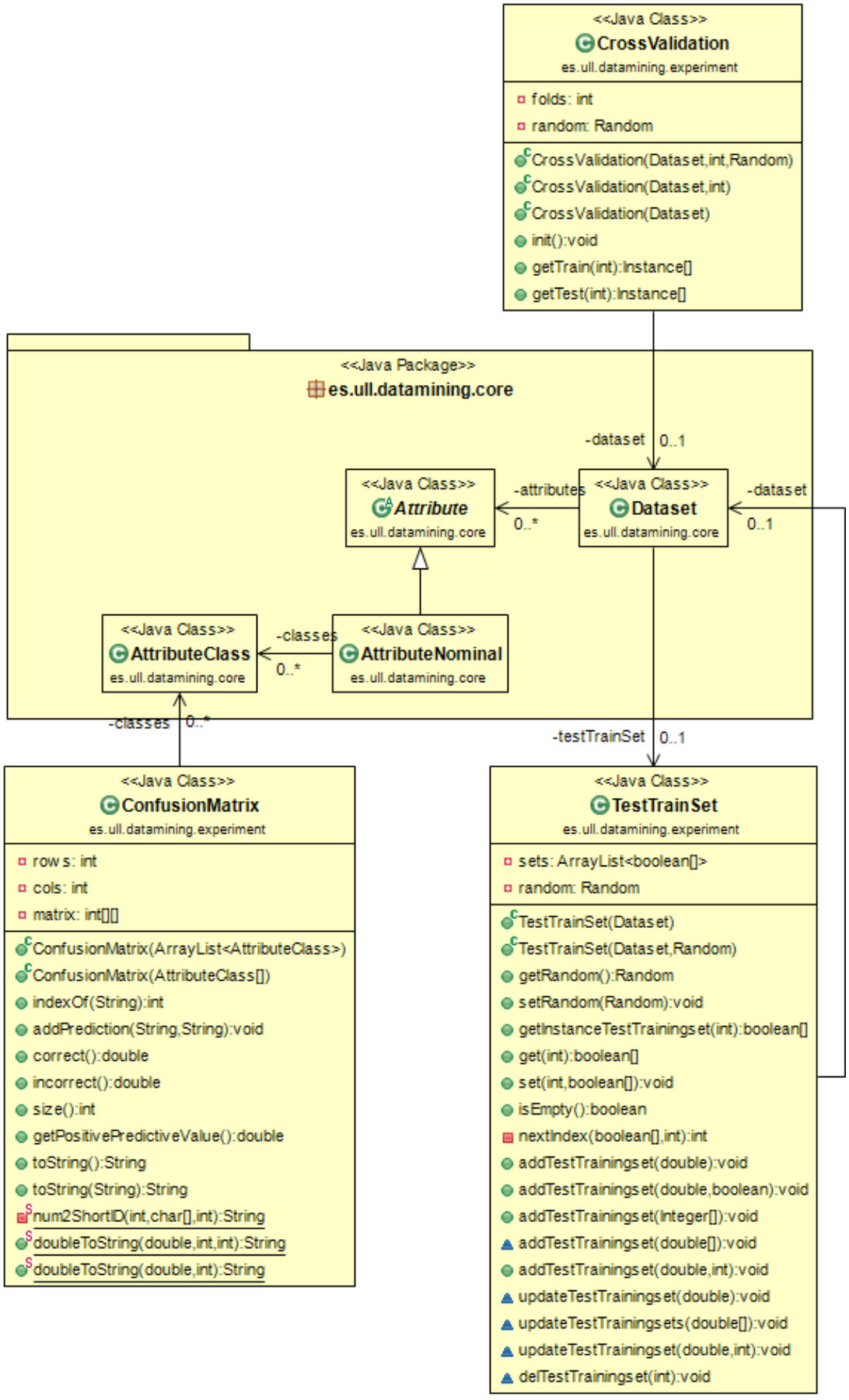


Figura 3.7. Diagrama de clases del paquete de experimentación.

Dataset (`class Dataset implements Instanciable`): Es la representación virtual de los datos cargados en memoria desde un fichero externo.

1. `void read(String fileName)`: Carga un fichero en formato CSV.
   - Parámetros:
     fileName – Nombre del archivo que se será cargado en memoria principal.
   - Lanzadores:
     FileNotFoundException – En caso que no se encuentre el archivo.
     IOException – En caso que el formato del archivo no sea correcto.
   - Ejemplo:

```
//Constructor necesario para el dataset
Dataset dataset = new Dataset();
//bloque try/catch necesario para la lectura de fichero
try {
      dataset.read("glass.csv");
   } catch (FileNotFoundException e) {
      System.err.println("Archivo no encontrado");
   } catch (IOException e) {
      System.err.println("El formato del fichero no corresponde");
}
```

2. `void write(String fileName)`: Carga en un fichero en formato CSV.
   - Parámetros:
     fileName – Nombre del archivo donde escribirá.
   - Ejemplo:

```
//Constructor necesario para el dataset
Dataset dataset = new Dataset();
//...
//Operaciones hechas al dataset
//...
dataset.write("NewGlass.csv");
```

3. `void show()`: Muestra el contenido de las instancias cargadas en memoria.
   - Ejemplo:

```
//Constructor necesario para el dataset
Dataset dataset = new Dataset();
//bloque try/catch necesario para la lectura de fichero
```

```
try {
      dataset.read("glass.csv");
   } catch (FileNotFoundException e) {
      System.err.println("Archivo no encontrado");
   } catch (IOException e) {
      System.err.println("El formato del fichero no corresponde");
}
dataset.show();
```

- Salida:

```
RI Na Mg Al Si K Ca Ba Fe Type
1.51793, 12.79, 3.5, 1.12, 73.03, 0.64, 8.77, 0.0, 0.0, 'build wind float'
1.51643, 12.16, 3.52, 1.35, 72.89, 0.57, 8.53, 0.0, 0.0, 'vehic wind float'
1.51793, 13.21, 3.48, 1.41, 72.64, 0.59, 8.43, 0.0, 0.0, 'build wind float'
1.51299, 14.4, 1.74, 1.54, 74.55, 0.0, 7.59, 0.0, 0.0, tableware
1.53393, 12.3, 0.0, 1.0, 70.16, 0.12, 16.19, 0.0, 0.24, 'build wind non-float'
…
1.51689, 12.67, 2.88, 1.71, 73.21, 0.73, 8.54, 0.0, 0.0, 'build wind non-float'
1.51852, 14.09, 2.19, 1.66, 72.67, 0.0, 9.32, 0.0, 0.0, tableware
```

4. `void showInfo()`: Muestra información de los atributos.

- Ejemplo:

```
//Constructor necesario para el dataset
Dataset dataset = new Dataset();
//bloque try/catch necesario para la lectura de fichero
try {
      dataset.read("glass.csv");
   } catch (FileNotFoundException e) {
      System.err.println("Archivo no encontrado");
   } catch (IOException e) {
      System.err.println("El formato del fichero no corresponde");
}
dataset.showInfo();
```

- Salida:

```
[Name,Weight,IsCLas,Max,Min,Mean,StdDev]
[RI,1.0,false,1.53393,1.51115,1.518365420560748,0.003036863739385434]
[Na,1.0,false,17.38,10.73,13.407850467289727,0.816603555714983]
[Mg,1.0,false,4.49,0.0,2.684532710280374,1.4424078448704418]
[Al,1.0,false,3.5,0.29,1.4449065420560752,0.4992696456004846]
[Si,1.0,false,75.41,69.81,72.65093457943928,0.7745457947651124]
[K,1.0,false,6.21,0.0,0.49705607476635505,0.6521918455589799]
[Ca,1.0,false,16.19,5.43,8.956962616822427,1.423153487281395]
[Ba,1.0,false,3.15,0.0,0.17504672897196266,0.49721926059970234]
[Fe,1.0,false,0.51,0.0,0.05700934579439252,0.09743870063650088]
[Type,1.0,true,false,['build wind float', 'vehic wind float', tableware,
```

```
'build wind non-float', headlamps, containers]]
```

5. `void applyFilter(Filter filter)`: Aplica un filtro al *dataset*.
   - Parámetros:

     filter – Filtro que se quiere aplicar.
   - Ejemplo: En el apartado de filtros se muestra cómo usar este método.
6. `AttributeClass[] getClassesTable()`: Obtiene un vector con cada una de las clases identificadas.
   - Devuelve:

     Un vector con cada una de las clases del *dataset*.
   - Ejemplo: El uso de este método se mostrará en el apartado matriz de confusión.

3.5.2 Paquete: org.ull.jdatamining.filters.

El paquete org.ull.jdatamining.filters, contiene todas las clases necesarias para que el usuario pueda aplicar filtros.

