Universidad de las Ciencias Informáticas

Facultad 3

Plataforma para el desarrollo de experimentos en el área de las reglas de asociación.

Trabajo de Diploma para optar por el título de

Ingeniero Informático

Autor: Ariandi Campos Rodríguez

Tutores: MSc. Julio César Díaz Vera

Ing. Guillermo Manuel Negrín Ortiz

La Habana, junio de 2018

Declaración jurada de autoría

Declaro ser el autor de la presente tesis y reconozco a la Universidad de las Ciencias Informáticas los derechos patrimoniales de la misma, con carácter exclusivo.

Para que así conste firmo la presente a los ____ días del mes de _____ del año _____.

_____

Ariandi Campos Rodríguez

Autor

_____ _____

Ing. Guillermo Manuel Negrín Ortiz **MSc. Julio César Díaz Vera**

Tutor Tutor

Dedicatoria

A mis padres por su apoyo incondicional.

Agradecimientos

A mi abuela por su cariño infinito.
A mis padres por apoyarme en todo momento, guiarme y ser mis ejemplos a seguir.
A mis tutores por sus recomendaciones estratégicas.
A mi oponente por sus recomendaciones al documento.
A Alexis por ser mi amigo.
A todos los que me han hecho pensar en todos estos años de estudios “Ya es lunes!” con alegría.
Muchas gracias.

Resumen

Existen dificultades asociadas al desarrollo de experimentos para comparar el desempeño de diferentes algoritmos para la extracción de reglas de asociación, como la utilización de lenguajes diferentes para la implementación de estos algoritmos y el uso de diferentes técnicas de pre procesamento de los datasets.

Este trabajo tiene como objetivo el desarrollo de un sistema capaz de homogenizar los detalles de implementación y unificar las entradas de los algoritmos y al mismo tiempo ejecutar las baterías de experimentos necesarios, para demostrar la superioridad de un algoritmo sobre otro.

Este sistema permite agregar los diferentes algoritmos para la extracción de reglas de asociación facilitando la comparación de los mismos, lo que permite la reducción del tiempo necesario para la culminación de las investigaciones sobre minería de reglas de asociación.

Palabras clave:

Regla de Asociación, Minería de datos, itemsets, Apriori, algoritmo, plataforma.

Índice

Índice de Ilustraciones

Índice de Tablas

Introducción

La minería de reglas de asociación es un tipo de descubrimiento de conocimiento que posee disímiles aplicaciones en diferentes áreas tales como medicina, biotecnología, seguridad y otras. La misma consiste en encontrar regularidades en forma de relaciones de implicación entre los atributos de los objetos de un conjunto de datos (Díaz, 2017).

Los mecanismos de extracción de reglas de asociación tienen que enfrentar varios problemas no resueltos en la literatura referente a la materia, dentro de ellos se pueden destacar:

- Se descubre un número demasiado grande de reglas incluso para bases de datos relativamente pequeñas.
- La mayoría de las reglas descubiertas no son interesantes para el usuario.
- La complejidad computacional de los algoritmos es alta.

Se han desarrollados númerosos trabajos de investigación encaminados a disminuir las afectaciones de los problemas antes mencionados en el ámbito de la minería de reglas de asociación (Díaz, 2016). Casi todos ellos tienen como resultado un nuevo algoritmo, o una modificación a los existentes, que alcanza mejor desempeño en alguno de los casos de estudio.

La inmensa mayoría de las investigaciones aplican un marco valorativo experimental en el que se comparan los resultados alcanzados por las variantes anteriores con los de la solución propuesta.

Existen un grupo de dificultades asociadas al desarrollo de experimentos para comparar el desempeño de diferentes algoritmos para el minado de reglas de asociación.

Particularmente importante en este sentido, son la utilización de lenguajes diferentes para la implementación de los algoritmos, que introduce un factor de distorsión a la hora de comparar el desempeño de dos propuestas y el uso de diferentes técnicas de pre procesamento de los dataset necesarios, que provocan variaciones en la entrada de los algoritmos que pueden afectar los resultados alcanzados. La repercusión negativa de estas dificultades en el veracidad de las investigaciones además del tiempo y el presupuesto necesario para realizarlas es innegable.

Con vista a disminuir el efecto negativo del uso de diferentes implementaciones y la no homogenización de las técnicas de pre porcesamiento se ha propuesto el desarrollo de un marco de trabajo para la investigación experimental. Uno de los principales componentes del mismo sería un sistema capaz de homogenizar los detalles de implementación y unificar las entradas de los algoritmos y al mismo tiempo ejecutar las baterías de experimentos necesarios para demostrar la superioridad de un algoritmo sobre otro.

La presente investigación está enmarcada en el desarrollo del referido sistema. Para ello se define como **problema a resolver**: ¿Cómo desarrollar una plataforma común para el desarrollo de

algoritmos de extracción de reglas de asociación que permita la aplicación de la misma entrada a todos los algoritmos y la comparación de los mismos?

Teniendo como **objeto de estudio**: Proceso de desarrollo de software.

Enmarcándose en el **campo de acción**: Proceso de desarrollo de software de aplicación.

Teniendo como **objetivo general**: Desarrollar una plataforma que permita incluir los algoritmos de extracción de reglas de asociación, de manera que puedan ser comparados entre ellos.

Para dar cumplimiento al objetivo general propuesto se definieron los siguientes **objetivos específicos**:

- Establecer el marco conceptual para el desarrollo de la investigación.
- Realizar el análisis, diseño e implementación de la plataforma para el desarrollo de experimentos en el área de las reglas de asociación.
- Validar la plataforma para el desarrollo de experimentos mediante casos de estudio.

Para el desarrollo del trabajo, se hace necesario definir las siguientes tareas de investigación:

- Estudio del marco teórico que fundamenta el objeto de investigación.
- Selección de las herramientas para el desarrollo.
- Diseño y modelación de la plataforma.
- Implementación de la plataforma.
- Validación de la solución propuesta.

Para la investigación se utilizaron los siguientes **métodos investigativos**:

Métodos Teóricos:

- Histórico-Lógico: se utilizó para conocer todos los antecedentes que existen sobre el minado de reglas de asociación, así como los principales algoritmos en este campo.
- Analítico-Sintético: se utilizó para el procesamiento de toda la información relacionada con el tema de investigación, analizando los documentos que permitieron extraer los elementos más significativos relacionados con el objeto de estudio.
- Modelación: posibilitó la creación de los diferentes diagramas y modelos que ayudaron a un mejor entendimiento de las funcionalidades que debe cumplir el sistema y al estudio de las relaciones entre las mismas.

El presente trabajo de diploma está compuesto por 3 capítulos estructurados de la siguiente manera:

En el **capítulo 1** se desarrolla toda la fundamentación teórica, se hace un análisis de los principales conceptos necesarios para el desarrollo de la propuesta, como son los conceptos relacionados con la minería de datos, plataforma, algoritmos y reglas de asociación, además se

hace un estudio de los modelos de desarrollo de software para definir la metodología que se va a utilizar en esta investigación.

En el **capítulo 2** se describen las actividades realizadas durante todo el proceso de desarrollo de la plataforma. Se detalla la propuesta de solución y se describe su arquitectura además de los artefactos que plantea la metodología utilizada.

En el **capítulo 3** se presentan los diferentes mecanismos utilizados para validar los resultados obtenidos. Entre ellos se encuentran los experimentos, el razonamiento lógico, y la demostración. Este último es el seleccionado para validar la solución propuesta en esta investigación por adaptarse a las características de la misma. Se describen los recursos computacionales utilizados para desarrollar la demostración. Se incluye además la descripción de todos los elementos que conforman el dominio de la demostración. Por último se presentan los resultados obtenidos y se realiza una detallada discusión de los mismos, demostrando la validez de la presente investigación.

Capítulo 1: Fundamentación teórica

En este capítulo se realiza un estudio sobre los elementos que fundamentan la base teórica conceptual necesaria para el desarrollo de la propuesta, como son los conceptos relacionados con la minería de datos, plataforma, algoritmos y reglas de asociación, alrededor de los cuales gira el problema que da origen a esta investigación. A partir de esto se estudia el proceso de desarrollo para dar solución al probleman analizando las definiciones de plataforma, software, requisito de alto nivel y finalmente se estudian los enfoques y metodologías para la selección de la más apropiada. Finalmente se propone por cada una de las fases de la metodología seleccionada los artefactos que se generan.

1.1 Regla de asociación

La minería de reglas de asociación es un tipo de descubrimiento de conocimiento que posee disímiles aplicaciones en diferentes áreas tales como mercado, medicina y biotecnología. La misma consiste en encontrar regularidades en forma de relaciones de implicación entre los atributos de los objetos de un conjunto de datos (Díaz, 2017).

Una regla de asociación (regla de asociación binaria) tiene la forma X → Y, donde X y Y son proposiciones, podría ser "Si un cliente compra pan y leche, entonces mayormente él también compra mantequilla". La interpretación de una regla de asociación indica que la ocurrencia del antecedente de la regla usualmente va acompañada de la ocurrencia del consecuente. El problema de minado de reglas de asociación consiste en encontrar todas las reglas interesantes sobre una colección de datos.

Formalmente una regla de asociación puede definirse como (Agrawal, 1993):

Sean I un conjunto finito de ítems, D una base de datos donde cada transacción T tenga un único identificador y contenga un conjunto de ítems. La regla de asociación es una implicación de la forma $X \rightarrow Y$ donde $X, Y \in I$, son conjuntos de ítems llamados itemsets cumpliendo que $X \cap Y = \emptyset$. Se tomará a X antecedente y a Y consecuente de la regla de asociación anterior. Siguiendo este formalismo la implicación {sobres, papel} → {sellos} denota la aparición conjunta de sellos, sobres y papel en las compras de un establecimiento de correo. Las reglas de asociación también traen relacionadas medidas que indican su calidad y/o valor para el usuario.

1.1.1 Medidas de evaluación de las reglas de asociación

Uno de los problemas principales en el descubrimiento de reglas de asociación es el desarrollo de una medida que permita evaluar la significación de las reglas descubiertas. Una regla que es interesante para un usuario pudiera no serlo para otro, por lo cual resulta muy difícil definir una

medida de interés ideal. No obstante, una medida de interés objetiva que se base en métodos estadísticos o lógicos ayuda a eliminar las reglas innecesarias y reducir el espacio de búsqueda (Pagola, y otros, 2007).

- Soporte de la regla:
  El soporte expresa la probabilidad a priori de la ocurrencia de un itemset o de una regla de asociación: supp($X \rightarrow Y$) = Fracción de las transacciones que contiene tanto a X como a Y; es decir: supp($X \cup Y$).
- Confianza de la regla:
  La confianza expresa la probabilidad condicional de que una transacción contenga al itemset Y dado que contiene al itemset X conf($X \rightarrow Y$) = Fracción de las transacciones en las que aparece X que también incluyen a Y, la confianza mide con qué frecuencia aparece Y en las transacciones que incluyen X (Berzal, 2001).

1.2 Minería de datos

La minería de datos o exploración de datos (es la etapa de análisis de "Knowledge Discovery in Databases" o KDD) es un campo de la estadística y las ciencias de la computación referido al proceso que intenta descubrir patrones en grandes volúmenes de conjuntos de datos (Rokach, 2010). Utiliza los métodos de la inteligencia artificial, aprendizaje automático, estadística y sistemas de bases de datos. El objetivo general del proceso de minería de datos consiste en extraer información de un conjunto de datos y transformarla en una estructura comprensible para su uso posterior.

Existen múltiples técnicas de minería de datos como agrupamiento, árboles de decisión y reglas de asociación, las que constituyen un modelo de descripción de conocimientos.

Los usos más comunes dados a la minería de datos son listados a continuación (Larose, 2005).

- Descripción: encontrar la manera de describir los patrones y las tendencias de los datos.
- Clasificación: para examinar los registros que contienen información sobre una variable objetivo categórica, que puede dividirse en varias clases o categorías y crear conjuntos de entrenamiento. Con base en las clasificaciones de los conjuntos de entrenamientos estas se le asignan a los nuevos registros.
- Estimación: similar a la clasificación, salvo que la variable de destino es más numérica que categórica.
- Predicción: es similar a la clasificación y la estimación a excepción de que para la predicción, los resultados están en el futuro.
- Agrupamiento (clustering): para agrupar los registros, observaciones o casos en grupos, que son colecciones de registros similares entre sí y diferentes a los registros de otros grupos.

- Asociación: para encontrar los atributos que “van de la mano”. Las reglas de la asociación son de la forma: “Si antecedente, entonces consecuente”, junto con una medida del soporte y la confianza asociados a la regla.

1.4 Algoritmos para la extracción de reglas de asociación

Un algoritmo es una secuencia finita de pasos lógicos necesarios para llevar a cabo una tarea especifica, como la solución de un problema, debe ser preciso, sin ambiguedades e indicar el orden de realización de cada paso (Martín, y otros, 1995).

