Universidad de las Ciencias Informáticas

Facultad 3

Extracción de reglas de asociación a partir de itemsets cerrados frecuentes e itemsets maximales frecuentes.

Trabajo de Diploma para optar por el título de

Ingeniero Informático

Autor: Adrian Batista López

Tutores: Ing. Guillermo Manuel Negrín Ortiz

MSc. Julio César Díaz Vera

La Habana, noviembre 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 _____.

_____

Adrian Batista López

Autor

_____ _____

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

Tutor **Tutor**

Frase

Tanto si piensas que puedes, como si piensas que no puedes, estás en lo cierto.

Henry Ford

AGRADECIMIENTOS

A mi familia.

A mis tutores.

A mi discípulo y eterno segundo lugar del tuxkart Ariandi.

A todos los profesores.

A mis compañeros de aula y amigos.

A mis hermanos Alberto, Luis y Pang.

Muchas gracias.

DEDICATORIA

A mis padres.

A mi hermano y mi cuñada.

A mi tía favorita Daysi.

RESUMEN

Los resultados de los algoritmos de extracción normalmente generan un volumen que es muy difícil de manejar por parte de los especialistas humanos provocando que se complejice la utilización de las mismas en el proceso de toma de decisiones. Existen varias alternativas encaminadas a disminuir la cantidad de reglas presentes en los modelos de reglas de asociación. Los dos representantes fundamentales de esta idea son los itemsets cerrados y los itemset maximales.

Este trabajo tiene como objetivo desarrollar una aplicación que permita comparar el tamaño de los modelos de reglas de asociación generados utilizando itemsets maximales e itemsets cerrados, teniendo en cuenta la misma entrada.

Se establece un 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.

PALABRAS CLAVE

Reglas de Asociación, Itemset Frecuentes Maximales, Itemset Frecuentes Cerrados, Itemset Frecuentes, Minería de Datos.

ÍNDICE

DECLARACIÓN JURADA DE AUTORÍA II

INTRODUCCIÓN 9

CAPÍTULO 1: FUNDAMENTACIÓN TEÓRICA 13

1.1 Aplicación 13
1.2 Taxonomía de tipos de software 14
1.3 Minería de datos 17
1.4 Regla de asociación 18
1.5 Principales conceptos asociados a las reglas de asociación 19
1.5.1 Itemset 19
1.5.2 Medidas de evaluación de las reglas de asociación 19
1.5.3 Frecuencia de un itemset 19
1.5.4 Itemset frecuente(IF) 19
1.5.5 Apriori 20
1.5.6 Conjunto de itemsets frecuentes 20
1.5.7 Subconjunto 20
1.5.8 Superconjunto 20
1.5.9 Superconjunto propio 21
1.5.10 Itemsets maximales frecuentes(IMF) 21
1.5.11 Itemsets cerrados frecuentes(ICF) 21
1.5.12 Propiedad de los Itemsets frecuentes cerrados y maximales 21
1.6 Desarrollo de software 22
1.6.1 Proceso de desarrollo de software 22
Conclusiones del capítulo 24
CAPÍTULO 2: ANÁLISIS Y DISEÑO DE LA PROPUESTA DE SOLUCIÓN 25

2.1 Metodología de desarrollo 25
2.1.1 Valores de la metodología XP 25
2.1.2 Ventajas de la metodología XP 27
2.1.3 Fases de la metodología XP 27
2.2 Descripción de la solución 28
2.2.1 Fase de planeación 28
2.2.2 Descripción de los procesos del negocio 28

2.2.3 Especificación de los requisitos 28
2.2.4 Descripción de historias de usuarios 29
2.2.5 Fase de Planificación de la Entrega 30
2.2.6 Plan de iteraciones 31
2.3 Arquitectura basada en eventos 31
2.4 tarjetas CRC 32
2.5 Estándares de codificación 33
2.6 Fase de implementación 33
2.7 Entorno de desarrollo integrado 33
2.8 Lenguaje de programación 34
2.9 Tareas de desarrollo 34
2.10 Diagrama de clases 36
2.10.1 Clase Apriori 37
2.10.2 Clase Mace 39
2.10.3 Clase Mcet 39
2.10.4 Clase Dataset 40
2.10.5 Interfaz de usuario 41
Conclusiones del capitulo 42
CAPÍTULO 3: EVALUACIÓN DE LA SOLUCIÓN PROPUESTA 43
3.1 Validación del sistema 43
3.1.2 Pruebas de caja blanca 43
3.1.3 Pruebas unitarias 43
3.1.4 Prueba del camino Básico 44
3.2.1 Pruebas de caja negra 46
3.2.2 Descripción de los casos de prueba 46
3.2.3 Casos de estudios 48
3.2.4 Casos de estudio 51
Conclusiones del capítulo 53
CONCLUSIONES 54
ANEXOS 55
Historias de usuario 55
BIBLIOGRAFÍA 58