Filtros (`abstract class Filter`): Clase abstracta que representa todos los filtros aplicables a los datos, además sirve de interfaz para la ampliación de la biblioteca.

1. `abstract void runFilter()`: Este método es común para todos los filtros. Es el desencadenante que ejecuta el filtro, desde el método `applyFilter` en la clase `Dataset`.

Normalización (`class Normalize extends Filter`), esta clase aplica a cada una de las instancias la ecuación de la normalización.

1. `void setTranslation(double translation)`: Modifica el valor de la traslación. Por defecto es 0.
   - Parámetros:

     translation – Número real que indica el índice de traslación.

2. `void setScale(double scale)`: Modifica el valor de la escala. Por defecto es 1.
   - Parámetros:

scale – Número real que indica el índice del escalado.

3. `void runFilter()`: Este método desencadena la aplicación de la normalización

Ejemplo: Este código muestra cómo se aplica un filtro a un *dataset*.

```
//Contructor necesario para el dataset
Dataset dataset = new Dataset();
//...
//Constructor del filtro
Filter filter =  new Normalize();
//Modificadores del filtro
((Normalize)filter).setScale(1);
((Normalize)filter).setTranslation(0);
//Aplicacion del filtro al dataset.
dataset.applyFilter(filter);
//En este caso se obtiene instancias con rango [0-1]
```

3.5.3 Paquete: org.ull.jdatamining.classifiers.

El paquete *org.ull.jdatamining.classifiers*, contiene todas las clases para que el usuario añada más clasificadores y pueda usar los que están implementados.

Clasificadores (`interface Classifier`): Representa todos los clasificadores que se pueden aplicar al *dataset*, además sirve como plantilla para la ampliación de la biblioteca.

1. `String runClassifier(Instance instance)`: Este método es común para todos los clasificadores. Es el desencadenante del clasificador, desde el método `applyFilter` en la clase `Dataset`.

- Parámetros:

  instance – Objeto instancia a que se aplica el clasificador.

- Devuelve:

  La clasificación requerida en cadena de caracteres.

Clasificador Knn (`class KnnClassifier implements Classifier`): Aplica el método de clasificación Knn a un *dataset*.

1. `String runClassifier(Instance instance)`: Este método desencadena la aplicación del clasificador Knn.

- Parámetros:

instance – Objeto instancia a que se aplica el clasificador.

- Devuelve:

  La clasificación requerida en cadena de caracteres.

Calculo de distancia (`interface DistanceMetric`): Representa todas las distancias que se pueden aplicar al clasificador Knn, además sirve como plantilla para la ampliación de la biblioteca.

1. double getDistance(Instance instance1, Instance instance2, double[] weights)

- Parámetros:

  instance1 – Primer instancia para el cálculo de distancia.
  instance2 – Segunda instancia para el cálculo de distancia.
  weights – Peso asignado a cada uno de los atributos.

- Devuelve:

  Un número real con el cálculo de la distancia.

Peso de distancia (`interface DistanceWeight`): Representa el tipo de pesado que se pueden aplicar al clasificador Knn. Se usa como plantilla para la ampliación de la biblioteca.

1. `void apply Weight(KnnHeap heap)`: Este método es común para los tipos de pesado. Aplica un determinado tipo de pesado a la estructura interna auxiliar del clasificador Knn.

- Parámetros:

  heap – Corresponde a la estructura interna del clasificador Knn el cual contiene los k vecinos más cercanos.

Regla de decisión (`interface DecisionRule`), representa las reglas de decisión que se pueden aplicar al clasificador Knn, se usa como plantilla para la ampliación de la biblioteca.

Mayoría con umbral (`class ThresholdMajority implements DecisionRule`), permite aplicar esta regla de decisión.

1. `ThresholdMajority(double threshold)`: Este constructor permite establecer el porcentaje del umbral permitido. El número va de 0 a 1. Si el parámetro es vacío, por defecto se dejara un umbral de 0.5.

- Parámetros:

  threshold – Número real que indica el porcentaje del umbral.

Ejemplo: Este código muestra cómo aplicar el clasificador Knn a un *dataset*.

```
//Contructor necesario para el dataset
Dataset dataset = new Dataset();
//...
//Se instancia la Distancia Manhattan como metrica principal.
DistanceMetric distanceMetric = new ManhattanDistance();
//Se instancia como opcion de peso en el voto, la distancia más corta.
DistanceWeight distanceWeight = new NearestWeight();
//Se instancia como regla de desicion, mayoria con umbral siendo el umbral del
50%.
DecisionRule decisionRule = new ThresholdMajority(0.5);
//Instanciacion del clasificador usando Builder pattern.
Classifier classifier = new KnnClassifier.KnnBuilder(dataset)
    //Se asigna un 3 para los k vencinos.
    .Knn(3)
    //Se asigna la distancia ya instanciada.
    .distanceMetric(distanceMetric)
    //Se asigna el peso ya instanciado.
    .distanceWeight(distanceWeight)
    //Se asigna la regla de decision ya instanciada.
    .decisionRule(decisionRule)
    //Se procede a contruir el clasificador.
    .build();
for (int i = 0; i < dataset.nInstances(); i++) {
    //Se aplica el clasificador a cada una de las instancias del dataset
    System.out.println(classifier.runClassifier(dataset.getInstance(i)));
}
//..
```

Clasificador Naive Bayes (`class NaiveBayes implements Classifier`), aplica el método de clasificación Naive Bayes a un *dataset*.

1. `String runClassifier(Instance instance)`: Desencadena la aplicación del clasificador Naive Bayes.

Ejemplo: Este código muestra cómo aplicar el clasificador Naive Bayes a un *dataset*.

```
//Contructor necesario para el dataset
Dataset dataset = new Dataset();
//...
```

```
//Instanciacion del clasificador.
Classifier classifier = new NaiveBayes(dataset);
for (int i = 0; i < dataset.nInstances(); i++) {
    //Se aplica el clasificador a cada una de las instancias del dataset
    System.out.println(classifier.runClassifier(dataset.getInstance(i)));
}
//..
```

3.5.4 Paquete: org.ull.jdatamining.experiments.

El paquete *org.ull.jdatamining.experiments*, contiene las clases necesarias para que el usuario pueda generar experimentos a partir de un *dataset*, aplicando los clasificadores anteriormente mencionados.

Validación cruzada (`class CrossValidation`): Proporciona los métodos necesarios para generar un experimento.

1. `CrossValidation(Dataset dataset, int folds, Random random)`: Este constructor altera cada uno de los atributos de la clase.

- Parámetros:

  dataset – Estructura de datos a la que se le aplica la segmentación. Este atributo es obligatorio.
  folds – Número de pliegue que se desea usar en el experimento. Por defecto es 10.
  random – Indica si se desea una prueba aleatoria controlada. Por defecto hace pruebas nos controladas.

2. `void init()`: Desencadena la aplicación de la validación cruzada.

Ejemplo: Este código muestra cómo aplicar una validación cruzada.

```
//Contructor necesario para el dataset
Dataset dataset = new Dataset();
//...
//Instanciacion la validación cruzada de un solo pliegue.
CrossValidation crossValidation = new CrossValidation(dataset,1);
//Aplicar la validación cruzada al dataset.
crossValidation.init();
//Con este segmento comprobamos que instancias quedaron en el conjunto de
entrenamiento.
for (int i = 0; i < dataset.nInstances(); i++) {
            Instance instance = dataset.getInstance(i);
            for (int j = 0; j < 10; j++) {
                //Comprueba si la instancia pertenece al conjunto de
```

```
entrenamiento.
                if (instance.isTrain(j))
                    //Si pertenece al conjunto de entrenamiento la instancia
se imprime por consola.
                    System.out.println(instance.toString());
            }
}
//..
```

Matriz de confusión (`class ConfusionMatrix`): Proporciona los métodos necesarios para construir una matriz de confusión a partir de un experimento.

1. `ConfusionMatrix(AttributeClass[] classes)`: Constructor que necesita las distintas clases de atributo de un *dataset*.

- Parámetros:

  classes – tablas con todas las clases de atributo.

2. `void addPrediction(String predict, String prediction)`: Añade las predicciones hechas por el experimento para ir construyendo la matriz.

- Parámetros:

  predict – clase actual.
  prediction – clase que ha sido predicha.

3. `double getPositivePredictiveValue()`: Este método obtiene la precisión predictiva del experimento.

- Devuelve:

  La precisión predictiva de un experimento.