Con el objetivo de obtener reglas de asociación se utilizan diferentes algoritmos para el minado que pueden causar consumo de recursos exponencial en el peor de los casos. Por lo tanto, puede tomar mucho tiempo el proceso de extracción de las reglas de asociación. El minado de forma exhaustiva de todas las reglas que satisfacen la restricción de soporte mínimo definido, puede dar lugar a la generación de un número excesivo de reglas. Entonces, el usuario final tendrá que determinar cuáles son las reglas que valen la pena utilizar (Triantaphyllou, 2010). Si la base de datos es muy densa, entonces la situación anterior puede ser aún peor. El tamaño de la base de datos (léase también cualquier formato de entrada de los datos) también juega un papel vital en los algoritmos de minería de datos (Toivonen, 1996).

1.4.1 AIS

El algoritmo AIS (Agrawal, Imielinski, Swami) fue el primer algoritmo propuesto para minería de reglas de asociación en (Agrawal, 1993). Este se centra en mejorar la calidad de las bases de datos junto con la funcionalidad necesaria para procesar consultas de soporte de decisión. En este algoritmo solo un elemento es generado en el consecuente de las reglas de asociación, por ejemplo, solo se generan reglas como $XY \Rightarrow Z$ pero no reglas como $X \Rightarrow YZ$.

Este algoritmo consiste en hacer pases múltiples sobre el conjunto de datos donde en cada pase se computan los nuevos itemsets que son generados. En cada lectura de los datos se calcula el soporte de los itemsets generados y si cumplen con el mínimo requerido pasan a ser itemsets candidatos, de lo contrario son desechados.

El principal inconveniente del algoritmo AIS es que se generan demasiados itemsets candidatos cuyo soporte es menor que el soporte mínimo, lo que requiere más espacio y procesamiento que resulta inútil. Además este algoritmo requiere demasiados pases sobre toda la base de datos.

1.4.2 SETM

El algoritmo SETM (Set Oriented Mining) fue propuesto para el minado de reglas de asociación usando operaciones relacionales en ambientes de bases de datos relacionales, motivado por el deseo de utilizar el Lenguaje Estructurado de Consulta (SQL por sus siglas en inglés) (Houtsma, 1993).

En el algoritmo SETM, los itemsets candidatos se generan sobre la marcha a medida que se escanea la base de datos, pero se cuentan al final del pase. Entonces, los nuevos itemsets candidatos se generan de la misma manera que en el algoritmo AIS, pero el identificador de transacción TID de la transacción generadora se guarda con el itemset candidato en una estructura secuencial (Khurana, et al., 2013).

1.4.3 Apriori

Varios han sido los algoritmos propuestos para generar itemsets frecuentes. De todos ellos, el método Apriori ha sido uno de los de mayor impacto, quizás el más referenciado de todos (Agrawal, 1993). La estrategia que este algoritmo sigue es del tipo descendente a lo ancho, siendo su concepción general la base para muchos de los algoritmos desarrollados posteriormente. De forma general, la mayoría de los algoritmos tipo Apriori primero construyen un conjunto de itemsets candidatos y, posteriormente, determinan el subconjunto que realmente contiene los itemsets frecuentes. Este proceso puede realizarse de forma repetitiva conociendo que los itemsets frecuentes obtenidos en una iteración servirán de base para la generación del conjunto candidato en la siguiente iteración. En particular, en el método Apriori, y variaciones de este como el AprioriTID y el Apriori Hybrid, en la k-ésima iteración se generan todos los itemsets frecuentes de tamaño k. En la siguiente iteración, para construir el conjunto de los (k+1)-itemsets candidatos, se expanden determinados k-itemsets frecuentes en un (k+1)-itemset, considerando ciertas reglas y “condiciones. Este proceso se repite hasta un cierto k, o hasta que no se puedan generar más itemsets frecuentes.

Una vez generados todos los itemsets frecuentes se procede a generar todas las posibles reglas de asociación por cada uno de ellos.

1.5 Plataforma de software

Los algoritmos para la extracción de reglas de asociación pueden tener diferentes entradas y diferentes lenguajes de implementación lo que dificulta los experimentos asociados a la comparación entre estos, por lo que se necesita crear un mecanismo para facilitar su comparación. Este mecanismo será una plataforma capaz de homogenizar los detalles de implementación de estos algoritmos, unificar sus entradas y permitir realizar comparaciones entre ellos. Plataforma es sinónimo de software y para que este exista debe tener un propósito u objetivo, en este caso es realizar experimentos con algoritmos de reglas de asociación lo que constituye un requisito de alto nivel para esta plataforma.

En informática, una plataforma es un sistema de software que sirve como base para hacer funcionar determinados módulos de hardware o de software con los que es compatible. Dicho sistema está definido por un estándar alrededor del cual se determina una arquitectura. Al definir plataformas se establecen el tipo de arquitectura, sistema operativo, lenguaje de programación o

interfaz de usuario compatibles (BuenasTareas.com, 2014).

Con el objetivo de crear esta plataforma se hace necesario un proceso de desarrollo que permita la culminación exitosa de la tarea propuesta. Por lo tanto es necesario buscar un modelo teórico que permita llevar a cabo la solución planteada. La parte de la ingeniería que se dedica al desarrollo de productos de software es la Ingeniería de Software guiado por las metodologías de dsarrrollo. Por lo tanto se hace un estudio de las mismas en las siguientes secciones para determinar cuál es la mejor y llevar a cabo la solución.

1.6 Metodología de desarrollo

El proceso de desarrollo de software, es definido como el conjunto de actividades necesarias para transformar los requisitos de un usuario en un sistema de software, tiene como finalidad la obtención de un producto que cumpla con las expectativas del cliente (Mendoza, 2004).

Las metodologías de desarrollo de software se clasifican en dos grupos. Las metodologías tradicionales, que se basan en una fuerte planificación durante todo el desarrollo y la alta resistencia a los cambios, además el usuario no ve el producto hasta el final, y las metodologías ágiles, en las que el desarrollo de software es incremental, cooperativo, sencillo y adaptado.

De acuerdo a estos elementos y a las experiencias en el desarrollo de software se determina utilizar una metodología ágil, dado que la prioridad es satisfacer al cliente mediante tempranas y continuas entregas de software (N. Juristo, 2006). Mientras que sería una dificultad para un equipo de desarrollo que en este caso es pequeño el adoptar una metodología robusta a causa de la cantidad de documentación a generar y la alta resistencia a los cambios durante el desarrollo (Joskowics, 2008), (Mendoza, 2004).

Lo anterio cual permitió identificar dentro de las metodologías ágiles a la Programación Extrema (XP) como una alternativa acertada. Dado que el equipo de desarrollo es de una sola persona no se podrá realizar la programación por pares, aunque sí se utilizarán todas las demás bondades que brinda de XP.

La metodología XP presenta varias ventajas, entre ellas (Mendoza, 2004):

- Comienza en pequeño y añade funcionalidad con retroalimentación continua.
- El manejo del cambio se convierte en parte sustantiva del proceso.
- El costo del cambio no depende de la fase o etapa.
- El cliente o el usuario se convierte en parte del equipo.

XP consta de 4 fases (D. Bustamante, 2014):

- Planificación: Es la fase donde los desarrolladores y clientes establecen los tiempos de implementación ideales de las historias de usuario, la prioridad con la que serán implementadas y las historias que serán implementadas en cada iteración.

- Diseño: La metodología XP hace especial énfasis en los diseños simples y claros. Por ello XP propone implementar el diseño más simple posible que funcione. Se sugiere nunca adelantar la implementación de funcionalidades que no correspondan a la iteración en la que se esté trabajando. En esta fase se definirá la arquitectura del software, se desarrollarán las tarjetas CRC, se definirán los patrones de diseño de software y los estándares de codificación utilizados.
- Codificación: En la fase de codificación de desarrolla en función de cada historia de usuario, además de ser fase donde se definen las tareas de la ingeniería y los tiempos reales en se realizaron cada una de las funcionalidades especificadas, en la cual la implementación, debe realizarse de acuerdo los estándares de codificación.

  En esta fase se implementarán las funcionalidades de la plataforma. No se podrá realizar la programación en pares porque el equipo tiene solo un programador.
- Pruebas: Estas pruebas se realizan al final del ciclo en el que se desarrollan, para verificar que las iteraciones no han afectado a las anteriores. Las pruebas de aceptación que hayan fallado en el ciclo anterior son analizadas para evaluar su corrección, así como para prever que no vuelvan a ocurrir.

  En esta fase se realizará la prueba del camino básico, además se realizarán casos de prueba y casos de estudio. No se usarán herramientas automatizadas porque el sistema es relativamente pequeño.

1.7 Conclusiones del capítulo

Se estableció el marco conceptual de referencia para desarrollar la investigación en el que se determinó que los algoritmos para la extracción de reglas de asociación poseen una complejidad computacional elevada. Se hizo un estudio de las metodologías de desarrollo de software llegando a la conclusión de que la metodología XP presenta varias ventajas cuando se trata de un equipo de desarrollo pequeño y existe una probabilidad alta a los cambios en los requisitos.

Capítulo 2: Plataforma de experimentos

En este capítulo se describen las actividades desarrolladas durante todo el proceso de análisis y diseño además de las fases de implementación. Se detalla la propuesta de solución y se describe su arquitectura. Se describe la fase inicial de la metodología XP: planificación, y se obtienen los artefactos importantes como las Historias de Usuarios, Plan de Iteraciones, Plan de Duración de Iteraciones y Plan de Entregas. Se presentan las fases de implementación generando las tareas de desarrollo que dan solución a cada una de las historias de usuarios identificadas en la fase de planificación.

2.1 Descripción de la solución

En el capítulo anterior se estableció la metodología de desarrollo XP como guía para la construcción de un software cuyo objetivo sea homogenizar los detalles de implementación y unificar las entradas de los algoritmos y al mismo tiempo ejecutar las baterías de experimentos se decide desarrollar una plataforma en la que se pueden agregar estos algoritmos a manera de complementos donde se permita crear y ejecutar experimentos así como la visualización de los resultados de estos. Se definen cada una de las fases de la metodología seleccionada y los artefactos genderdos por cada una a continuación.

2.2 Fase de planeación

La fase de planeación es la etapa inicial del desarrollo de software de la metodología XP. En este punto se comienza a interactuar con el cliente para identificar cuáles son las historias de usuario. Es donde se definen el número y tamaño de las historias de usuario, en donde se plantean los ajustes necesarios a la metodología según las características del proyecto y el cliente define el nivel de prioridad de las historias de usuario, además del tiempo y el esfuerzo que conllevarán su desarrollo (Echeverry, y otros, 2007).

2.2.1 Especificación de los requisitos

La ingeniería de requisitos ayuda a los ingenieros de software a entender mejor el problema en cuya solución trabajarán. Incluye el conjunto de tareas que conducen a comprender cuál será el impacto del software sobre el negocio, qué es lo que el cliente quiere y cómo interactuarán los usuarios finales (Pressman, 2010).

El proceso de recopilar, analizar y verificar las necesidades del cliente para un sistema de software es llamado Ingeniería de Requerimientos. La meta de esta es entregar una especificación de requerimientos de software correcta y completa. La misma apunta a mejorar la forma en que se comprenden y definen sistemas de software complejos (R. Grau, 2010), trata los principios, métodos, técnicas y herramientas que permiten descubrir, documentar y mantener los

requisitos para sistemas basados en computadora de forma sistemática y repetible (Méndez, 2008).

Los requisitos funcionales son declaraciones de las funcionalidades que debe proporcionar el sistema. Definen la manera en que el software debe reaccionar a determinadas entradas. Especifican cómo debe comportarse el sistema en situaciones particulares. Pueden declarar explícitamente lo que el sistema no debe hacer (Sommerville, 2005). Estos se obtuvieron a través de entrevistas con el cliente.

Los requisitos funcionales de la plataforma son los siguientes:

- Cargar experimentos.
- Cargar algoritmos.
- Cargar datasets.
- Crear experimento
- Ejecutar experimento.
- Importar experimento.
- Exportar experimento.
- Importar dataset.
- Importar algoritmo.

Un requisito no funcional especifica criterios que pueden usarse para juzgar la operación de un sistema en lugar de sus comportamientos específicos, son las restricciones o condiciones que impone el cliente al programa que necesita (Stellman, et al., 2006).

Los requisitos no funcionales de la plataforma son los siguientes:

- El tiempo de aprendizaje del sistema por un usuario deberá ser menor a media hora.
- El sistema debe poseer interfaces gráficas bien formadas.
- El sistema será desarrollado para Linux y Windows.
- El sistema será desarrollado usando Python como lenguaje de programación.
- El lenguaje del sistema será en inglés.

2.2.2 Descripción de historias de usuarios

Los requisitos funcionales describen lo que debe cumplir el sistema en un lenguaje técnico. Una historia de usuario es una representación de un requisito de software escrito en una o dos frases al utilizar el lenguaje común del usuario, son una forma rápida de administrar los requisitos de los usuarios sin tener que elaborar gran cantidad de documentos formales y sin requerir de mucho tiempo para administrarlos, permiten responder rápidamente a los requisitos cambiantes, es una manera simple de describir una tarea concisa que aporta valor al usuario o al negocio (Cohn, 2006).