ÍNDICE FIGURAS

Ilustración 1: **Taxonomía con predominio de cómputo** (Samuel Ojeda Pereira 2017)**.**..... 16
Ilustración 2: Diagrama de clases 37
Ilustración 3: Clase Apriori..... 38
Ilustración 4: Clase Mace..... 39
Ilustración 5: Clase Mcet..... 40
Ilustración 6: Clase Dataset 41
Ilustración 7: Interfaz de usuario 42
Ilustración 8: Representación del grafo de flujo de camino básico..... 45
Ilustración 9: Casos de prueba 45
Ilustración 10: No Conformidades...... 48
Ilustración 11: Datasets del experimento..... 48
Ilustración 12: Interfaz principal 49
Ilustración 13: Interfaz principal con datases cargados y procesados 50
Ilustración 14: Comparación de los resultados...... 51
Ilustración 15: Itemsets cerrados obtenidos...... 52
Ilustración 16: Itemsets maximales. 53

ÍNDICE DE TABLAS

Tabla 1: Historia de usuario 1. 30
Tabla 2: Estimación de tiempo por historia de usuario 30
Tabla 3: Estimación de tiempo por iteración 31
Tabla 4: Tarjeta CRC de la clase mace 32
Tabla 5: Tarjeta CRC de la clase apriori 32
Tabla 6: Tarjeta CRC de la clase fcm 33
Tabla 7: Tabla de ingeniería de la funcionalidad Cargar datasets 34
Tabla 8: Tabla de ingeniería de la funcionalidad Preprocesamiento de datasets 35
Tabla 9: Tabla de ingeniería de la funcionalidad Extraer itemsets frecuentes 35
Tabla 10: Tabla de ingeniería de la funcionalidad Extraer itemsets maximales frecuentes . 35
Tabla 11: Tabla de ingeniería de la funcionalidad Extraer itemsets cerrados frecuentes 36
Tabla 12: Tabla de ingeniería de la funcionalidad comparar resultados 36
Tabla 13: Descripción del caso de prueba para la historia de usuario: Cargar dataset. 47
Tabla 14: Historia de Usuario 2 55
Tabla 15: Historia de Usuario 3 55
Tabla 16: Historia de Usuario 4 56
Tabla 17: Historia de Usuario 5 56
Tabla 18: Historia de Usuario 6 56

INTRODUCCIÓN

Los datos son la materia prima que se obtiene a partir de las operaciones de los negocios. En el momento que el usuario les atribuye algún significado especial pasan a convertirse en información. Cuando los especialistas elaboran o encuentran un modelo, haciendo que la interpretación que surge entre la información y ese modelo represente un valor agregado, entonces se refiere al conocimiento. (Carrion 2017)

Los avances acaecidos en las ciencias informáticas han posibilitado que el proceso de transformación de datos en conocimiento ocurra de manera automática. La minería de datos es una de las tecnologías utilizadas con este fin.

Es un hecho que en los últimos años el volumen de los datos almacenados ha crecido de forma considerable, ya sea por la informatización de las tareas o por el almacenamiento de la información en formato digital. Contar con registros históricos ha incrementado el interés por poder utilizar estos datos para obtener información desconocida y de valor para producir un impacto positivo sobre la organización. Este fenómeno ha favorecido el surgimiento de nuevas herramientas que permiten manejar grandes volúmenes de datos y a su vez adquirir información oculta en ellos que pueda ser de utilidad en algún sentido. Estas herramientas forman parte del campo de la Extracción de Conocimiento.(Fuentes Herrera 2016)

Los conjuntos frecuentes desempeñan un papel esencial en muchas tareas de minería de datos que intentan encontrar patrones interesantes de bases de datos, como reglas de asociación, correlaciones, secuencias, episodios, clasificadores y clústeres. La extracción de reglas de asociación es uno de los problemas más populares de todos estos. La identificación de conjuntos de artículos, productos, síntomas y características, que a menudo ocurren juntos en la base de datos dada, puede verse como una de las tareas más básicas en la Minería de Datos.

La motivación original para buscar conjuntos frecuentes provino de la necesidad de analizar los llamados datos de transacciones de supermercados, es decir, examinar el comportamiento del cliente en términos de los productos comprados. Los conjuntos frecuentes de productos describen con qué frecuencia se compran los artículos juntos (Han, Pei y Kamber 2011).

La minería de datos, es el conjunto de técnicas y tecnologías que permiten explorar grandes bases de datos, de manera automática o semiautomática, con el objetivo de encontrar patrones repetitivos, tendencias o reglas que expliquen el comportamiento de los datos en un determinado contexto.(Félix y Carlos 2002)

Una de las técnicas más estudiadas y utilizadas en minería de datos es el minado de reglas de asociación. Las mismas pueden ser aplicadas en diferentes áreas tales como: medicina, biotecnología y seguridad.

Desde el punto de vista semántico una regla de asociación tiene una estructura si-entonces. Este tipo de reglas son fundamentales en los sistemas basados en reglas gracias a que son fáciles de interpretar por agentes humanos.

Una regla de asociación representa las regularidades en forma de relaciones de implicación entre los atributos de los objetos de un conjunto de datos. De manera general la implicación se representa como X=>Y donde X se denomina antecedente de la regla mientras Y es denominado consecuente de la regla. Ambos X e Y están constituidos por conjuntos de elementos de la base de datos, no necesariamente distintos, aunque es común que no se solapen elementos entre X e Y. Una regla de asociación refleja cuánto influye la presencia del antecedente X en una transacción de la base de datos en la aparición del consecuente Y en la misma transacción (Diaz Vera, Molina Fernández y Vila Miranda 2016).