Ejemplo: Este código enseña cómo mostrar una matriz de confusión y la precisión predictiva.

```
//Constructor necesario para el dataset
Dataset dataset = new Dataset();
//...
//Instancia de la matriz de confusion.
ConfusionMatrix confusionMatrix = new
ConfusionMatrix(dataset.getClassesTable());
//Bloque de ejecución de un clasificador.
for (int i = 0; i < dataset.nInstances(); i++) {
    //Obtención de la instancia a predecir
    String predict = dataset.getInstance(i).getInstanceClass();
    //Obtención de la instancia predicha.
    String prediction = classifier.runClassifier(dataset.getInstance(i));
    //Se añade el experimento a la matriz de confusion.
    confusionMatrix.addPrediction(predict, prediction);
}
```

```
//Muesta la matriz de confusión.
System.out.println(confusionMatrix.toString());
//Muestra la precisión predictiva.
System.out.println("Precision predictiva:
"+confusionMatrix.getPositivePredictiveValue());
//..
```

Salida:

```
=== Confusion Matrix ===

  a  b  c  d  e  f     actual class
 64  3  0  3  0  0 |  a = ['build wind float']
  6 11  0  0  0  0 |  b = ['vehic wind float']
  0  0  7  1  1  0 |  c = [tableware]
 10  0  1 64  0  1 |  d = ['build wind non-float']
  0  0  1  1 27  0 |  e = [headlamps]
  0  0  0  0  1 12 |  f = [containers]

Precision predictiva: 0.8644859813084113
```

3.5.5 Paquete: org.ull.jdatamining.util.

El paquete org.ull.jdatamining.util, contiene una serie de métodos útiles para cálculos estadísticos, cargar y guardar ficheros con el menú grafico de SWING, etc.

Utilidades (`class Util`): Contiene una serie de métodos para hacer pruebas y manejar el *dataset*.

1. `static String getOpenFileName()`: Abre un cuadro de dialogo de SWING preguntando por un fichero para abrir.

- Devuelve:

  La ruta del fichero que se desea abrir.

2. `static String getSaveFileName()`: Abre un cuadro de dialogo de SWING preguntando por un fichero para guardar.

- Devuelve:

  La ruta del fichero que se desea guardar.

3. `static boolean isNumeric(String string)`: Permite al usuario saber si una cadena de texto es de números.

- Parámetros:

  string – cadena que se desea comprobar.

- Devuelve:

Verdadero si la cadena solo tiene números.

4. `static int getIndexMax(double[] vector)`: Permite al usuario saber el índice del valor máximo en un vector se números reales.

- Parámetros:

  vector – vector de números reales a comprobar.

- Devuelve:

  El índice del valor máximo del vector.

5. `static void normalizeVector(double[] vector)`: Permite al usuario normalizar un vector de números reales. Correspondiente con la frecuencia relativa estadística.

- Parámetros:

  vector – vector de números reales a normalizar.

Probabilidad (`class Probability`): Contiene una serie de métodos útiles para hacer cálculos estadísticos orientados a probabilidades condicionadas.

1. `static double normalDens(double x, double mean, double stdDev)`: Calcula la función de densidad de una distribución normal.

- Parámetros:

  x – punto de la probabilidad a calcular.
  mean – media de la distribución.
  stdDev – desviación típica de la distribución.

- Devuelve:

  El cálculo de la densidad.

2. `static double conditionalProbability(String[] values, String[] instancesClass, String value, String target)`: Calcula la probabilidad condicionada.

- Parámetros:

  values – conjunto de valores.
  instancesClass – clases de la instancia.
  value – valor actual.

target – valor condicionado.

- Devuelve:

  La probabilidad condicionada de las clases de instancia.

3. `static double condProbLaplaceSmooth(String[] values, String[] instancesClass, String value, String target, int nAttributeClass)`: Calcula la probabilidad condicionada, aplicando el suavizado de Laplace.

- Parámetros:

  values – conjunto de valores.
  instancesClass – clases de la instancia.
  value – valor actual.
  target – valor condicionado.
  nAttributeClass – Numero de las clases de atributos.

- Devuelve:

  La probabilidad condicionada de las clases de instancia aplicando el suavizado de Laplace.

4. `static double conditionalMean(double[] values, String[] instancesClass, String target)`: Calcula la media condicionada.

- Parámetros:

  values – conjunto de valores.
  instancesClass – clases de la instancia.
  target – valor condicionado.

- Devuelve:

  La media condicionada de las clases de instancia.

5. `static double conditionalStandardDeviation(double[] values, String[] instancesClass, String target, double mean)`: Calcula la desviación típica condicionada.

- Parámetros:

  values – conjunto de valores.
  instancesClass – clases de la instancia.
  target – valor condicionado.

mean – media condicionada.

- Devuelve:

  La desviación típica condicionada de las clases de instancia.

Capítulo 4. Descripción de la aplicación de escritorio.

4.1 Requisitos funcionales del entorno gráfico.

Los requisitos funcionales que debe cumplir la interfaz gráfica de usuario son:

R5. Se proporcionara un entorno grafico para comprobar el funcionamiento de la biblioteca JDataMining

R5.1. En la pantalla de bienvenida se podrá elegir un modo de análisis

R5.1.1. El modo análisis tendrá una pestaña para la exploración de los datos

R5.1.1.1. En la pestaña de exploración se podrá cargar un *dataset* desde un fichero CSV

R5.1.1.2. En la pestaña de exploración se podrá guardar un *dataset* en un fichero CSV

R5.1.1.3. En la pestaña de exploración se podrá ver el contenido de un *dataset*

R5.1.1.4. En la pestaña de exploración se podrá elegir y aplicar diferentes filtros

R5.1.1.4.1. Los filtros se podrán configurar según su necesidad.

R5.1.1.5. En la pestaña de exploración se observara la información de los atributos cargados

R5.1.1.6. En la pestaña de exploración se podrá seleccionar qué atributos son una clase

R5.1.1.7. En la pestaña de exploración se asignará un peso determinado a los atributos. Por defecto el peso es 1

R5.1.1.8. En la pestaña de exploración se observará datos estadísticos de los atributos

R5.1.2. El modo análisis tendrá una pestaña para hacer pruebas con los clasificadores

R5.1.2.1. En la pestaña de pruebas se podrá gestionar un conjunto de pruebas

R5.1.2.1.1. El conjunto de pruebas se podrá cargar desde un fichero en formato CSV

R5.1.2.1.2. El conjunto de pruebas se podrá crear de forma manual

R5.1.2.1.3. El conjunto de pruebas se podrá modificar de forma manual

R5.1.2.1.4. El conjunto de pruebas se podrá eliminar de forma manual

R5.1.2.2. En la pestaña de pruebas se podrá seleccionar y aplicar los clasificadores

R5.1.2.2.1. A los clasificadores se les podrá cambiar su configuración.

R5.1.3. El modo análisis tendrá una pestaña para ver los resultados de las pruebas

En la pestaña de resultados se podrá ver:

R5.1.3.1. Ruta del *dataset* cargado

R5.1.3.2. Pesos de los atributos

R5.1.3.3. Los filtros aplicados

R5.1.3.4. Clases del *dataset*

R5.1.3.5. Ruta del conjunto de pruebas

R5.1.3.6. Clasificador seleccionado

R5.1.3.6.1. Opciones del clasificador

R5.1.3.7. Todas las instancias de prueba con su respectiva predicción

R5.2. En la pantalla de bienvenida se podrá elegir un modo experimentación

R5.2.1. En el modo experimentación se tendrá una pestaña para configurar el experimento

R5.2.1.1. Se podrá crear un experimento nuevo

R5.2.1.1.1. Se podrá cargar un *dataset*

R5.2.1.1.2. Se podrá ver el *dataset* cargado

R5.2.1.1.3. Se podrá elegir una experimentación con validación cruzada

R5.2.1.1.3.1 Se podrá asignar el número de pliegues que se desean crear

R5.2.1.1.4. Se podrá elegir una experimentación especificando el porcentaje de los conjuntos

R5.2.1.1.4.1 Se podrá elegir la segmentación de forma aleatoria

R5.2.1.1.4.2 Se podrá elegir la segmentación de forma secuencial

R5.2.1.1.5. En caso de hacer un experimento con segmentación aleatoria se podrá asignar una semilla

R5.2.1.2. Se podrá cargar un experimento ya creado para repetirlo

R5.2.1.2.1. Se podrá cargar un conjunto de entrenamiento creado

R5.2.1.2.1.1 Se podrá ver un conjunto de entrenamiento cargado.

R5.2.1.2.2. Se podrá cargar un conjunto de pruebas creado

R5.2.1.2.2.1 Se podrá ver un conjunto de pruebas cargado

R5.2.1.3. Se podrá seleccionar y aplicar los clasificadores

R5.2.1.3.1. Se podrá modificar las opciones si procede.