Historia de usuario	
Número: 1	Nombre: Cargar experimentos.
Modificación a la Historia de Usuario: 0	
Usuario: Guillermo Negrín Ortíz	Iteración asignada: 1
Prioridad en negocio: Alta	Puntos estimados: 1 semana
Riesgo en desarrollo: Alto	Puntos reales:
Programador responsable: Ariandi Campos Rodríguez	
Descripción: La historia de usuario comienza al iniciar el programa que es donde se cargan los datos necesarios para el funcionamiento de la aplicación. Debe cargar los experimentos que se encuentran guardados en el archivo “experiments” que se encuentra en la carpeta de la aplicación.	
Observaciones: Debe ser comprobada la validez de cada experimento antes de agregarlo a la aplicación.	

Tabla 1: Historia de usuario 1.

2.2.3 Fase de Planificación de la Entrega

En la fase de la planificación de la entrega se definen las prioridades de cada historia de usuario, y consecuentemente se realiza una estimación del esfuerzo necesario de cada una de ellas. Además se definen las entregas con el cliente (Kent, et al., 2000).

En la siguiente tabla se muestran cada una de las historias de usuario, así como la estimación del tiempo en que se cumplirá, obteniéndose una duración total estimada de 10 semanas.

No	Historias de usuario	Estimación(en semanas)
1	Cargar experimentos.	1
2	Cargar algoritmos.	1
3	Cargar datasets.	1
4	Crear experimento	1
5	Ejecutar experimento.	2
6	Importar experimento.	1

7	Exportar experimento.	1
8	Importar dataset.	1
9	Importar algoritmo.	1

Tabla 2: Estimación de tiempo por historia de usuario.

2.2.4 Plan de iteraciones

Una vez definidas las historias de usuarios e identificar el tiempo para su implementación, se diseña un plan de iteraciones donde las historias de usuario están contenidas.

Se desea realizar el desarrollo en 3 iteraciones:

- **Iteración 1:**
  En esta iteración se realiza la carga de datasets, algoritmos y experimentos necesarios para el funcionamiento de la aplicación.
- **Iteración 2:**
  Esta iteración se encarga de la creación y ejecución de los experimentos.
- **Iteración 3:**
  El objetivo de esta iteración es la de importar datasets y algoritmos.

De lo anterior se deduce el tiempo estimado de cada iteración (en semanas):

Iteración	Tiempo estimado
Iteración 1	3
Iteración 2	3
Iteración 3	4

Tabla 3: Estimación de tiempo por iteración.

2.3 Diseño de la solución

La metodología XP sugiere que hay que conseguir diseños simples. Hay que procurar hacerlo todo lo menos complicado posible para conseguir un diseño fácilmente entendible y fácil de implementar, que a la larga costará menos tiempo y esfuerzo desarrollar.

2.3.1 Arquitectura basada en eventos

Para el desarrollo de la plataforma se eligió la arquitectura basada en eventos. Estos sistemas incluyen procedimientos de llamada tradicionales y vínculos entre anuncios de eventos e invocación de procedimientos. La idea dominante en la invocación implícita es que, en lugar de invocar un procedimiento en forma directa, un componente puede anunciar mediante

difusión uno o más eventos. Un componente de un sistema puede anunciar su interés en un evento determinado asociando un procedimiento con la manifestación de dicho evento (Reynoso, et al., 24).

Cuando el evento se anuncia, el sistema invoca todos los procedimientos que se han registrado para él.

La interfaz de usuario de la plataforma se mantiene a la espera de algún evento producido por el usuario para realizar sus funcionalidades. Cuando se ejecutan experimentos la plataforma debe actuar como consumidor hasta que el experimento (productor) notifique que está listo para mostrar los resultados. Esto posibilita liberar a la plataforma y que se mantenga disponible para el usuario y así permitirle realizar otras tareas, hasta que el experimento esté listo.

2.3.2 Tarjetas CRC

Las tarjetas CRC (Clase, Responsabilidad y Colaboración) son utilizadas para representar las responsabilidades de las clases y sus interacciones. Estas tarjetas permiten trabajar con una metodología basada en objetos (Beck, 1999).

Estas tarjetas representan una entidad del sistema, a la cual asignar responsabilidades y colaboraciones. El formato físico de las tarjetas CRC facilita la interacción entre los participantes del proyecto, en sesiones en las que se aplican técnicas de grupos como tormenta de ideas o juego de roles y se ejecutan escenarios a partir la de especificación de requisitos, historias de usuarios o casos de uso. De esta forma, van surgiendo las entidades del sistema junto con sus responsabilidades y colaboraciones (S. Casas, 2009).

A continuación, se muestran la tarjetas CRC generadas:

Clase: Algorithm	
Responsabilidades	**Colaboradores**
execute_command execute_command_linux execute_command_windows is_excecutable do_excecutable is_valid load_info_from_excecutable run	Popen Dataset ReturnCodeError AlgorithmError

Tabla 4: Tarjeta CRC de la clase Algorithm.

Clase: AlgorithmManager	
Responsabilidades	**Colaboradores**
load_algorithms populate_algorithms_info remove_algorithm	Algorithm

Tabla 5: Tarjeta CRC de la clase AlgorithmManager.

Clase: DataLabView	
Responsabilidades	**Colaboradores**
set_experiment calculate_metrics	Experiment

Tabla 6: Tarjeta CRC de la clase DataLabView.

2.3.3 Patrones de diseño de software

Los patrones son soluciones simples y elegantes a problemas específicos y comunes del diseño orientado a objetos. Sus soluciones están basadas en los problemas del diseño que se repiten y que se presentan en situaciones particulares (Mühlrad, 2008).

Un patrón es un conjunto de información que proporciona respuesta a un conjunto de problemas similares. Para ello se aíslan los aspectos comunes y su solución y se añaden cuantos comentarios y ejemplos sean oportunos. Los patrones ayudan a capturar conocimiento y a crear un vocabulario técnico, hacen el diseño orientado a objetos más flexibles, elegante y en algunos casos reusable (E. Gamma, 1995).

Entre los patrones de diseño se pueden mencionar los patrones de asignación de responsabilidades GRASP (patrones generales de software para asignación de responsabilidades, siglas de General Responsibility Assignment Software Patterns) y los patrones GOF (siglas de Gang of Four) que es el nombre con el que se conoce comúnmente a los autores del libro Design Patterns. Los patrones GRASP describen los principios fundamentales de la asignación de responsabilidades a objetos. Dentro de este grupo de patrones se encuentran los siguientes:

Experto, Creador, Bajo Acoplamiento, Alta Cohesión, Controlador, Fabricación Pura, Indirección, Variaciones Protegidas, No Hables con Extraños y Polimorfismo (Larman, 2003).

En el diseño del sistema se evidencian los siguientes patrones:

Patrones GRASP:

- Creador: El patrón Creador nos ayuda a identificar quién debe ser el responsable de la creación de nuevos objetos o clases. Se utilizó en la clase *AlgorithmManager* para identificar que clase es la responsable de crear los algoritmos.

```
class AlgorithmManager:
    def __init__(self):
        self.algorithms = dict()

    def load_algorithms(self):
        files = [file for file in listdir('algorithms') if
                 isfile('algorithms' + SEP + file) and
                 file.startswith('adl_')]
        for file in files:
            algorithm = Algorithm(file)
            if algorithm.is_valid_cras():
                self.algorithms[file] = algorithm
        self.populate_algorithms_info()
```

- Experto: Asigna la responsabilidad al experto en la información, es decir, a la clase que cuenta con la información necesaria para cumplir con la responsabilidad. Se utilizó en la clase *Algorithm* para identificar que clase tiene la información nesesaria para que pueda cumplir con la responsabilidad de ejecutar un algoritmo.

```
class Algorithm:
    def run(self, dataset):
        """Run the algorithm and returns the output."""
        dataset_names = dataset[:dataset.rfind('_')] + '_names.csv'
        self.results[dataset] = self.execute_command([dataset,
                                dataset_names] + self.input_params)
```

- Alta Cohesión: La información que almacena una clase debe ser coherente y en la medida de lo posible relacionada con ella. Se utilizó en la clase *Algorithm* para asignar su responsabilidad exacta.

```
class Algorithm:
    def __init__(self, file):
        self.name = file
        self.version = 'unknow'
        self.file = 'algorithms' + SEP + file
        self.info = ''
        self.inputs = []
        self.input_params = []
        self.outputs = []
        self.is_valid = False
        self.results = {}
        if not self.is_excecutable():
            self.make_excecutable()
```

- Bajo Acoplamiento: El propósito del patrón es tener las clases lo menos ligadas posible y de producirse una modificación en alguna de ellas, se tenga la mínima repercusión en el resto. Se utilizó en la clase *ExperimentManager* para lograr la menor relación posible entre las clases.

```
class ExperimentManager:
    def __init__(self, experiments):
        self.experiments = experiments

    def remove_experiment(self, name):
        del self.experiments[name]
```

Patrones GOF:

- Fachada: Provee una interfaz unificada y simple para acceder a una interfaz o grupo de interfaces de un subsistema. Se utilizó en la clase *Program* para establecer la interfaz principal.

```
class Program(QtGui.QMainWindow):
    def __init__(self):
        QtGui.QMainWindow.__init__(self)
        self.ui = Ui_MainWindow()
        self.ui.setupUi(self)
```

- Módulo: Agrupa varios elementos relacionados, como clases, singletons y métodos utilizados globalmente en una entidad única.
- Estrategia: Permite disponer de varios métodos para resolver un problema y elegir cuál utilizar en tiempo de ejecución. Se utilizó en la clase *Algorithm* para identificar que método ejecutar dependiendo de si se ejecuta en Windows o Linux.

```
class Algorithm:
    def __init__(self, file):
        self.execute_command = self.execute_command_linux if
                               is_linux else
                               self.execute_command_windows

    def execute_command_linux(self, cmd):
        rr = self.file.replace(' ', '\\ ') + ' ' + '
             '.join([s.replace('
             ', '\\ ') for s in cmd])
        process = Popen(rr, stderr=PIPE, stdout=PIPE, shell=True)
        out, err = process.communicate()
        if process.returncode:
            raise ReturnCodeError(err)
        if err:
            raise AlgorithmError(err)
        return out

    def execute_command_windows(self, cmd):
        rr = [self.file] + cmd
        process = Popen(rr, stderr=PIPE, stdout=PIPE, shell=True)
        out, err = process.communicate()
        if process.returncode:
            raise ReturnCodeError(err)
        if err:
            raise AlgorithmError(err)
        return out
```

2.3.4 Estándares de codificación

Un estándar de codificación consiste en una guía que facilita la lectura del código y la consistencia entre programas de distintos usuarios. Para el desarrollo de la aplicación se utilizó PEP 8 una guía

de estilo para Python desarrollada por Guido van Rossum, Barry Warsaw y Nick Coghlan. A continuación algunas de sus recomendaciones (van Rossum, et al., 2001):

- Utilizar siempre 4 espacios para tabular y nunca mezclar tabuladores y espacios.
- Las líneas deben limitarse a un máximo de 79 caracteres.
- Los nombres de clase deben tener el estilo “CamelCase”.
- Los nombres de las funciones deben tener el estilo “snake_case”.

2.4 Fase de implementación

Dentro de la metodología del desarrollo del software, la parte más importante es la implementación. Se especifica en esta fase, la implementación de las historias de usuario en su correspondiente iteración, obteniéndose en cada una de ellas una versión funcional del producto.

2.4.1 Lenguaje de programación

Un lenguaje de programación es un idioma artificial diseñado para expresar instrucciones que pueden ser llevadas a cabo por un ordenador. Puede usarse para crear programas que controlen el comportamiento físico y lógico de una máquina, para expresar algoritmos con precisión o como modo de comunicación humana. Permiten especificar de manera precisa sobre qué datos debe operar una computadora, cómo deben ser almacenados o transmitidos y qué acciones debe tomar bajo una gran cantidad de opciones posibles. Todo esto, a través de un lenguaje que intenta ser relativamente próximo al lenguaje humano o natural (Suarez, 2017). Se escogió como lenguaje de programación a Python 2.7, por tener una sintaxis simple, clara y sencilla, es multiplataforma y orientado a objetos, además posee una gran cantidad de bibliotecas disponibles y la potencia del lenguaje hacen que desarrollar una aplicación en Python sea sencillo y muy rápido. Características como estas permiten que Python pueda ser usado para combinar varios componentes juntos, lo que lo hace ideal para el desarrollo de la aplicación.

2.4.2 Entorno integrado de desarrollo

Un entorno de desarrollo integrado o entorno de desarrollo interactivo, en inglés *Integrated Development Environment* (IDE), es una aplicación informática que proporciona servicios integrales para facilitarle al desarrollador o programador el desarrollo de software (Isidro Ramos Salavert, 2000).

Normalmente, un IDE consiste de un editor de código fuente, herramientas de construcción automáticas y un depurador. La mayoría de los IDE tienen auto-completado inteligente de código. Algunos IDE contienen un compilador, un intérprete, o ambos.