R5.2.2. En el modo experimentación se tendrá una pestaña para ver los resultados del experimento

En la pestaña de resultados se podrá ver:

R5.2.2.1. Las opciones seleccionadas del experimento

R5.2.2.2. La matriz de confusión del experimento

R5.2.2.3. La precisión predictiva del experimento

4.2 Diagramas de casos de uso.

Para lograr una comprensión grafica de los requisitos anteriormente presentados, se usaron los diagramas de casos de uso.

4.2.1 Caso de uso 1.

En este diagrama se contemplan los requisitos R5.1 y R5.2. El usuario que interactúa con el sistema puede elegir entre hacer un análisis de datos o hacer un experimento de clasificación.

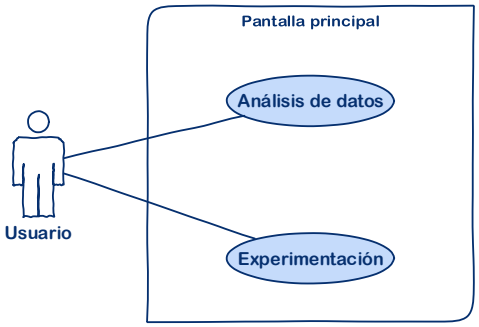


Figura 4.1. Diagrama de caso de uso 1: Pantalla principal.

4.2.2 Caso de uso 2.

El usuario elije la opción de hacer un análisis de datos. El sistema le presentará tres pestañas distintas que le servirán para llevar a cabo su tarea de análisis. Este caso de uso se corresponde con los requisitos R5.1.1, R5.1.2 y R5.1.3.

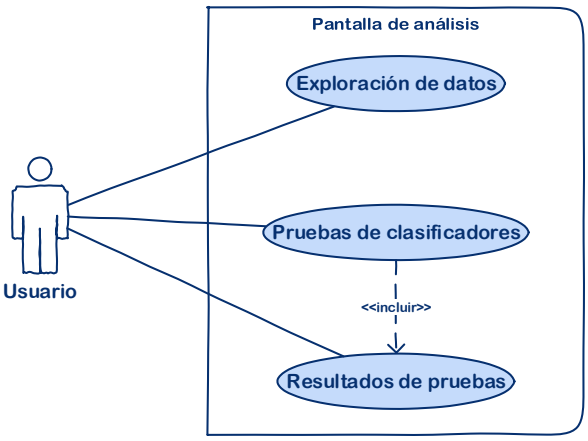


Figura 4.2. Diagrama de caso de uso 2: Pantalla de análisis.

4.2.3 Caso de uso 3.

Hace referencia a los requisitos del R5.1.1.1 al R5.1.1.8. En el diagrama se puede distinguir las opciones que el usuario puede emplear, al querer hacer una exploración del *dataset*.

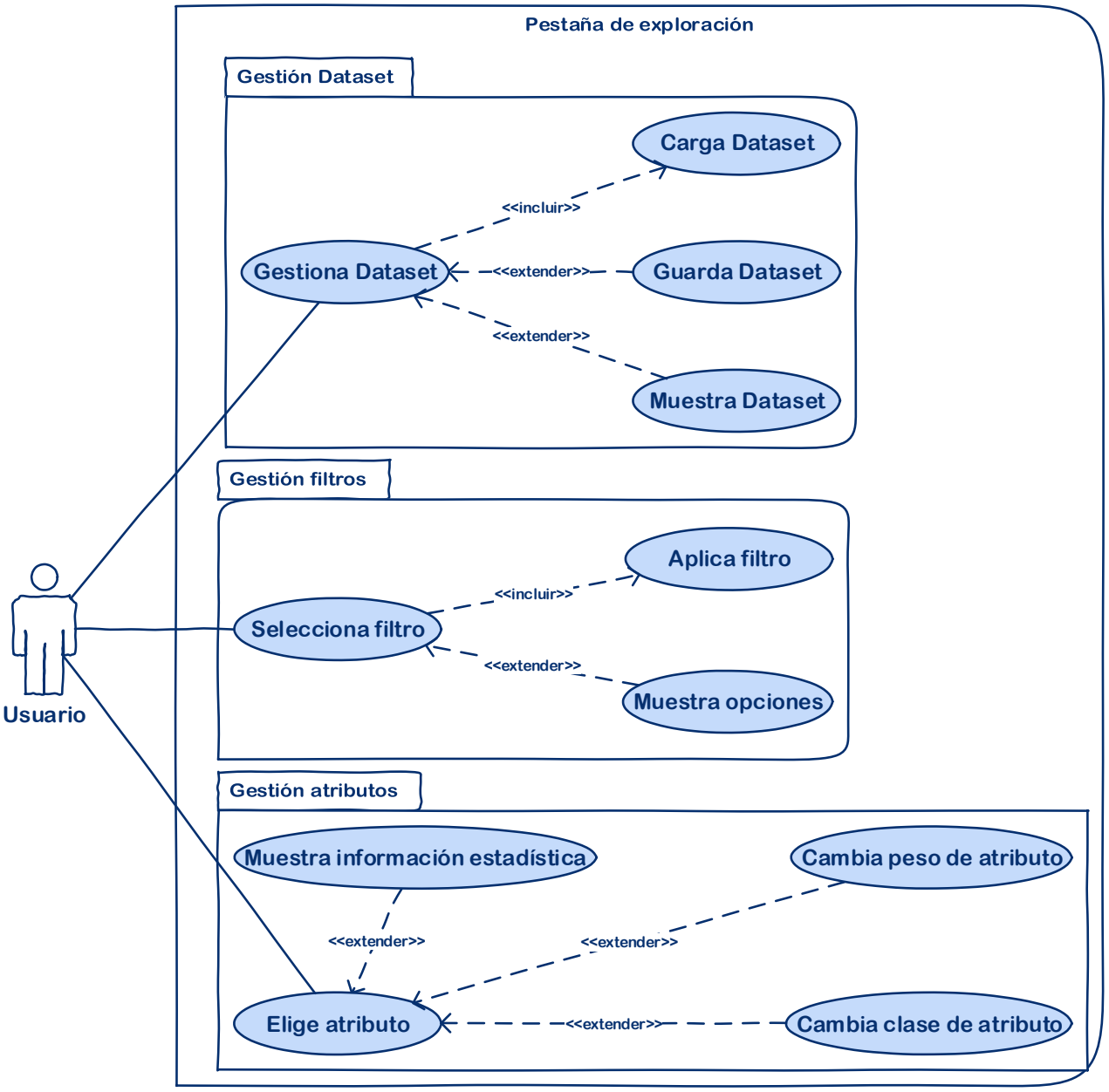


Figura 4.3. Diagrama de caso de uso 3: Pestaña de exploración.

4.2.4 Caso de uso 4.

Se presenta al usuario las formas de hacer pruebas a los clasificadores de la aplicación, correspondiendo con los requisitos R5.1.2.1 y R5.1.2.2 y sus respectivos subapartados.

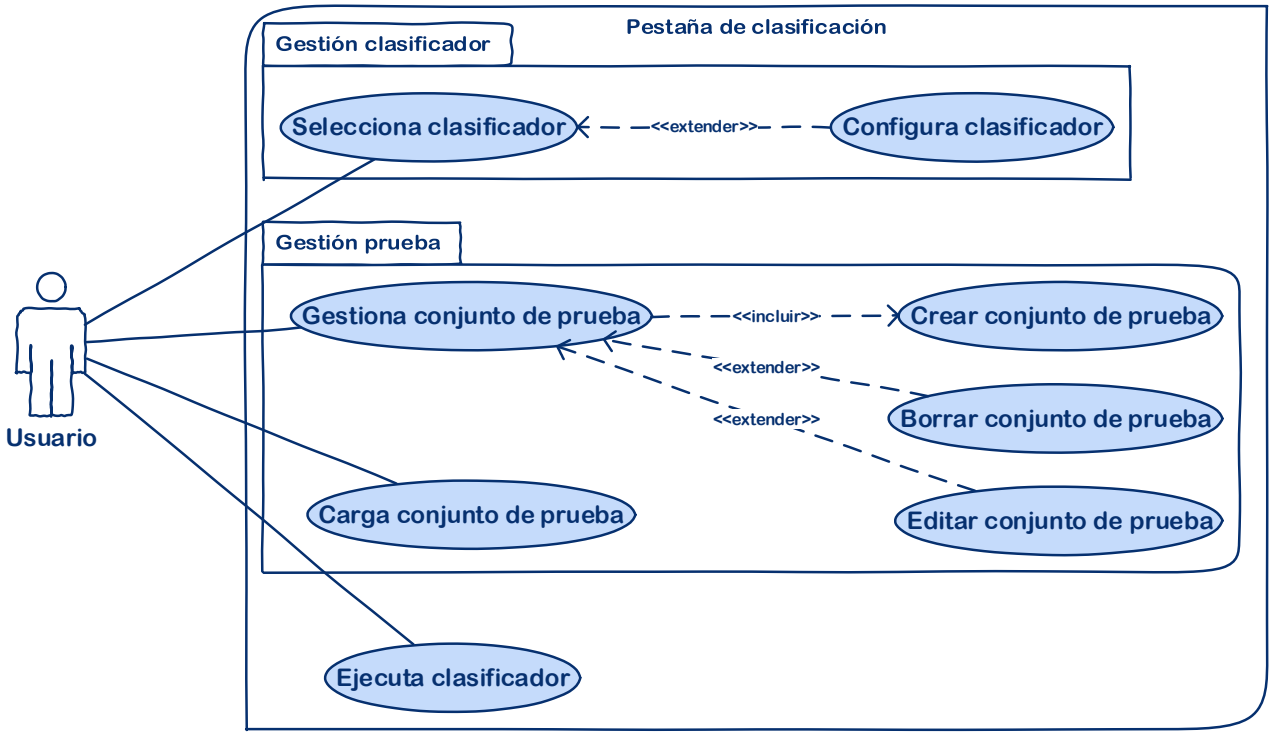


Figura 4.4. Diagrama de caso de uso 4: Pestaña de clasificación.

4.2.5 Caso de uso 5.

El sistema mostrará una pestaña de configuración y otra pestaña donde se observan los resultados del experimento. Este caso de uso se corresponde con los requisitos R5.2.1 y R5.2.2.

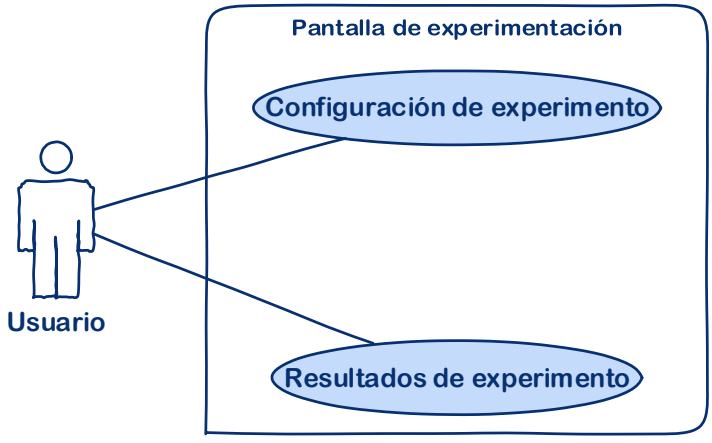


Figura 4.5. Diagrama de caso de uso 5: Pantalla de experimentación.

4.2.6 Caso de uso 6.

Expone los requisitos correspondientes con los numerales R5.2.1.1, R5.2.1.2 y R5.2.1.3 e incluye los subapartados de cada uno de ellos. Representa la situación a la que el usuario se enfrenta, cuando quiere hacer un experimento.

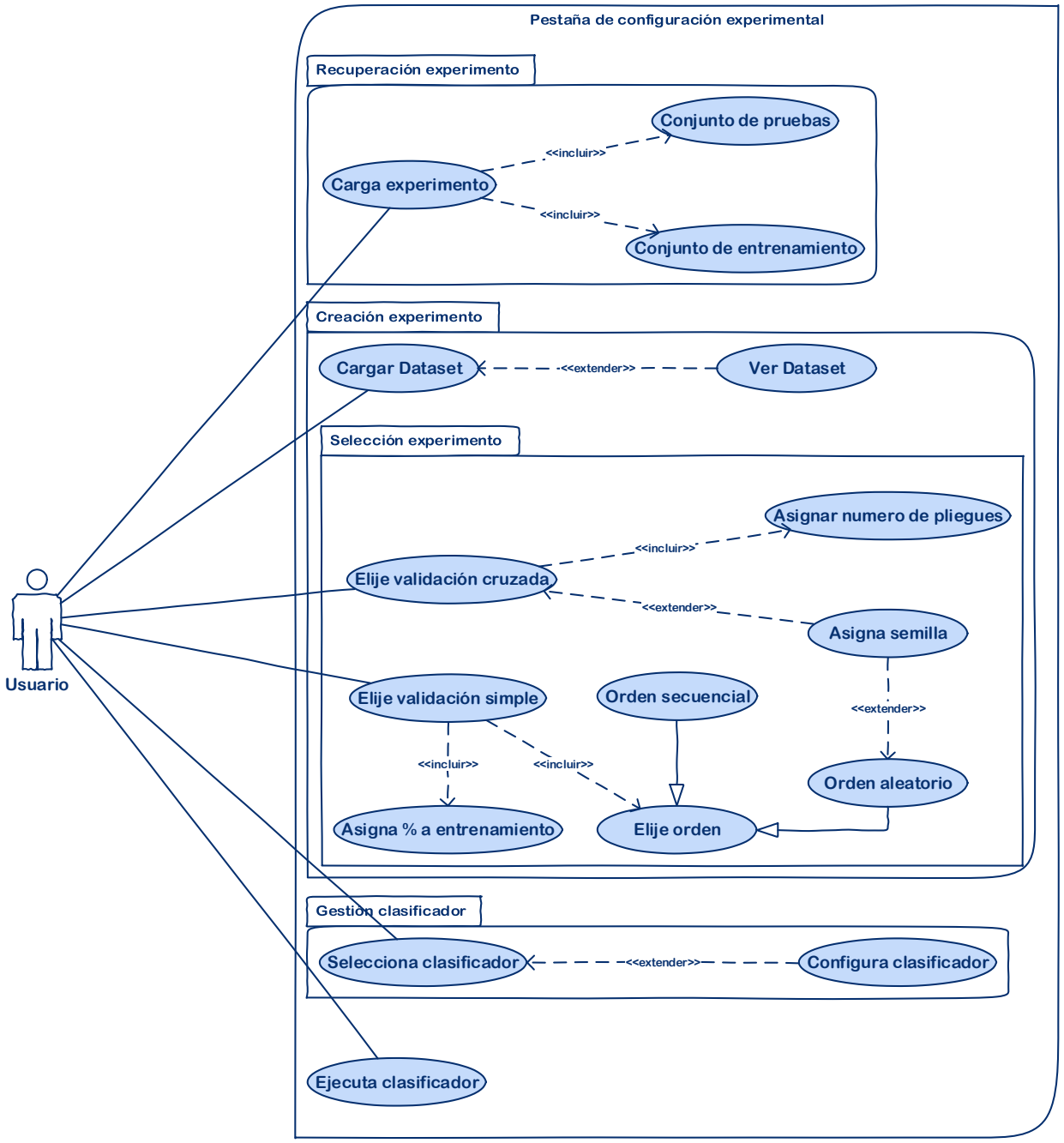


Figura 4.6. Diagrama de caso de uso 6: Configuración del experimento.

4.3 Presentación de la interfaz gráfica.

Este apartado muestra algunas capturas del resultado final de la interfaz gráfica y explica los controles de mayor influencia en la aplicación.

4.3.1 Pantalla de inicio.

El botón *"Analysis"* (opción 1) nos muestra una ventana nueva donde se hace el análisis de los datos. El botón *"Experimenter"* (opción 2) nos muestra una ventana donde se generar un experimento de clasificación. El botón *"About"* (opción 3) nos muestra la ficha técnica de la aplicación.

Figura 4.7. Pantalla de inicio.

4.3.2 Pantalla análisis de datos: Exploración.

Esta es la vista general de la pestaña de exploración en la ventana de análisis. En la sección *"Dataset"* (1) están todos los controles referentes a la gestión de entrada/salida de ficheros. En la sección *"Filters"* (2) se puede seleccionar los filtros que se desea aplicar al *dataset*. En la sección *"Attributes"* (3) se muestra una lista de todos los atributos que contiene el *dataset*. En la sección *"Info Attributes"* (4) se obtiene información más detallada de los atributos seleccionados en la sección de *"Attributes"*.