Para el desarrollo de la solución se seleccionó como IDE Pycharm 2017.2 . PyCharm es un IDE o entorno de desarrollo integrado multiplataforma utilizado para desarrollar principalmente en el lenguaje de programación Python por poseer las siguientes características (Naranjo, 2018):

- Asistencia y análisis de codificación, con finalización de código , sintaxis y resaltado de errores.
- Navegación de proyectos y códigos: vistas de proyectos especializados, vistas de estructura de archivos y saltos rápidos entre archivos, clases, métodos y usos.
- Refactorización de código Python : incluye renombrar, extraer método, introducir variable, introducir constante y otros.
- Depurador integrado de Python.
- Integración de control de versiones para Git.

2.4.3 Tareas de desarrollo

Lo primero es hacer un chequeo de cada historia de usuario, en conjunto con el plan de iteraciones y se modifica en caso de ser necesario, para esto se crean tareas de desarrollo para de esta forma poder organizar la implementación.

A continuación se muestran las tareas de ingeniería efectuadas para las funcionalidades implementadas en cada una de las iteraciones definidas en la fase de planificación.

2.4.3.1 Iteración 1

En esta iteración se implementaron las funcionalidades que tiene como fin cargar experimentos, algoritmos y datasets.

Tarea	
Número de tarea: 1	Número: 1
Nombre: Cargar experimentos.	
Tipo de tarea: Desarrollo	Puntos de estimación: 1
Programador responsable: Ariandi Campos Rodríguez	
Descripción: Se cargan los experimentos almacenados en el fichero “experiments” en la carpeta del programa.	

Tabla 5: Tabla de ingeniería de la funcionalidad Cargar Experimentos.

Tarea	
Número de tarea: 2	Número: 2
Nombre: Cargar algoritmos.	
Tipo de tarea: Desarrollo	Puntos de estimación: 1
Programador responsable: Ariandi Campos Rodríguez	
Descripción: Se cargan los algoritmos CRAS que se encuentran en la carpeta “algorithms/”	

Tabla 6: Tabla de ingeniería de la funcionalidad Cargar Algoritmos.

Tarea	
Número de tarea: 3	Número: 3
Nombre: Cargar datasets.	
Tipo de tarea: Desarrollo	Puntos de estimación: 1
Programador responsable: Ariandi Campos Rodríguez	
Descripción: Se cargan los datasets de la carpeta “algorithms/data”	

Tabla 7: Tabla de ingeniería de la funcionalidad Cargar datasets.

2.4.3.2 Iteración 2

En esta iteración se implementaron las funcionalidades que tiene como fin la creación y ejecución de experimentos.

Tarea	
Número de tarea: 4	Número: 4
Nombre: Crear experimento.	
Tipo de tarea: Desarrollo	Puntos de estimación: 1
Programador responsable: Ariandi Campos Rodríguez	
Descripción: Se crea un experimento con los datos introducidos por el usuario.	

Tabla 8: Tabla de ingeniería de la funcionalidad Ejecutar experimento.

Tarea	
Número de tarea: 5	Número: 5
Nombre: Ejecutar experimento.	
Tipo de tarea: Desarrollo	Puntos de estimación: 2
Programador responsable: Ariandi Campos Rodríguez	
Descripción: Se ejecuta el experimento seleccionado, mostrando los resultados del mismo.	

Tabla 9: Tabla de ingeniería de la funcionalidad Ejecutar experimento.

2.4.3.3 Iteración 3

En esta iteración se implementaron las funcionalidades que tiene como fin importar y exportar experimentos e importar datasets y algoritmos.

Tarea	
Número de tarea: 6	Número: 6
Nombre: Importar experimento.	
Tipo de tarea: Desarrollo	Puntos de estimación: 1
Programador responsable: Ariandi Campos Rodríguez	
Descripción: Se permite importar experimentos desde un archivo.	

Tabla 10: Tabla de ingeniería de la funcionalidad Importar experimento.

Tarea	
Número de tarea: 7	Número: 7
Nombre: Exportar experimento.	
Tipo de tarea: Desarrollo	Puntos de estimación: 1
Programador responsable: Ariandi Campos Rodríguez	
Descripción: Se permite exportar experimentos hacia un archivo.	

Tabla 11: Tabla de ingeniería de la funcionalidad Exportar experimento.

Tarea	
Número de tarea: 8	Número: 8
Nombre: Importar dataset.	
Tipo de tarea: Desarrollo	Puntos de estimación: 1
Programador responsable: Ariandi Campos Rodríguez	
Descripción: Se permite importar datasets desde el disco.	

Tabla 12: Tabla de ingeniería de la funcionalidad Importar dataset.

Tarea	
Número de tarea: 9	Número: 9
Nombre: Importar algoritmo.	
Tipo de tarea: Desarrollo	Puntos de estimación: 1
Programador responsable: Ariandi Campos Rodríguez	
Descripción: Se permite importar algoritmos desde el disco.	

Tabla 13: Tabla de ingeniería de la funcionalidad Importar algoritmo.

2.4.4 Interfaz de usuario

La interfaz principal contiene una lista con los experimentos creados y la información referente al experimento seleccionado además de las funcionalidades para gestionar experimentos, algoritmos y datasets. También permite ejecutar las baterías de experimentos así como la creación de estos.

Desde la interfaz principal se puede acceder a la documentación de los algoritmos que contiene la plataforma y a la ayuda de la plataforma.

A continuación se muestra la interfaz principal del sistema:

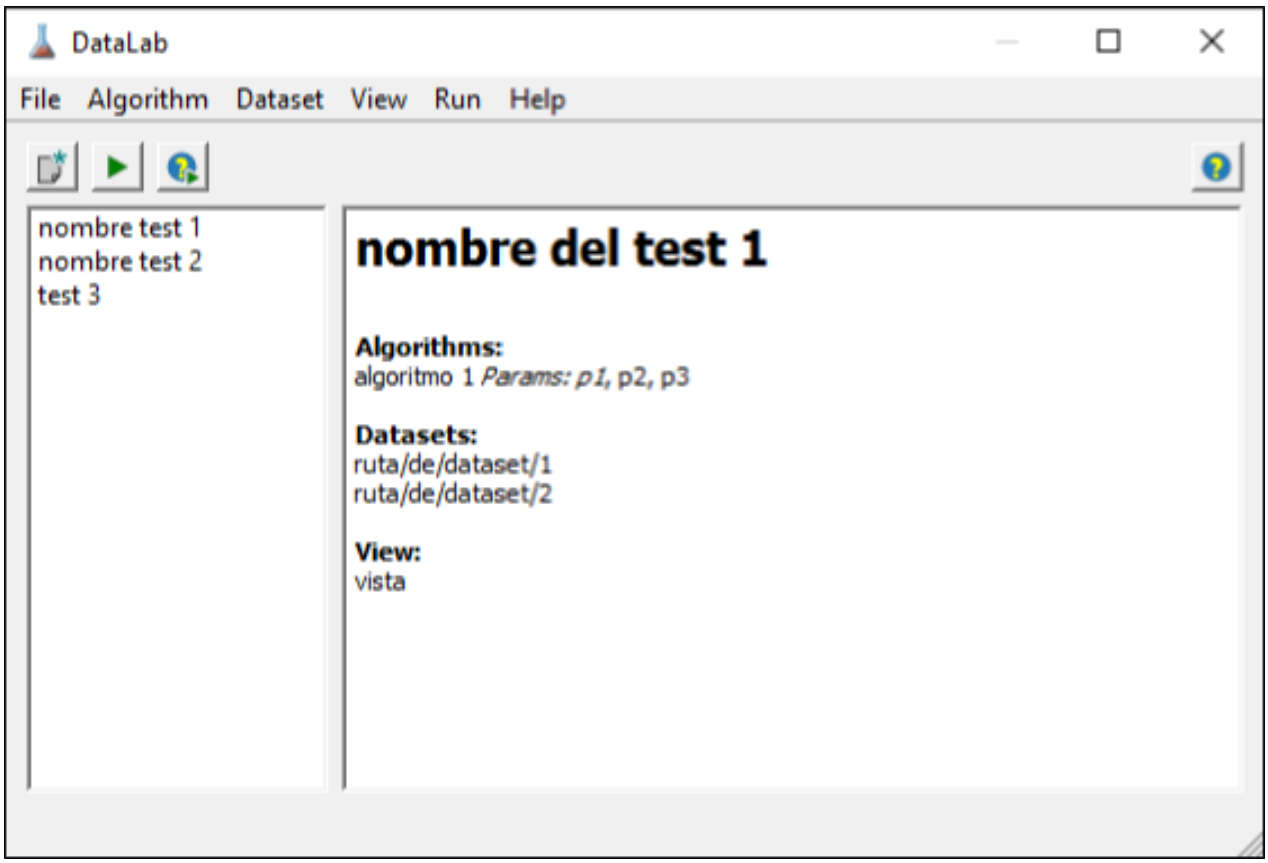


Ilustración 1: Interfaz principal.

A continuación, se muestra un diagrama de procesos donde se describen las actividades a realizar en la interfaz principal.

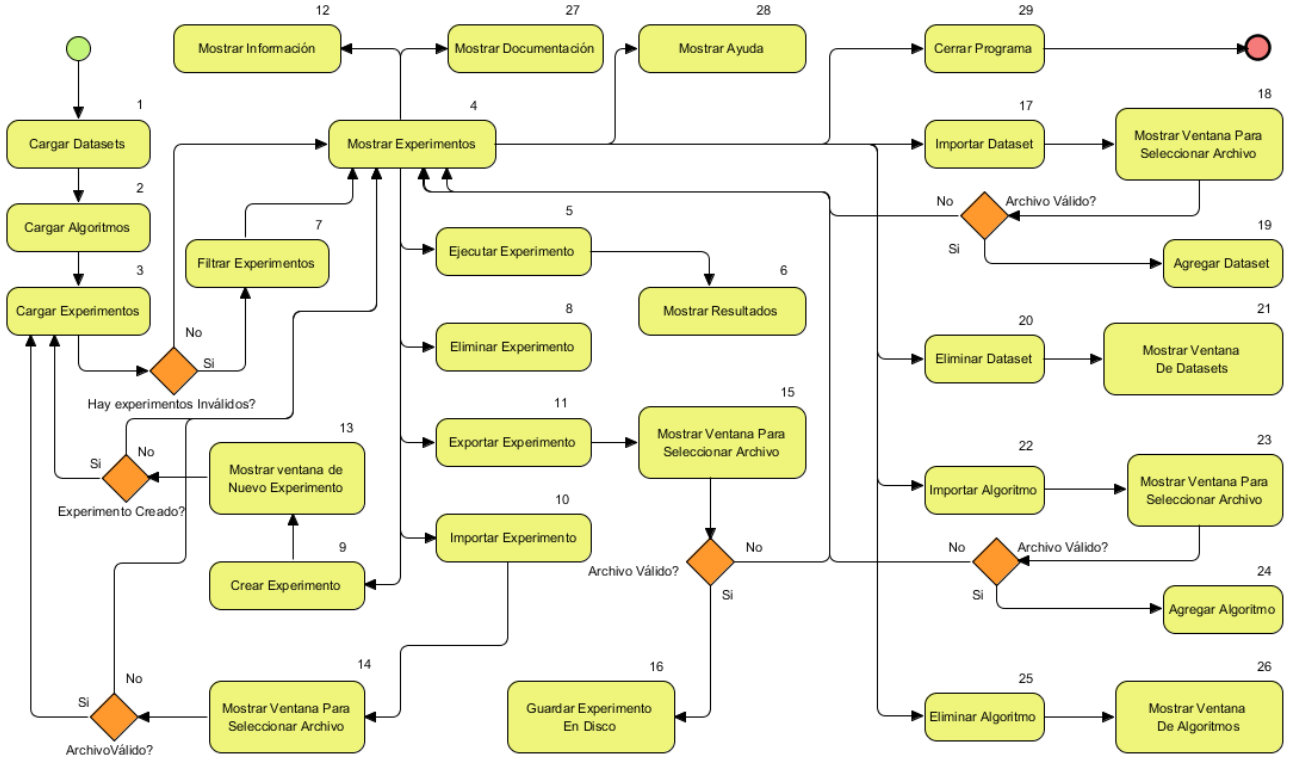


Ilustración 2: Diagrama de flujo de la interfaz principal.