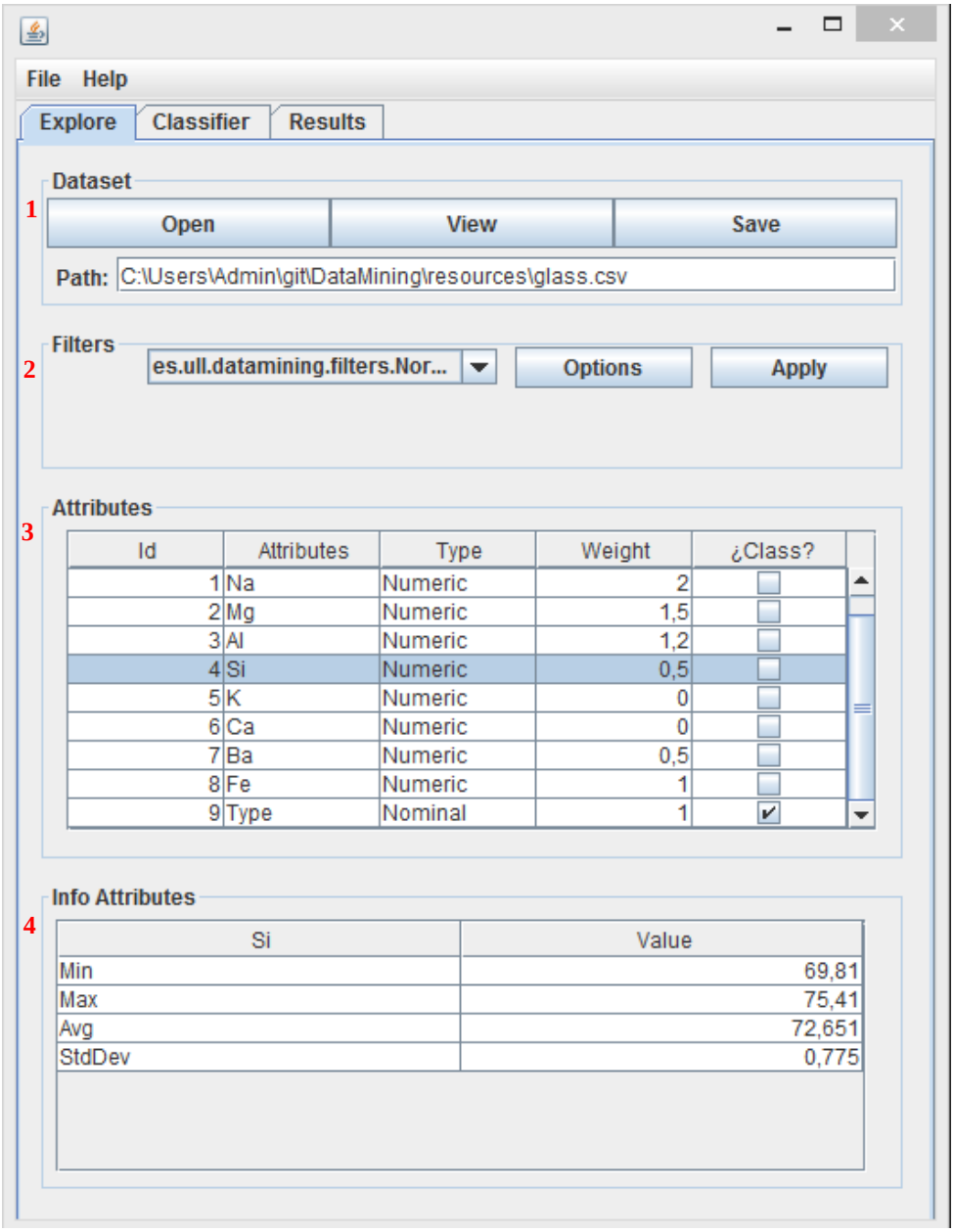


Figura 4.8. Pantalla análisis de datos: Exploración.

4.3.3 Pantalla análisis de datos: Clasificadores.

Esta es la vista general de los clasificadores de la ventana de análisis donde se comprueba el funcionamiento de los clasificadores. En la sección *"Prediction Set"* (1) se hace el manejo de un conjunto de pruebas generado por el usuario. En la sección de *"Select Classifier"* (2) se selecciona el clasificador que se desea probar. En la sección de *"Configuration"* (3) alteramos los valores por defecto del clasificador, seleccionado previamente en

el la sección 2; la configuración se muestra o no, según el comportamiento del clasificador. La sección 4 muestra los controles de ejecución.

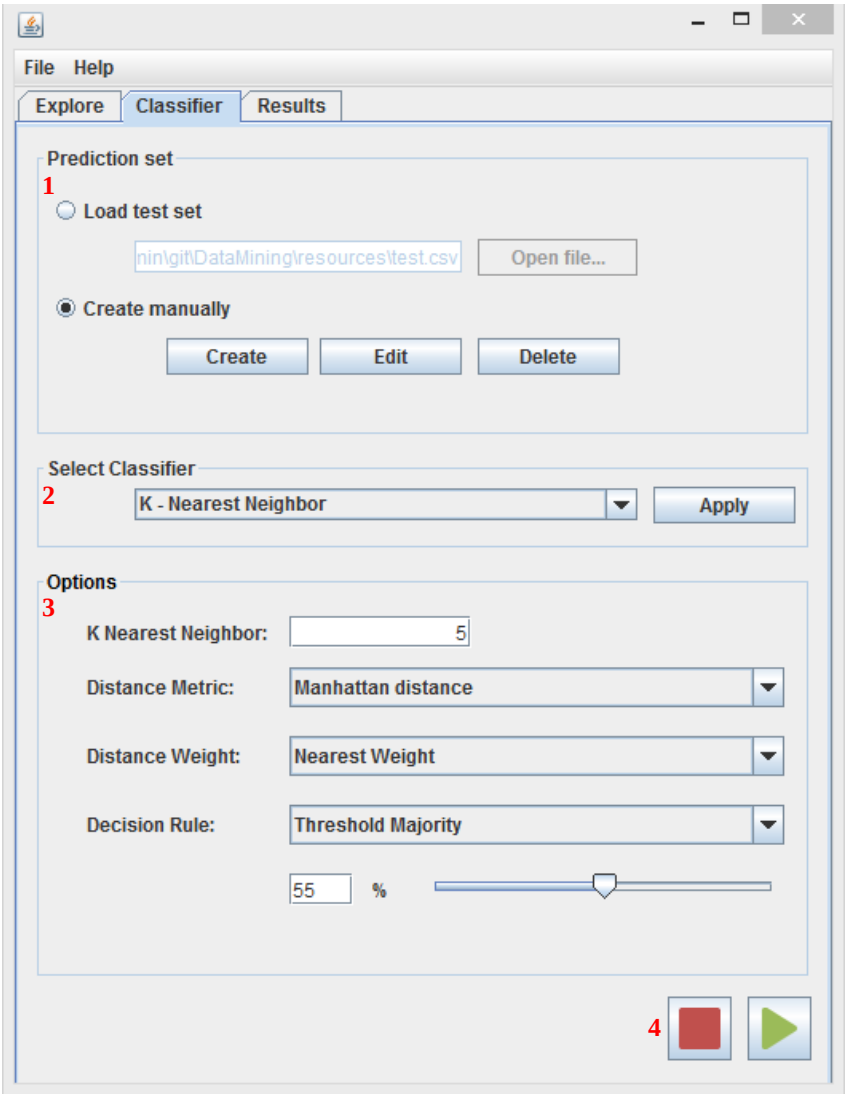


Figura 4.9. Pantalla análisis de datos: Clasificadores.

4.3.4 Pantalla análisis de datos: Resultados.

Esta pestaña muestra cada una de las opciones en el análisis de datos y por ultimo (enmarcado en rojo) muestra el resultado de las pruebas hechas a los clasificadores.