1. Cargar los datasets disponibles desde el disco que se encuentran en el directorio “algorithms/data/”. Cada dataset se compone de dos archivos: “Nombre_names.csv” y “Nombre.csv”. En el primero se encuentran mapeados los identificadores con los nombres de los atributos mientras que en el segundo se encuentra el conjunto de datos.
2. Cargar los algoritmos disponibles desde el disco que se encuentran en el directorio “algorithms/”.Los algoritmos deben ser validados para que puedan ser agregados a la plataforma.
3. Cargar los experimentos disponibles desde el disco que se encuentran en el archivo “experiments”. Si no existe este archivo, es creado con el objetivo de guardar los futuros experimentos.
4. Mostrar las baterías de experimentos válidos. Debe mostrarse la lista con todos los experimentos, esta lista permite seleccionar un elemento para mostrar la información sobre el experimento seleccionado.
5. Ejecutar experimentos. El experimento seleccionado es ejecutado con los parámetros introducidos por el usuario en la creación del experimento.
6. Mostrar los resultados del experimento ejecutado. Se muestran los resultados del experimento en la vista correspondiente al experimento ejecutado.
7. Filtrar los experimentos válidos para ser mostrados. Es necesario filtrar los experimentos ya que el archivo donde se almacenan puede estar corrupto.
8. Eliminar experimento.
9. Crear un nuevo experimento. Un experimento contiene un identificador único, uno o más algoritmos, uno o más datasets y una vista para mostrar los resultados.
10. Importar un experimento desde el disco. Debe comprobarse que se trate de un experimento válido.
11. Exportar experimento hacia un fichero.
12. Mostrar información sobre el experimento seleccionado. Debe mostrarse el identificador, los algoritmos que contiene con los parámetros de entrada, los datasets que contiene, así como la vista en donde se mostrarán los resultados del experimento.
13. Mostrar ventana de nuevo experimento. El usuario debe elegir un nombre para el nuevo experimento, al menos un algoritmo, al menos un dataset y una vista.
14. Mostrar ventana para seleccionar un experimento a importar desde el disco.
15. Mostrar ventana para seleccionar un experimento a exportar hacia el disco.
16. Guardar experimento seleccionado hacia un fichero.
17. Importar dataset desde archivo.
18. Mostrar ventana para seleccionar un dataset a importar desde el disco.
19. Agregar dataset importado.
20. Eliminar dataset seleccionado.

21. Mostrar ventana de datasets para seleccionar los que se desean eliminar.
22. Importar algoritmo desde archivo.
23. Mostrar ventana para seleccionar un algoritmo a importar desde el disco.
24. Agregar algoritmo importado.
25. Eliminar algoritmo.
26. Mostrar ventana de algoritmos para seleccionar los que se desean eliminar.
27. Mostrar la documentación obtenida desde los algoritmos, dicha información se obtiene de la documentación que proporciona el algoritmo al ejecutarlo con el parámetro “–h”.
28. Mostrar ayuda de la aplicación.
29. Cerrar el programa.

La interfaz de Nuevo Experimento permite seleccionar al menos un algoritmo, al menos un dataset y la vista donde se mostrarán los resultados para crear un nuevo experimento con un nombre dado.

A continuación se muestra la interfaz de Nuevo Experimento:

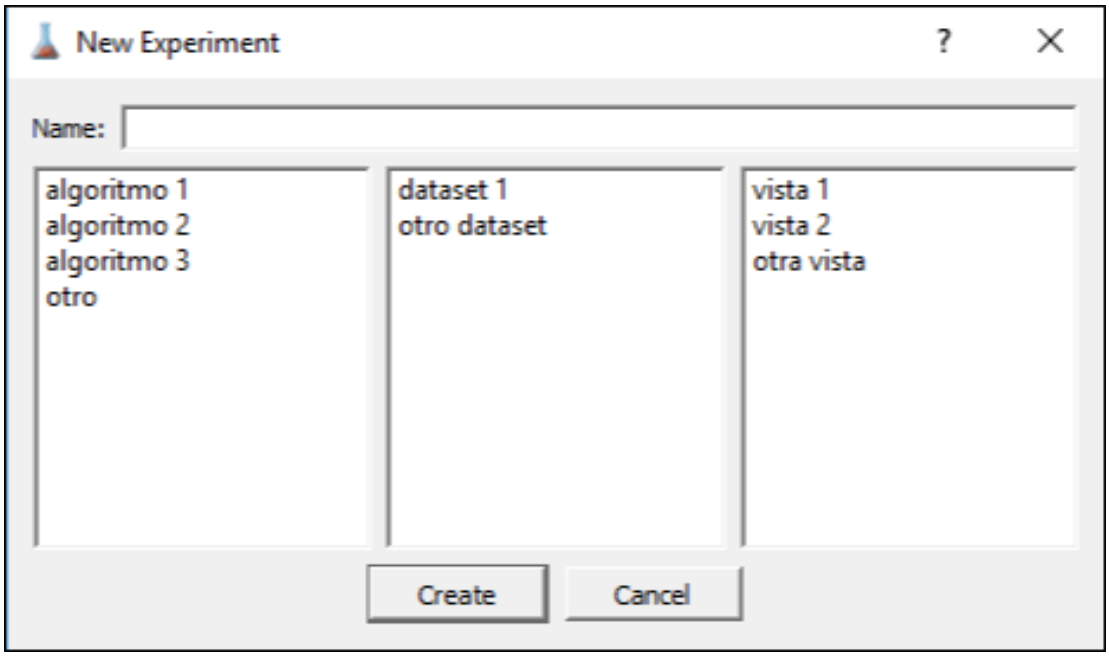


Ilustración 3: Interfaz Nuevo Experimento.

A continuación, se muestra un diagrama de procesos donde se describen las actividades a realizar en la interfaz de nuevo experimento.

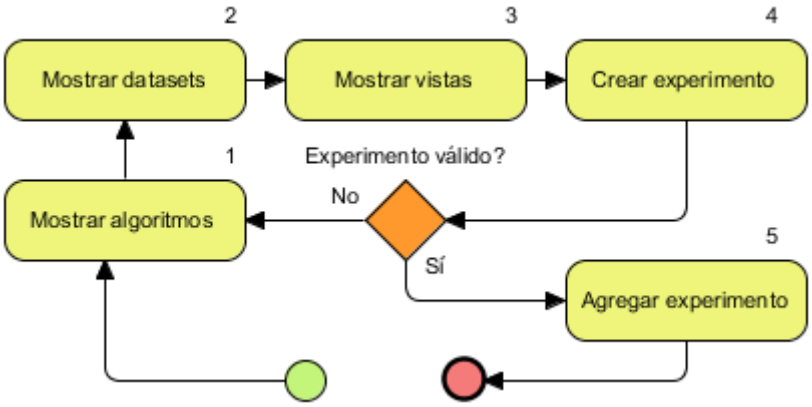


Ilustración 4: Diagrama de flujo de la interfaz Nuevo Experimento.

1. Mostrar algoritmos disponibles.
2. Mostrar datasets disponibles.
3. Mostrar vistas disponibles.
4. Crear un nuevo experimento con los datos proporcionados, se debe comprobar que sean válidos dichos datos.
5. Agregar a la lista de experimentos el experimento creado.

La interfaz de Ayuda de Algoritmos es la encargada de mostrar la documentación que se obtiene de cada algoritmo con el objetivo de facilitar el uso a usuarios distintos del que implementa el algoritmo.

A continuación se muestra la interfaz de Ayuda de Algoritmos:

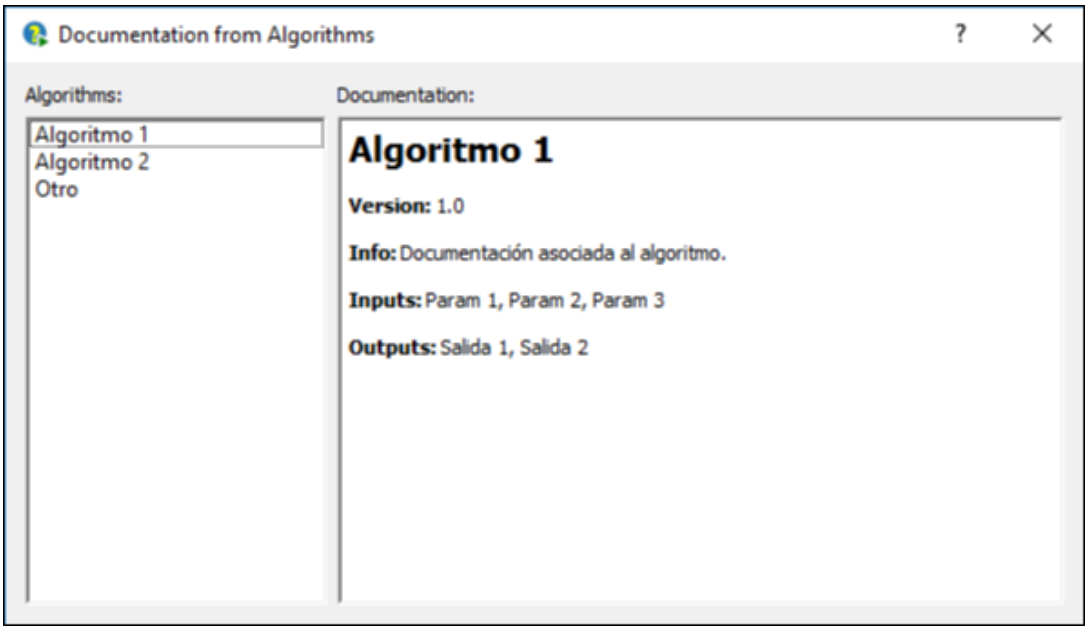


Ilustración 5: Interfaz Ayuda de Algoritmos.

Las vistas son interfaces de usuario encargadas de mostrar los resultados luego de la ejecución de un algoritmo.

A continuación se muestra un ejemplo de una vista que permite realizar varios tests estadísticos con el objetivo de comprobar si existen diferencias significativas entre varias métricas de los algoritmos de un experimento.

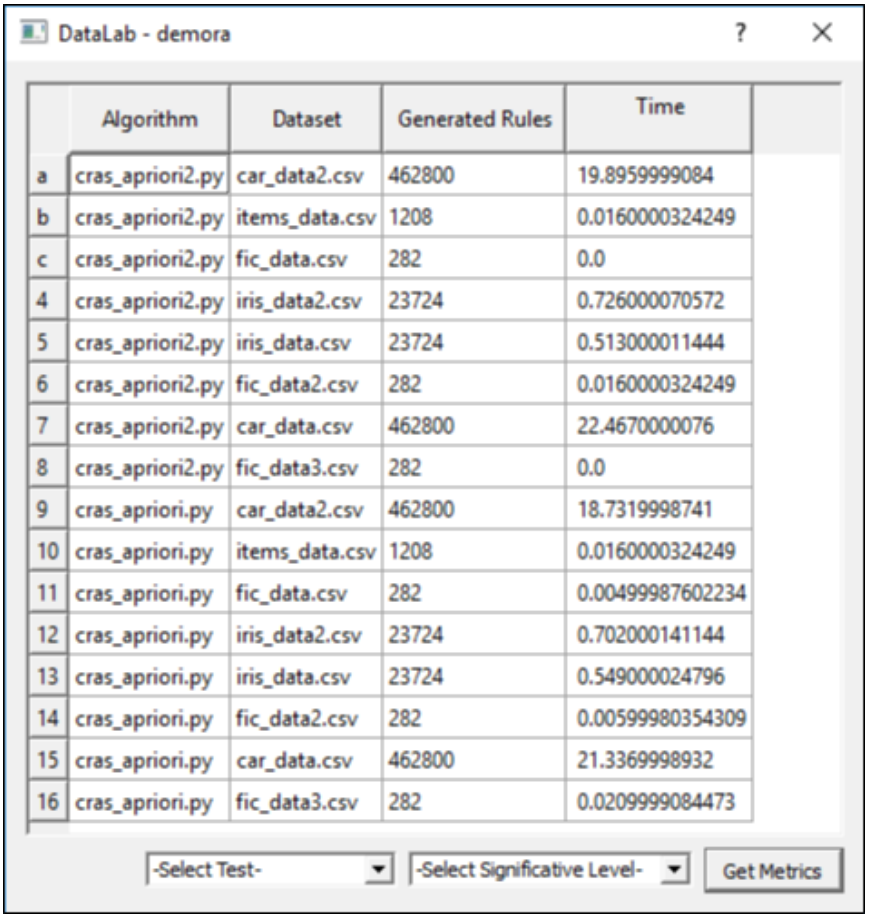


Ilustración 6: Ejemplo de resultado para un algoritmo.

2.4.5 Algoritmos válidos para la plataforma

Para el desarrollo de la plataforma que da cumplimiento al objetivo general de este trabajo se hace necesario contar con un conjunto de características adicionales sobre los algoritmos para que puedan ser reconocidos a manera de complementos (*plugins*) válidos:

Para la comunicación entre la plataforma y los algoritmos se establece stdin, stdout y stderr que son la entrada, la salida y la salida de errores respectivamente. La terminal del sistema es la encargada, de abrir estas entradas y conectarla con el terminal al que está asignada la plataforma para establecer la comunicación.

- El nombre del archivo debe comenzar con “cras_”.
- El primer parámetro de la línea de comandos debe ser el dataset de entrada si no se utilizan los parámetros de opción, en otro caso debe ser uno de los parámetros de opción de la tabla: ‘‘Parámetros de opción”.
- Los datos de entrada se envían a través de la entrada estándar (stdin).
- Los datos de salida se envían a la salida estándar (stdout).
- Los datos de salida de error se envían hacia el error estándar (stderr).

Además debe ser capaz de responder adecuadamente a los siguientes parámetros de opción:

Significado	Parámetro	Salida esperada
Tipo de algoritmo.	-t	“Complemento-Datalab”
Obtener la documentación del algoritmo.	-h	Ayuda del algoritmo.
Obtener la versión del algoritmo.	-v	Versión del algoritmo.
Obtener la lista de elementos de entrada del algoritmo.	-p	Lista de elementos de entrada.
Obtener la lista de elementos de salida del algoritmo.	-r	Lista de elementos de salida.

Tabla 14: Parámetros de opción.

2.5 Conclusiones del capítulo

Se implementó la plataforma para el desarrollo de experimentos en el área de las reglas de asociación demostrando que con modificaciones mínimas es posible adaptar los algoritmos existentes en minería de reglas de asociación de forma tal que puedan ser agregados a la plataforma. Mediante la descripción de las nueve historias de usuario divididas en tres iteraciones se logró una buena organización del trabajo y se establecieron diez semanas como tiempo de duración para las tres iteraciones. La utilización de la arquitectura basada en eventos permitió una buena estructuración de la aplicación.

Capítulo 3: Validación de resultados

Durante el desarrollo del presente capítulo se realiza la validación de la investigación. La primera parte se dedica a usar la validación y verificación que presenta la metodología de desarrollo. En este caso se realizan pruebas de caja blanca y de caja negra con su diseño de casos de prueba. Las iteraciones permitieron identificar las no conformidades y resolverlas. Posteriormente se dedica el resto del capítulo a realizar experimentos con tres casos de estudio ofreciendo para cada caso la salida del sistema.

3.1 Validación del sistema

Los errores sencillos de un sistema que suelen estar ocultos, con el trascurso del desarrollo del mismo son más difíciles de detectar que los grandes errores que están a simple vista. Por esto es de suma importancia tener en cuenta la aplicación de las pruebas de software en las distintas etapas de desarrollo del producto, pues las correctas aplicaciones de estas garantizan su calidad, provocando satisfacción al cliente y conformidad con lo realizado. Teniendo dominio sobre todos los procesos involucrados en el mismo, se aplican diferentes métodos y técnicas.

3.1.1 Pruebas de caja blanca

Este método se centra en cómo diseñar los casos de prueba atendiendo al comportamiento interno y la estructura del programa. Se examina así la lógica interna del programa sin considerar los aspectos de rendimiento (J Gutierrez, 2010).

El objetivo de la técnica es diseñar casos de prueba para que se ejecuten, al menos una vez, todas las sentencias del programa, y todas las condiciones tanto en su vertiente verdadera como falsa.

3.1.1.1 Prueba del camino Básico

Para la aplicación de las pruebas de caja blanca se hizo uso de la técnica camino básico. El método del camino básico permite obtener una medida de la complejidad de un diseño procedimental, y utilizar esta medida como guía para la definición de una serie de caminos básicos de ejecución, diseñando casos de prueba que garanticen que cada camino se ejecuta al menos una vez.

Se toma como ejemplo el método *export_experiment* de la clase *Program* como base para realizar la técnica del camino básico:

```
def export_experiment(self):
    selected = self.ui.listWidget_tests.selectedItems()
    selected_length = len(selected)
    if selected_length > 0:
        file_name = str(QtGui.QFileDialog.getSaveFileName(self, 'Select file to
                                                          export:'))
        if file_name != '':
            name = str(self.ui.listWidget_tests.selectedItems()[0].text())
            exp_shelve = shelve.open(file_name)
            exp_shelve['experiment'] = self.experiment_manager.experiments[name]
            exp_shelve.close()
    else:
        QtGui.QMessageBox.information(self, 'Upss: You must select experiments
                                      first.')
```

Partiendo del fragmento de código tomado se obtiene el siguiente grafo de flujo:

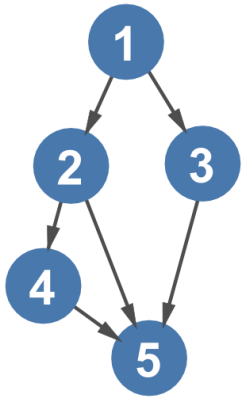


Ilustración 7: Representación del grafo de flujo de camino básico.

Luego se calculó la complejidad ciclomática V (G), obteniendo el resultado siguiente:

V (G) = Aristas – Nodos + 2
V (G)= 6 – 5 + 2

V (G)= 3

El valor V (G) expresa la cantidad de caminos linealmente independientes de la estructura de control del programa, por lo que se definen los siguientes 3 casos de prueba:

Camino	Condición de ejecución	Prueba	Salida	Resultado de la prueba
1245	Se seleccionó al menos 1 experimento. Se seleccionó un archivo.	selected_length=5 file_name=”/home/exp.dle”	Experimentos seleccionados guardados.	Prueba satisfactoria
125	Se seleccionó al menos 1 experimento. No se seleccionó un archivo.	selected_length=5 file_name=””	Se cancela guardar experimentos seleccionados.	Prueba satisfactoria
135	No se seleccionó al menos 1 experimento. No se seleccionó un archivo.	selected_length=0	Se muestra mensaje de aviso:Debe seleccionar al menos un experimento.	Prueba satisfactoria

Tabla 15: Casos de prueba.

3.1.2 Pruebas de caja negra

Las pruebas de caja negra se centran en lo que se espera de un módulo, es decir, intentan encontrar casos de prueba en que el módulo no se atiene a su especificación. Esto se refiere a que se llevan a cabo para verificar el ajuste del sistema con los requerimientos determinados. Además, se enfocan especialmente en los módulos que se relacionan con la interfaz de usuario. No requieren el conocimiento de la estructura interna del programa para su puesta en marcha (J Gutierrez, 2010).

Estas pruebas tienen como propósito detectar (Pressman, 2010):

- Funciones incorrectas o ausentes.
- Errores de interfaz.
- Errores en estructuras de datos o en accesos a bases de datos externas.
- Errores de rendimiento.
- Errores de inicialización y de terminación.

3.1.2.1 Descripción de los casos de prueba

Cada historia de usuario está asociada a una prueba funcional, las mismas se realizan en esta etapa del proyecto y en ellas se describen las posibles formas de utilización del software. Las pruebas funcionales no solo validan la transformación de una entrada en una salida, sino que validan una característica completa.

En estos documentos de prueba se indican las posibles respuestas que tiene el software en la utilización de cada funcionalidad, así como los posibles mensajes de error, información o de aceptación que emite el software cuando se utiliza dicha funcionalidad (José Canós, 2003).

A continuación, se ejemplifica la descripción del caso de prueba de la historia de usuario Cargar experimentos:

Escenario	Descripción	Observaciones	Respuesta del sistema	Flujo central
1. Cargar experimentos.	La historia de usuario comienza cuando inicia el programa, este debe cargar desde el archivo “experiments/” todos los experimentos guardados anteriormente.	El archivo “experiments/” contiene los experimentos almacenados para Windows y Linux, sólo deben cargarse los experimentos que corresponden al sistema operativo actual.	Se cargan los experimentos correspondientes.	Al iniciar el programa, se cargan los experimentos guardados.

Tabla 16: Descripción del caso de prueba para la historia de usuario: Cargar experimentos.

Para la aplicación se realizaron 9 diseños de casos de prueba (DCP) uno por cada requisito funcional definido, los cuales se basan en una evaluación de las clases de equivalencia para una condición de entrada. Una clase de equivalencia representa un conjunto de estados válidos o inválidos para condiciones de entrada. Regularmente, una condición de entrada es un valor numérico específico, un rango de valores, un conjunto de valores relacionados o una condición lógica (Pressman, 2010).

A continuación, se muestra el comportamiento de las no conformidades (NC) según las iteraciones de pruebas. En la primera iteración se detectaron un total de 5 no conformidades de las cuales 3 son significativas y 2 no significativas. En la segunda iteración se resolvieron las anteriores, encontrándose 2 nuevas no conformidades significativas y una no significativa, quedando resueltas todas en la tercera iteración.

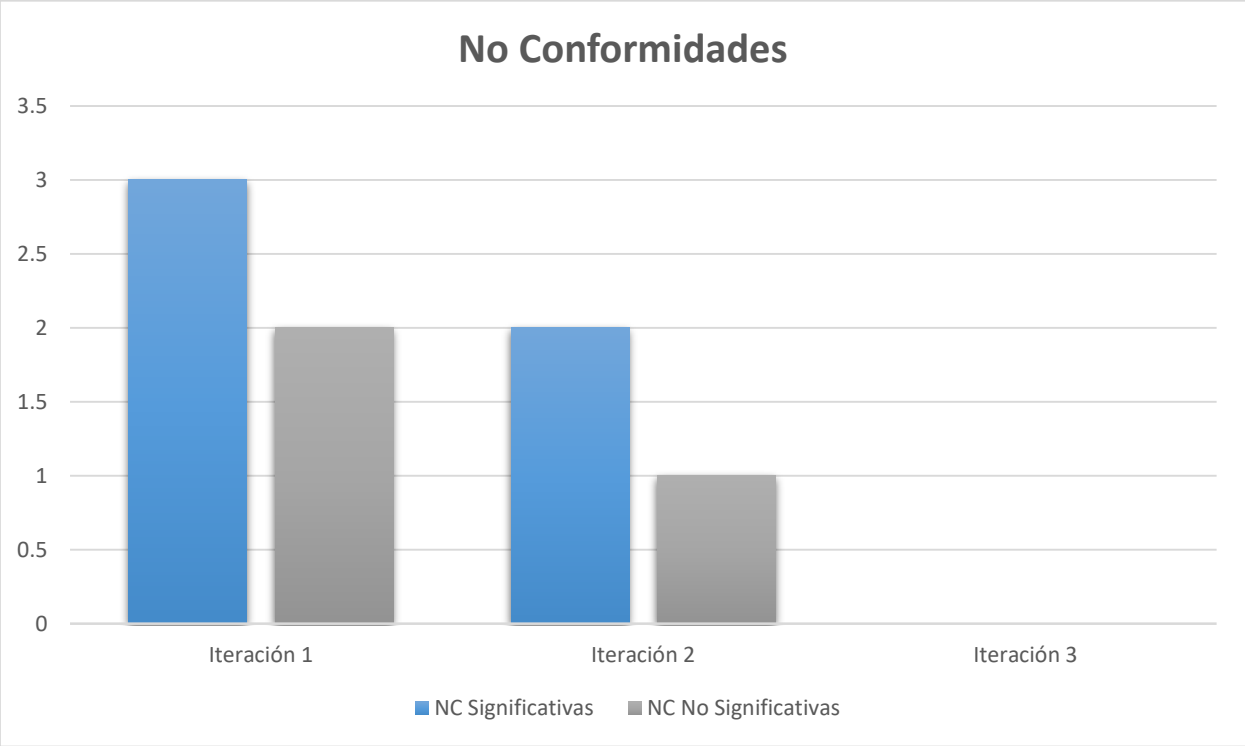


Ilustración 8: No Conformidades.

3.1.2.2 Casos de estudios

Se decide aplicar la plataforma casos de estudio con el objetivo de validar las funcionalidades de la misma. El primer caso de estudio consta de un experimento donde se utilizaron 2 versiones del algoritmo Apriori y 3 datasets obtenidos del repositorio de datasets de la Universidad de California en Irvine.

En la siguiente tabla se muestra información sobre los datasets:

Nombre	Descripción	Número de instancias	Atributos
iris	Clasificación de la planta Iris.	150	6
hr	Decidir si se contrata o no a una persona.	12	5
car	Clasificación de automóviles.	1728	7

A continuación se muestra el proceso realizado:

1) Creación de un nuevo experimento.

   Debe seleccionarse al menos un algoritmo, al menos un dataset y una vista, además se debe proporcionar un nombre para el experimento:

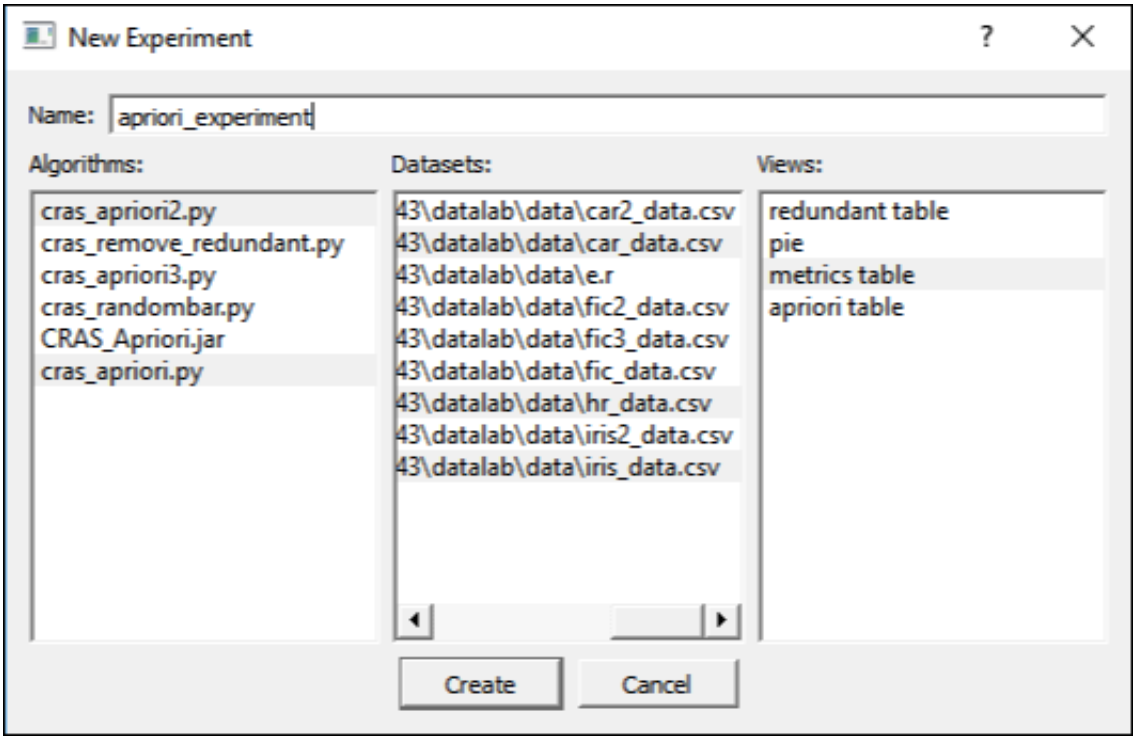


Ilustración 9: Creación de un nuevo experimento.