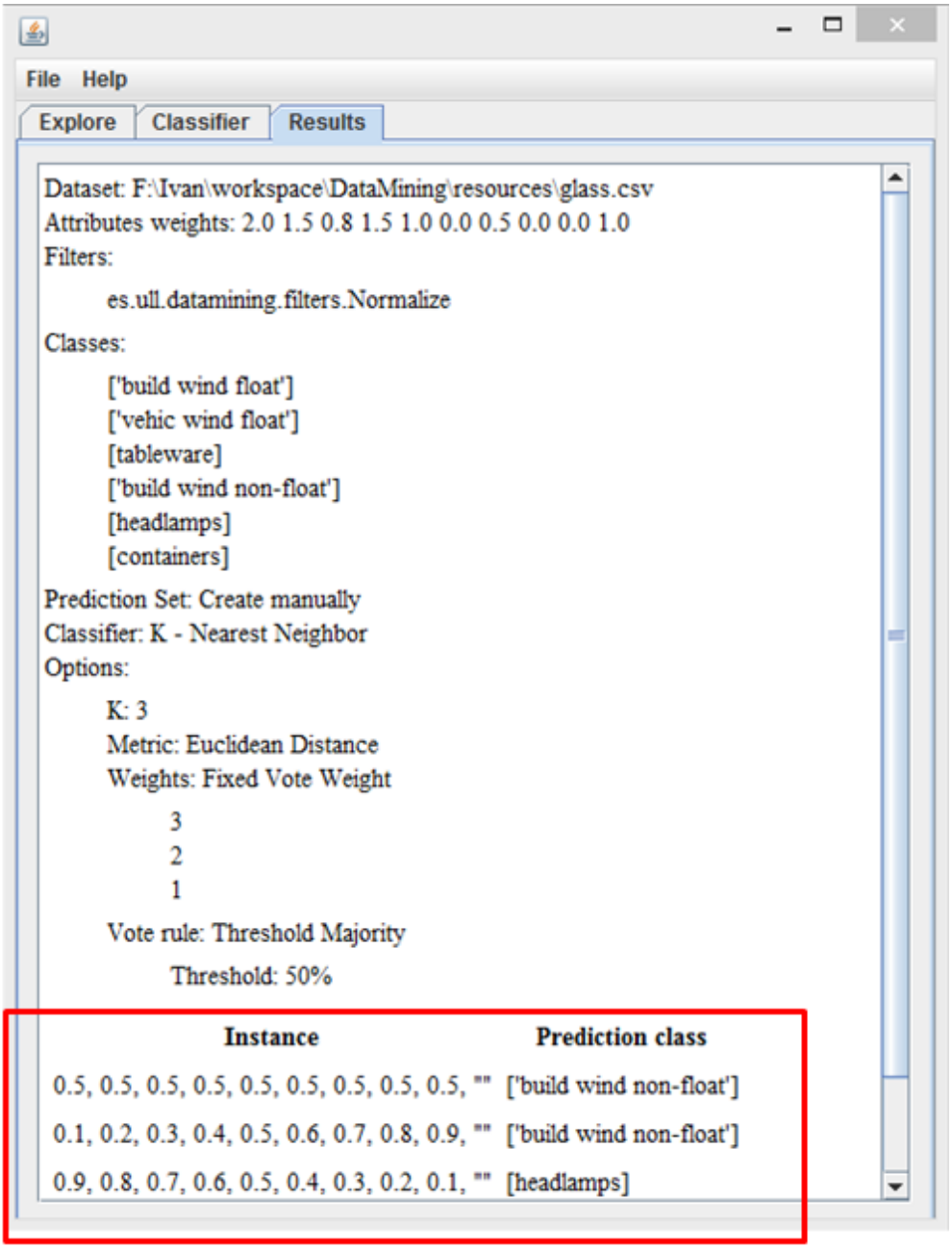


Figura 4.10. Pantalla análisis de datos: Resultados.

4.3.5 Pantalla experimentación: Configuración.

Esta es la vista general de la pestaña de configuración en la ventana de experimentación. En la primera sección (1) se elige entre un experimento nuevo o se cargar un experimento ya creado. En la sección *"New Train/Test Set"* (2) se modifica la configuración para crear nuevos conjuntos de entrenamiento y pruebas. En la sección *"Select Classifier"* (3) seleccionamos el clasificador que usamos en el experimento. El botón *"Generate"* (4) genera

el experimento; si se ha seleccionado la opción *"Train/Test percentage split"* arroja un mensaje permitiendo guardarlo.

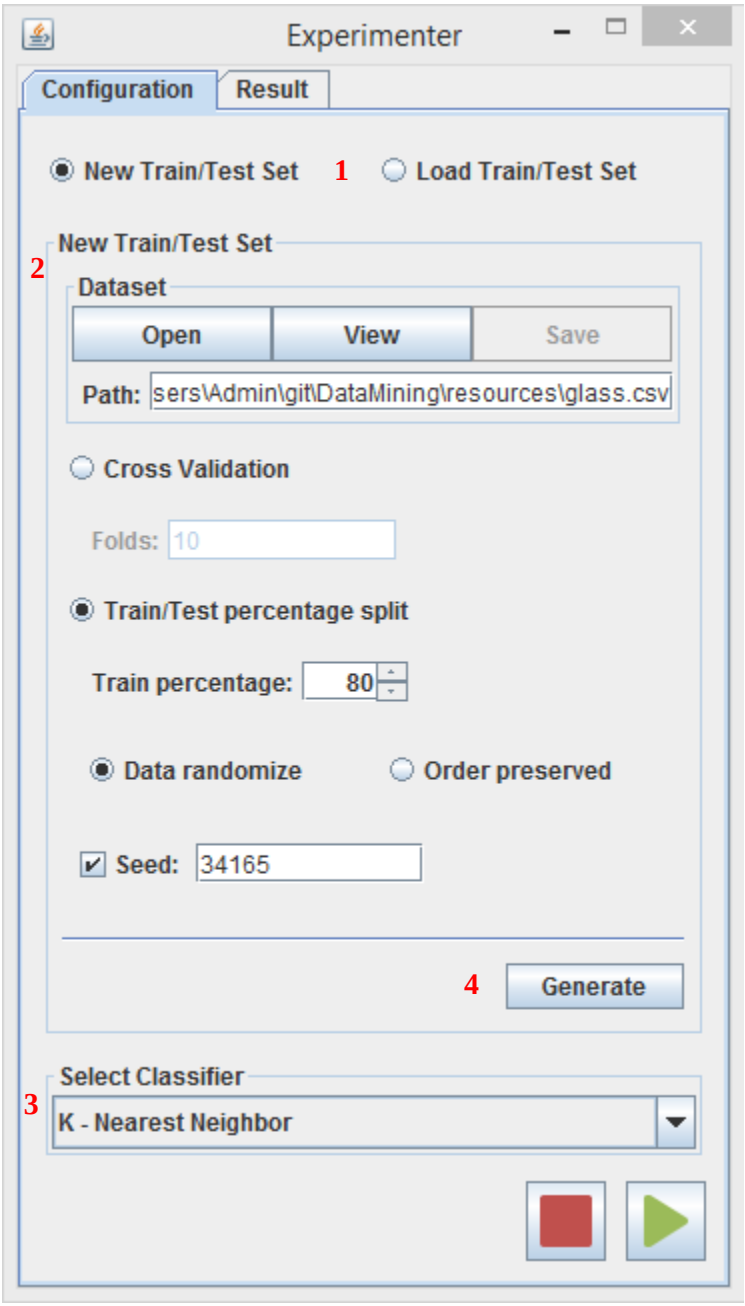


Figura 4.11. Pantalla experimentación: Configuración.

4.3.6 Pestaña de configuración: Cargar conjunto de entrenamiento y pruebas.

En la sección *"Load Train/Test Set"* (1) se cargan los conjuntos de entrenamiento y pruebas previamente generados. La sección 2 señala los controles de ejecución.

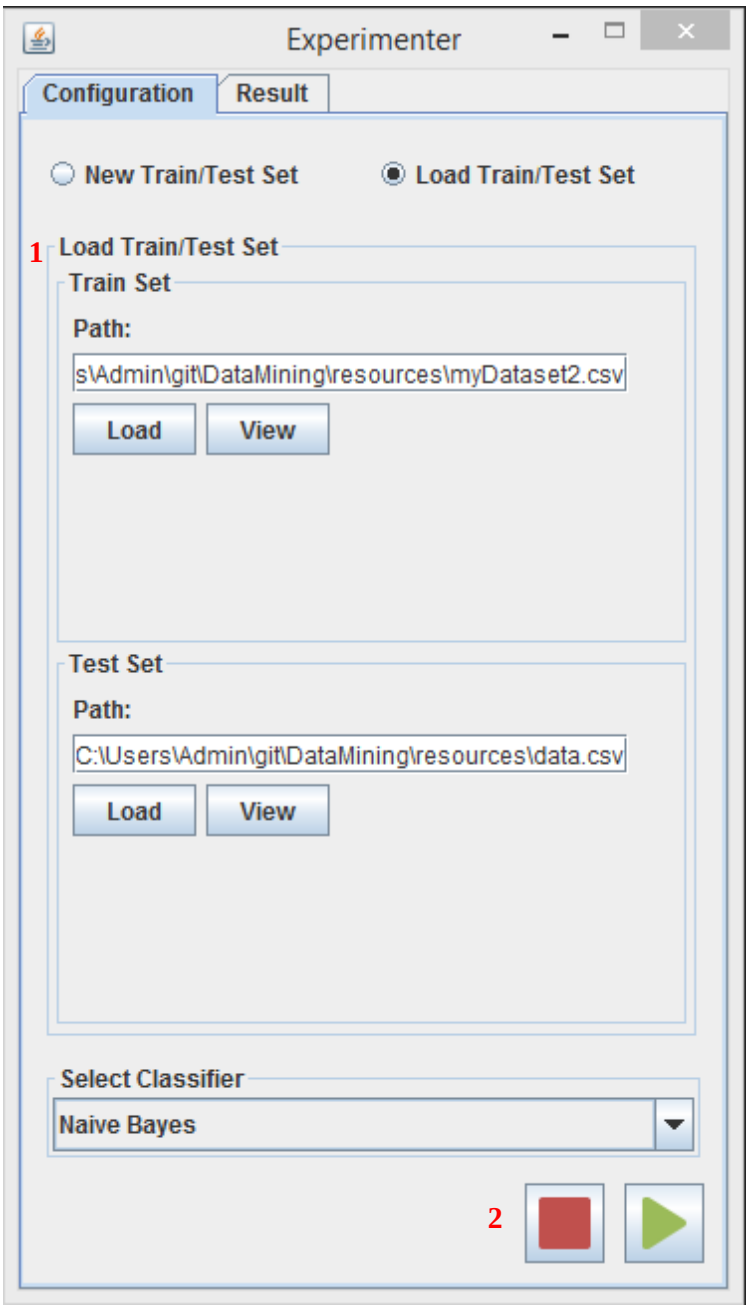


Figura 4.12. Pantalla experimentación: Cargar conjunto de entrenamiento y pruebas.