2) Proporcionar datos de entrada:

   Se deben proporcionar los datos de entrada para cada uno de los algoritmos seleccionados, a continuación se muestra una captura para el primer parámetro del primer algoritmo.

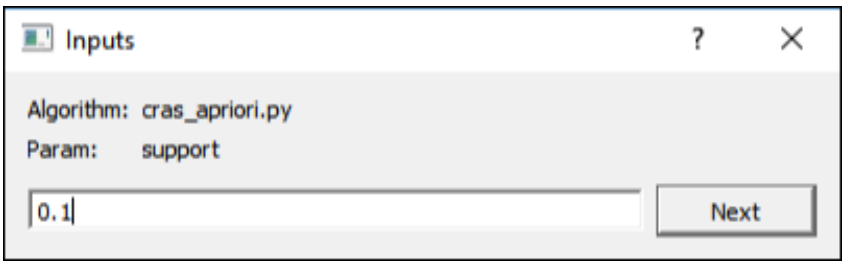


Ilustración 10: Proporcionar datos de entrada.

3) Seleccionar Experimento:

   Se selecciona el experimento que se desea ejecutar y se hace click en el botón de ejecutar experimento:

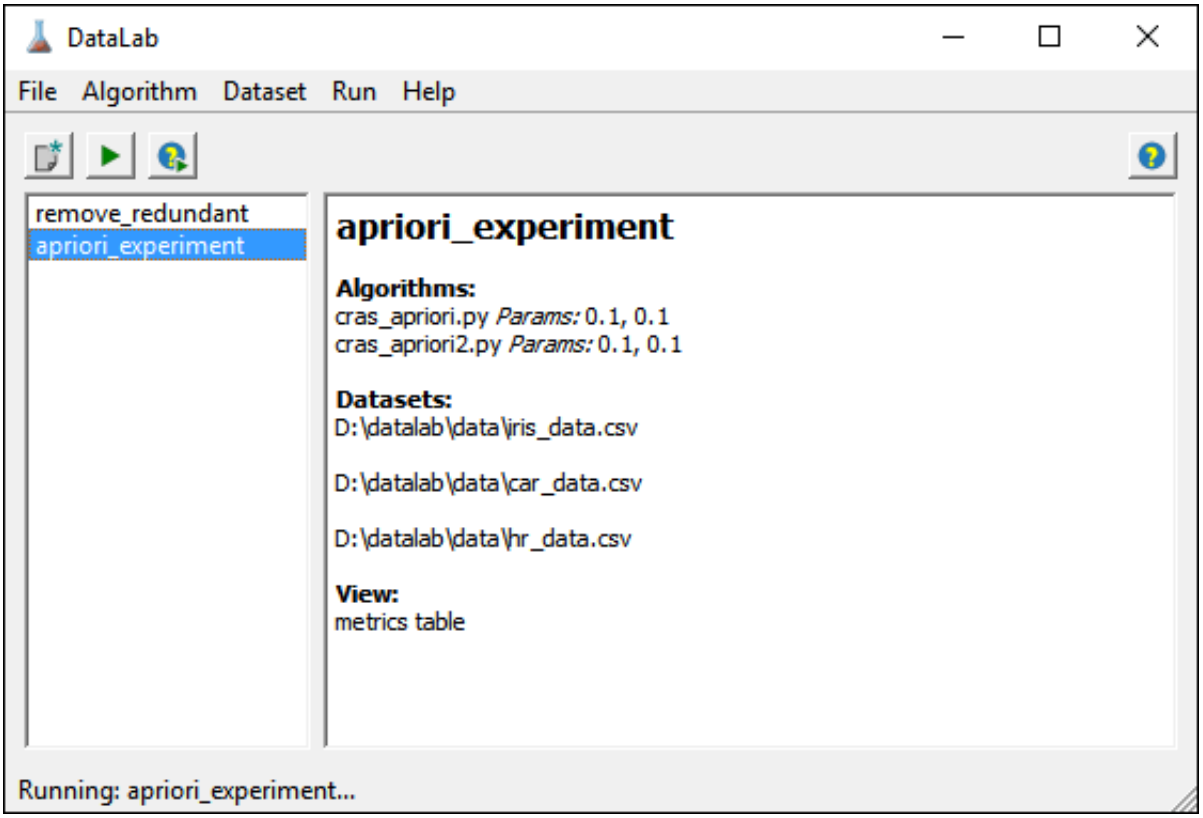


Ilustración 11: Seleccionar Experimento.

4) Luego se muestran los resultados en la vista del experimento ejecutado:

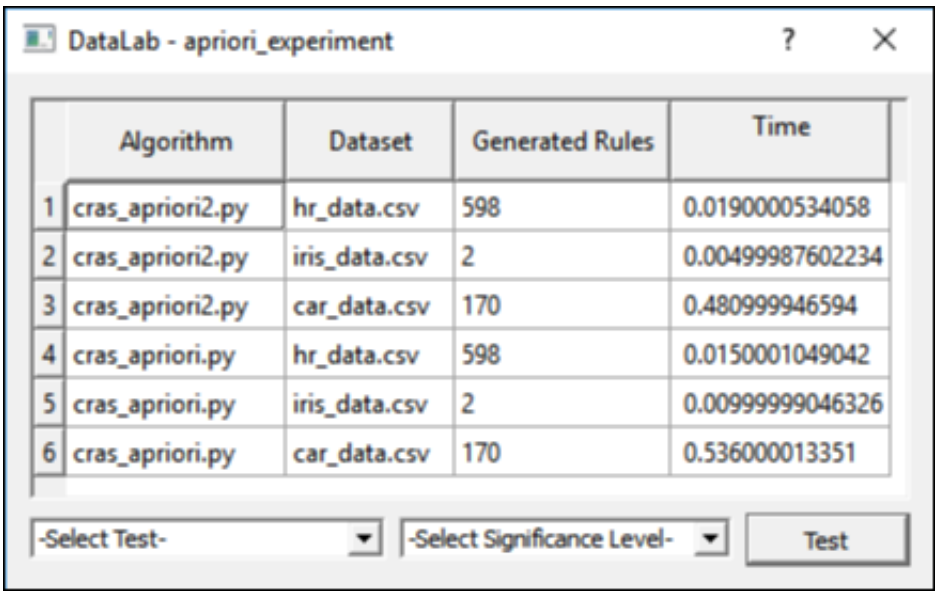


Ilustración 12: Vista con los resultados del experimento "apriori_experiment".

En el caso de esta vista es posible aplicar test estadísticos para comprobar si existen diferencias significativas entre el tiempo de ejecución de los dos algorimos del experimento, para esto se selecciona uno de los test disponibles y el nivel de significancia y luego se presiona el botón “Test”. A continuación se muestra el resultado del test para la prueba del signo para muestras pareadas con un nivel de significancia de 0.01:

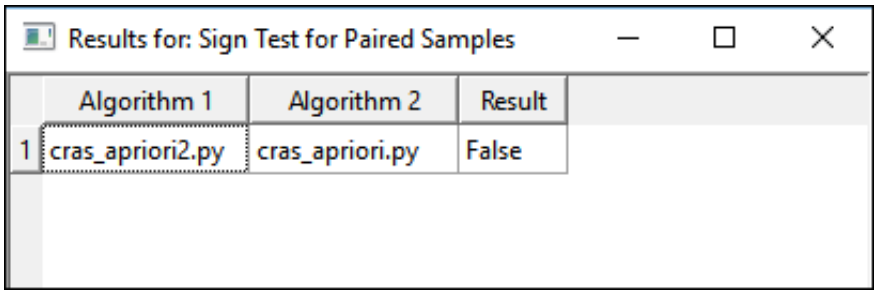
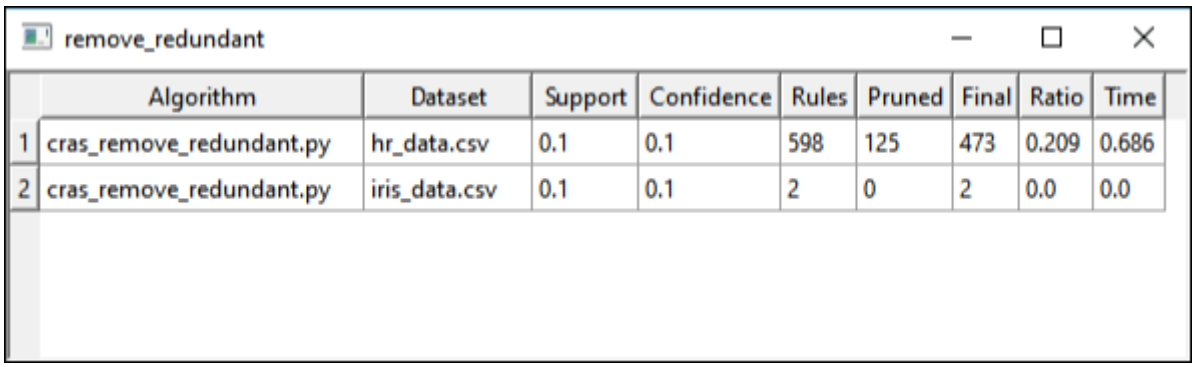


Ilustración 13: Resultado de la prueba del signo para muestras pareadas.

El segundo caso de estudio consta de un experimento donde se utilizó un algoritmo para reducción de redundancia en reglas de asociación con conocimiento del experto y los datasets iris y hr, se utilizó como conocimiento del experto un conjunto de reglas de asociación de producción propia.

A continuación se muestran los resultados del proceso realizado:

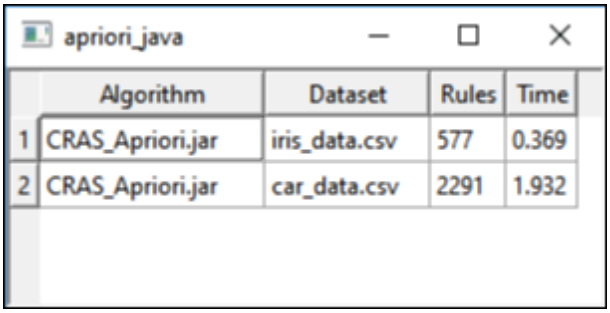

remove_redundant

	Algorithm	Dataset	Support	Confidence	Rules	Pruned	Final	Ratio	Time
1	cras_remove_redundant.py	hr_data.csv	0.1	0.1	598	125	473	0.209	0.686
2	cras_remove_redundant.py	iris_data.csv	0.1	0.1	2	0	2	0.0	0.0

Ilustración 14: Vista con los resultados para el caso de estudio de reducción de redundancia.

El tercer caso de estudio consta de un experimento donde se utilizó el algoritmo Apriori implementado en Java con un soporte de 0.01, y los datasets iris y car.

A continuación se muestran los resultados del proceso realizado:


apriori_java

	Algorithm	Dataset	Rules	Time
1	CRAS_Apriori.jar	iris_data.csv	577	0.369
2	CRAS_Apriori.jar	car_data.csv	2291	1.932

Ilustración 15: Vista con los resultados del experimento "apriori_java".

Después de realizarse los casos de estudio pudo comprobarse la facilidad para realizar experimentos sobre los algoritmos de extracción de reglas de asociación permitiendo disminuir el tiempo necesario en las investigaciones para comparar dos propuestas de estos algoritmos.

3.2 Conclusiones del capítulo

Se realizó la validación a la solución propuesta a través de las pruebas de aceptación y pruebas unitarias empleando los métodos de caja negra y caja blanca, demostrando que después de tres iteraciones, el sistema quedó libre de no conformidades, por lo que el cliente puede utilizarlo en un entorno real dado que las diferentes interfaces gráficas permiten cumplir con el objetivo propuesto. Además se ejecutaron pruebas con tres casos de estudio, generando para cada uno la salida del sistema. Después de realizada las pruebas de aceptación el cliente evidenció su satisfacción con el producto desarrollado.

Conclusiones Generales

Como resultado de la presente investigación se obtuvo una plataforma que permite reducir el tiempo necesario de los experimentos para comparar algoritmos en el área de las reglas de asociación. Además se pudieron arribar a las siguientes conclusiones:

- Se estableció el marco conceptual de referencia para desarrollar la investigación en el que se determinó que los algoritmos para la extracción de reglas de asociación poseen una complejidad computacional elevada. Permitiendo hacer un estudio de las metodologías de desarrollo de software llegando a la conclusión de que la metodología XP presenta varias ventajas cuando se trata de un equipo de desarrollo pequeño y existe una probabilidad alta a los cambios en los requisitos.
- Se realizó el análisis, diseño e implementación de la plataforma para el desarrollo de experimentos en el área de las reglas de asociación demostrando que con modificaciones mínimas es posible adaptar los algoritmos existentes en minería de reglas de asociación de forma tal que puedan ser agregados a la plataforma. Mediante la descripción de las nueve historias de usuario divididas en tres iteraciones se logró una buena organización del trabajo y se establecieron diez semanas como tiempo de duración para las tres iteraciones. La utilización de la arquitectura basada en eventos permitió una buena estructuración de la aplicación.
- Se realizó la validación a la solución propuesta a través de las pruebas de aceptación y pruebas unitarias empleando los métodos de caja negra y caja blanca, demostrando que después de tres iteraciones, el sistema quedó libre de no conformidades, por lo que el cliente puede utilizarlo en un entorno real dado que las diferentes interfaces gráficas permiten cumplir con el objetivo propuesto. Además se ejecutaron pruebas con tres casos de estudio, generando para cada uno la salida del sistema. Después de realizada las pruebas de aceptación el cliente evidenció su satisfacción con el producto desarrollado.

Recomendaciones

Con el objetivo de mejorar los resultados alcanzados en la presente investigación se muestran las siguientes recomendaciones:

- Investigar sobre los diferentes mecanismos estadísticos para la validación de experimentos.
- Investigar sobre formas de representación visual que mejore el entendimiento de los resultados que se obtienen de los algoritmos de extracción de reglas de asociación.
- Modificar y agregar algoritmos existentes a la plataforma.
- Agregar a la plataforma técnicas de preprocesamiento de los datos.

Referencias Bibliográficas

[En línea] http://www.lsi.us.es/docencia/get.php?id=361.

[En línea] https://ljk.imag.fr/membres/Bernard.Ycart/emel/cours/ts/ts.html.

[En línea] https://rafalafena.files.wordpress.com/2010/11/significacion-estadistica.doc.

[En línea] https://support.minitab.com/es-mx/minitab/18/help-and-how-to/statistics/basic-statistics/supporting-topics/tests-of-means/what-is-a-z-test/.

A Comparative Analysis of Association Rules Mining Algorithms. **Khurana, Komal y Sharma, Mrs. Simple. 2013.** 2013, International Journal of Scientific and Research Publications, Vol. 3.

Agrawal, R., Imielinksi, T. y Swami, A. 1993. *Mining association rule between sets of items in large databases.* 1993.

Azure, Microsoft. Microsoft Azure. [En línea] [Citado el: 15 de 6 de 2018.] https://docs.microsoft.com/es-es/azure/architecture/guide/architecture-styles/event-driven.

Badii, M.H., A. Guillen, L.A. Araiza, E. Cerna, J. Valenzuela & J. Landeros. 2012. *Non Parametric Methods of Common Usage.* 2012.

Beck, Kent. 1999. *Extreme Programming Explained.* 1999.

Berzal, F., et al. 2001. *A new framework to assess association rules. Proceedings of the 4th International Conference on Intelligent Data Analysis.* 2001.

2014. Buenas Tareas. [En línea] 2 de 11 de 2014. [Citado el: 5 de 6 de 2017.] http://www.buenastareas.com/ensayos/Plataformas-Abiertas-y-Cerradas/56542968.html.

codificación, Estándares de. Estándares de codificación. [En línea] [Citado el: 21 de 02 de 2018.] https://docs.google.com/document/d/1rbxDFM0zsbFDNRZeM2FoXfRDbYSiSt6tCdbYPA0qdzs/edit?hl=en_US#bookmark=id.6e6a203b40fe..

D. Bustamante, J. Rodríguez. 2014. *Metodología Actual:Metodología XP.* 2014.

E. Gamma, R. Helm, J. Vlissides. 1995. *Elements of Design Patterns. Addison-Wesley.* 1995.

Foundation, Python Software. python. [En línea] [Citado el: 4 de 6 de 2018.] https://www.python.org/dev/peps/pep-0008/.

García, Francisco Juárez, Velázquez, Jorge A. Villatoro y Lugo, Elsa Karina López. 2002. *Apuntes de Estadística Inferencial.* 2002.

Generación de conjuntos de ítems y reglas de asociación. **Pagola, J. E. 2007.** 2007.

Houtsma, H. and Swami, A. 1993. *Set Oriented Mining of Association Rules.* 1993.

Isidro Ramos Salavert, María Dolores Lozano Pérez. 2000. *Ingeniería del software y bases de datos: tendencias actuales.* 2000.

J Gutierrez, M Escalona, M Mejías, J Torres. 2010. *Pruebas del sistema en programación extrema.* s.l. : Universidad se Sevilla, 2010.

José Canós, Patricio Letelier, M.Carmen Panadés. 2003. *Metodologías Ágiles en el Desarrollo de Software.* Universidad Politécnica de Valencia : s.n., 2003.

Joskowics, J. 2008. *Reglas y prácticas en eXtreme Programming.* 2008.

Kent, B. y Fowler, M. 2000. *Planning Extreme Programming.* s.l. : Addison Wesley, 2000.

Larman, Craig. 2003. *UML y Patrones.* Madrid : s.n., 2003.

Larose, D. T. 2005. *Discovering Knowledge in Data: An Introduction to Data Mining.* 2005.

Lidia Fuentes, José M. Troya, Antonio Vallecillo. *Desarrollo de Software Basado en Componentes.*

Ltd, Visual Paradigm International. Soft 112. [En línea] https://visual-paradigm-for-uml-professional.soft112.com/.

Méndez, G. 2008. *Ingeniería de requisitos.* Madrid : s.n., 2008.

Mendoza, M. A. Sánchez. 2004. *Metodologías de desarrollo de software.* 2004.

—. 2004. *Metodologías de desarrollo de software.* 2004.

Mercedes Vitturini, Karina M Cenci, Silvia Mabel Castro. 2000. *Visualización de grandes volúmenes de datos.* 2000.

Mühlrad, D. 2008. *Patrones de diseño.* 2008.

N. Juristo, S. Vegas. 2006. *Técnicas de evaluación de software.* 2006.

Naranjo, David. 2018. DESDE LINUX. [En línea] 26 de 4 de 2018. [Citado el: 21 de 6 de 2018.] https://blog.desdelinux.net/pycharm-un-entorno-de-desarrollo-para-python/.

Obtención de conjuntos frecuentes usando computo reconfigurable. **Alejandro Mesa Rodriguez, y otros. 2009.** 2009, Serie Gris.

Pressman, Roger S. 2010. *Ingeniería del Software. Un Enfoque Práctico.* 2010.

R. Grau, K. Lauenroth. 2010. *Requirements engineering and agile development.* 2010.

Reducción de Redundancia en Reglas de Asociación. **Vera, Julio Diaz, Fernández, Carlos Molina y Miranda, María-Amparo Vila. 2016.** 2016, Revista Cubana de Ciencias Informáticas.

Rodríguez, Frank. 2008. *ESTUDIO DE MÉTODOS NO PARAMÉTRICOS.* Caracas : s.n., 2008.

Rokach, Oded Maimon and Lior. 2010. *Data Mining and Knowledge Discovery Handbook.* New York : s.n., 2010.

S. Casas, H. Reinaga. 2009. *Aspectos tempranos: un enfoque basado en Tarjetas CRC.* 2009.

Sampling Large Databases for Association Rules. **Toivonen, H. 1996.** 1996.

2018. Significados. *Significados.* [En línea] 1 de 6 de 2018. [Citado el: 7 de 6 de 2018.] https://www.significados.com/algoritmo/.

Sommerville. 2005. *Ingeniería de software.* España : s.n., 2005.

Stellman, Andrew y Greene, Jennifer. 2006. *Applied Software Project Management.* 2006.

Suarez, M. 2017. Competencias en TIC: Colección de Fasículos Digitales. [En línea] 10 de 12 de 2017. http://competenciastic.educ.ar/pdf/lenguajes_de_programacion_1.pdf.

Triantaphyllou, Evangelos. 2010. *DATA MINING AND KNOWLEDGE DISCOVERY VIA LOGICBASEDMETHODS.* 2010.

Vera, Julio César Díaz. 2017. *Técnicas basadas en conocimiento para la reducción de redundancia en reglas de asociación.* 2017.

Historias de usuario

Historia de usuario	
Número: 2	Nombre: Cargar algoritmos.
Modificación a la Historia de Usuario: 0	
Usuario: Guillermo Negrín Ortíz	Iteración asignada: 1
Prioridad en negocio: Alta	Puntos estimados: 1 semana
Riesgo en desarrollo: Alto	Puntos reales:
Programador responsable: Ariandi Campos Rodríguez	
Descripción: La historia de usuario comienza al iniciar el programa que es donde se cargan los datos necesarios para el funcionamiento de la aplicación. Debe cargar los algoritmos que se encuentran guardados en el archivo “algorithms” que se encuentra en la carpeta de la aplicación.	
Observaciones:	

Tabla 17: Historia de Usuario 2

Historia de usuario	
Número: 3	Nombre:Cargar datasets.
Modificación a la Historia de Usuario: 0	
Usuario: Guillermo Negrín Ortíz	Iteración asignada: 1
Prioridad en negocio: Alta	Puntos estimados: 1 semana
Riesgo en desarrollo: Alto	Puntos reales:
Programador responsable: Ariandi Campos Rodríguez	
Descripción: La historia de usuario comienza al iniciar el programa	

que es donde se cargan los datos necesarios para el funcionamiento de la aplicación. Debe cargar los datasets que se encuentran guardados en el archivo “algorithms/data/” que se encuentra en la carpeta de la aplicación.
Observaciones:

Tabla 18: Historia de Usuario 3

Historia de usuario	
Número: 4	Nombre:Ejecutar experimento.
Modificación a la Historia de Usuario: 0	
Usuario: Guillermo Negrín Ortíz	Iteración asignada: 2
Prioridad en negocio: Alta	Puntos estimados: 2 semana
Riesgo en desarrollo: Alto	Puntos reales:
Programador responsable: Ariandi Campos Rodríguez	
Descripción: La historia de usuario comienza cuando el usuario presiona el botón o el elemento de menú de ejecutar experimento. Debe ejecutarse el experimento seleccionado, mostrando los resultados.	
Observaciones:	

Tabla 19: Historia de Usuario 4

Historia de usuario	
Número: 5	Nombre:Importar experimento.
Modificación a la Historia de Usuario: 0	
Usuario: Guillermo Negrín Ortíz	Iteración asignada: 2
Prioridad en negocio: Alta	Puntos estimados: 1 semana
Riesgo en desarrollo: Alto	Puntos reales:

Programador responsable: Ariandi Campos Rodríguez
Descripción: La historia de usuario comienza cuando el usuario presiona el elemento de menú de importar experimento. Debe mostrar un cuadro de diálogo para elegir un archivo para luego cargar el experimento seleccionado desde ese archivo hacia el programa.
Observaciones: El experimento a importar debe ser validado.

Tabla 20: Historia de Usuario 5

Historia de usuario	
Número: 6	Nombre:Exportar experimento.
Modificación a la Historia de Usuario: 0	
Usuario: Guillermo Negrín Ortíz	Iteración asignada: 2
Prioridad en negocio: Alta	Puntos estimados: 1 semana
Riesgo en desarrollo: Alto	Puntos reales:
Programador responsable: Ariandi Campos Rodríguez	
Descripción: La historia de usuario comienza cuando el usuario presiona el elemento de menú de exportar experimento. Debe mostrar un cuadro de diálogo para elegir un archivo para luego guardar el experimento seleccionado en ese archivo.	
Observaciones:	

Tabla 21: Historia de Usuario 6

Historia de usuario	
Número: 7	Nombre:Importar algoritmo.
Modificación a la Historia de Usuario: 0	
Usuario: Guillermo Negrín Ortíz	Iteración asignada: 2

Prioridad en negocio: Alta	Puntos estimados: 1 semana
Riesgo en desarrollo: Alto	Puntos reales:
Programador responsable: Ariandi Campos Rodríguez	
Descripción: La historia de usuario comienza cuando el usuario presiona el elemento de menú de importar algoritmo. Debe mostrar un cuadro de diálogo para elegir un archivo para luego cargar el algoritmo seleccionado desde ese archivo hacia el programa.	
Observaciones:	

Tabla 22: Historia de Usuario 7