Capítulo 5.
Conclusiones y líneas futuras

Se ha alcanzado el objetivo principal de crear una biblioteca de clases extensibles, en el lenguaje de programación JAVA. Gracias a las técnicas de modelado de sistemas software, así como el uso de los principios de programación SOLID y de los patrones de diseño, se ha logrado estructurar el código adecuadamente para que sea ampliado en líneas futuras.

La biblioteca realiza labores tales como: gestión y preprocesado de datos, clasificación de datos, generación de experimentos con validación simple/cruzada y presentación de resultados mediante la matriz de confusión; ayudando al usuario a analizar y extraer conocimiento de conjuntos de datos a gran escala.

Es importante recalcar el protagonismo que han tenido las tecnologías usadas. Gracias a ellas se ha facilitado la labor del desarrollador, ahorrando tiempo y mejorando el resultado final.

Para terminar, este trabajo de fin de grado ha sido la oportunidad perfecta para aplicar las habilidades adquiridas en el grado de ingeniería informática, usando conocimientos de estadística, probabilidad, programación, ingeniería de software, entre otras. Además incita a adentrarse en el apasionante mundo de la minería de datos.

5.1 Líneas futuras.

La biblioteca servirá de base para la elaboración de una aplicación completa de minería de datos.

En la gestión de datos se pueden agregar funciones como: cargar ficheros con otro tipo de extensión o desde una URL, crear una extensión que tenga metadatos para agilizar la lectura y escritura de ficheros, cargar una base de datos relacional, entre otras.

Respecto al preprocesado de datos, se han implementado algunos filtros no supervisados. Sería acertado añadir otros filtros de este mismo tipo (añadir ruido a los datos, cambiar atributos numéricos a nominales y viceversa o alterar aleatoriamente el orden de las instancias).

También se puede contemplar el añadir filtros supervisados (discretización de un atributo, pasar un atributo de nominal a binario, crear una instancia nueva a partir de un conjunto de datos,...).

Este trabajo implementa algunos algoritmos de clasificación, sin embargo se pueden añadir otras tareas de minería de datos tales como: modelos de regresión paramétrica y no paramétrica, reglas de asociación y dependencia, métodos relacionales y estructurales, aprendizaje en redes neuronales artificiales, lógica difusa, y demás.

Aunque la matriz de confusión es una de las herramientas de evaluación más importantes en la minería de datos, se podría añadir otras presentaciones de resultados como: estadístico Kappa, error medio absoluto, error absoluto relativo, la raíz del error cuadrático medio o relativo y matriz de coste.

Finalmente, en el entorno gráfico sería interesante añadir histogramas para el análisis de datos, configuración de experimentos mediante grafos y un asistente interactivo que facilite el uso de la aplicación. Por otro lado, se puede hacer una web para facilitar el acceso al uso de la biblioteca.

Capítulo 6.
Summary and Conclusions

This project has fulfilled the main objective of creating an extensible class library in the Java programming language. By using modeling software techniques, first five principles (SOLID) and software design patterns, it has succeeded in structuring the code properly to be useful in future works.

The library performs tasks such as data management, data preprocessing and classification, generating of experiments with single and cross-validation and display results by the confusion matrix; allowing the user to analyze and extract knowledge from large-scale datasets.

It's important to point out the role of the technologies that have been used. Thanks to them, the work of developers has been facilitated, saving time and improving results.

Finally, this work has been the perfect opportunity to apply the acquired skills during the degree of computer engineering, using knowledge of statistics, probability, programming, software engineering, among others. Also it encouraged to venture into the exciting of data mining world.

6.1 Future woks

The library will be the basis for the development of a complete data mining application.

In the data management, we can add functions like load files with other extensions, create an extension with metadata for improve read/write performance, load a relational database, load files from a URL, among others.

Regarding data preprocessing, some unsupervised filters have been implemented, It would be wise to add other filters of the same type (add noise to the data, change numeric attributes into nominal ones and vice versa or randomly shuffles the order of the instances).

It can also consider adding filters supervised (attributes discretization, change nominal attribute into binary values, create a new instance from a dataset...).

This work implements some classification algorithms; furthermore it could add other data mining tasks, such as parametric and nonparametric regression models, association rules and dependence, relational and structural methods, artificial neural networks learning, fuzzy logic and any more.

Although the confusion matrix is one of the most important evaluation methods in data mining, it could add other presentations of results such as Kappa statistics, mean absolute error, relative absolute error, root or relative mean squared error and cost matrix.

Finally, for the GUI will be interesting; add histograms for data analysis, configure experiments using graphs and an interactive wizard to make easy the use of the application. On the other hand, it can make a website to facilitate access to library use.

Apéndice A.
Ficheros de configuración.

A.1. Fichero de configuración pom.xml de Maven.

```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>es.ull.datamining</groupId>
    <version>0.0.1-SNAPSHOT</version>
    <packaging>jar</packaging>

    <name>DataMining</name>
    <url>http://maven.apache.org</url>

    <properties>
        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    </properties>

    <dependencies>
        <dependency>
            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
            <version>3.8.1</version>
            <scope>test</scope>
        </dependency>
        <dependency>
            <groupId>org.renjin</groupId>
            <artifactId>renjin-script-engine</artifactId>
            <version>0.7.0-RC2</version>
        </dependency>
        <dependency>
            <groupId>net.sourceforge.javacsv</groupId>
            <artifactId>javacsv</artifactId>
            <version>2.0</version>
        </dependency>
    </dependencies>
    <repositories>
        <repository>
            <id>bedatadriven</id>
            <name>bedatadriven public repo</name>
            <url>http://nexus.bedatadriven.com/content/groups/public/</url>
        </repository>
    </repositories>
    <artifactId>datamining</artifactId>
</project>
```

Apéndice B.
Enlaces de interés.

B.1. Repositorio público de la biblioteca JDataMining.

https://github.com/jahguide89/JDataMining.git

B.2. Documentación de la biblioteca JDataMining.

http://jahguide89.github.io/JDataMining/

Bibliografía

[1] J. Hernandez Orallo, M. J. Ramírez Quintana and C. Ferri Ramírez, *Introducción a La Minería De Datos*. Madrid: Pearson Educación, S.A, 2004.

[2] R. E. Walpole, Raymond H. Myers, Sharon L. Myers and Keying Ye, *Probabilidad y Estadística Para Ingeniería y Ciencias*. México: Pearson Educación, 2007.

[3] Mitchell, T.M. *Machine Learning*. McGraw-Hill, 1997.

[4] [online]. Disponible en: http://docs.oracle.com/javase/7/docs/index.html

[5] [online]. Disponible en: http://www.oracle.com/technetwork/java/architecture-142923.html

[6] [online]. Disponible en: http://www.ntu.edu.sg/home/ehchua/programming/java/images/Swing_JComponentClassDiagram.png

[7] [online]. Disponible en: http://www.csvreader.com/java_csv.php

[8] [online]. Disponible en: https://www.r-project.org/

[9] [online]. Disponible en: http://www.renjin.org/

[10] [online]. Disponible en: https://maven.apache.org/

[11] [online]. Disponible en: http://docs.oracle.com/javase/1.5.0/docs/guide/javadoc/index.html

[12] [online]. Disponible en: https://git-scm.com/

[13] [online]. Disponible en: https://github.com/

[14] [online]. Disponible en: http://www.eclipse.org/

[15] [online]. Disponible en: https://eclipse.org/windowbuilder/

[16] [online]. Disponible en: http://www.uml-lab.com/

[17] [online]. Disponible en: http://www.eclipse.org/egit/

[18] [online]. Disponible en: http://www.eclipse.org/m2e/

[19] Martin, R.C. *Design Principles and Design Patterns* [Online]. Disponible en:
http://www.objectmentor.com/resources/articles/Principles_and_Patterns.pdf

[20] E. Gamma, R. Johnson, R. Helm and Vlissides J, *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1994.