Los mecanismos de extracción de reglas de asociación funcionan en dos etapas, en la primera se encuentran los itemset frecuentes y en la segunda se generan las reglas a partir de los itemset frecuentes. Para una base de datos con n ítems se pueden generar 2^n Itemset frecuentes y $3^n - 2^{n+1} + 1$ reglas de asociación (Fernando 2006). Estas funciones exponenciales expresan la imposibilidad de utilizar algoritmos de fuerza bruta para resolver el problema. 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 (Diaz Vera, Molina Fernández y Vila Miranda 2016):

- 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.

Una de las alternativas utilizadas para disminuir la incidencia de estos problemas en la generación de reglas está asociada a la disminución del número de itemset generados, sin que esto disminuya la información obtenida. Dentro de estas alternativas una línea que ha alcanzado éxito dentro de la comunidad pretende crear representaciones más compactas de los itemset.

Los dos representantes fundamentales de esta idea son los itemsets cerrados y los itemset maximales. Debido a la incompatibilidad de las implementaciones y la no estandarización de los dataset se hace muy difícil comparar estas alternativas (Grandinetti 2005).

Para este trabajo se define como **problema a resolver**. ¿Cómo desarrollar una aplicación que permita comparar el tamaño de los modelos de reglas de asociación generados utilizando itemsets maximales e itemsets cerrados, teniendo en cuenta la misma entrada?

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

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

Teniendo como **objetivo general** desarrollar una plataforma para la realización de experimentos en el área de las reglas de asociación utilizando las técnicas de itemsets maximales e itemsets cerrados.

Objetivos específicos:

- Establecer el marco conceptual para el desarrollo de la investigación.
- Implementar la aplicación para la comparación.
- Validar la aplicación mediante casos de estudio.

Tareas de investigación:

- Estudio del marco teórico que fundamenta el objeto de investigación.
- Selección del lenguaje de programación.
- Diseño y modelación de la plataforma.

- Implementación de la plataforma.
- Validación de la solución propuesta.

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.

A continuación, se detalla la estructura que tendrá el documento.

En el **capítulo 1:** se desarrolla toda la fundamentación teórica, se hace análisis de los principales conceptos necesarios para el desarrollo de la propuesta, como son los conceptos relacionados con reglas de asociación, itemset frecuentes, ítemset cerrados, itemset maximales, minería de datos. 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 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 encuentran descritos los diferentes mecanismos utilizados para la validación de los resultados obtenidos. Entre ellos se encuentran los experimentos y el razonamiento lógico.

CAPÍTULO 1: FUNDAMENTACIÓN TEÓRICA

En este capítulo se presentan los elementos que fundamentan la base teórica conceptual necesaria para el desarrollo de la propuesta. Se estudian los enfoques y metodologías para la selección de la más apropiada y se propone por cada una de las fases de la metodología seleccionada los artefactos que se generan.

1.1 Aplicación

Este trabajo pretende construir una plataforma para la comparación de diferentes algoritmos de extracción de itemsets frecuentes. La plataforma es una aplicación y en ese sentido es necesario formalizar su definición.

El termino aplicación está estrechamente relacionado con la palabra en idioma inglés software que según (Pressman 2005) es:

- Instrucciones (programas de cómputo) que cuando se ejecutan proporcionan las características, función y desempeño buscados.
- Estructuras de datos que permiten que los programas manipulen en forma adecuada la información.

En la clasificación de software destacan, principalmente, tres tipos de software: Software de aplicación, software de sistema y software de programación. (Šmite et al. 2014)

- Software de sistema: El software de sistema es el que permite al usuario utilizar el sistema operativo incorporado en el ordenador o dispositivo en cuestión. El software de sistema lo componen una serie de programas que tienen dos objetivos, el primero es gestionar los recursos de los que dispone el hardware, pudiendo coordinar tareas, como por ejemplo la memoria, las unidades de disco, las impresoras o escáneres e, incluso, el mouse y el segundo es ofrecer una interfaz al usuario para interactuar con el sistema (Prieto, Lloris y Torres 1995).

- Software de aplicación: Se llama software de aplicación a todo programa que otorga a los usuarios la capacidad para realizar diferentes trabajos. Es decir, software de aplicación serían todos los procesadores de texto, hojas de cálculo o videojuegos (Pressman y Troya 1988).

- Software de programación: El software de programación está constituido por las herramientas utilizadas por el programador para crear programas. El programador emplea diferentes lenguajes de programación. Como por ejemplo los editores de texto o los compiladores. Por ejemplo, el programador crea el código en el editor de texto y luego lo compila (Joyanes Aguilar 2003).

Dado estas características es claramente apreciable que el software que se va a desarrollar en este trabajo clasifica como software de aplicación.

Una de las características fundamentales del software de aplicación es que manejan un requisito de alto nivel que constituye el objetivo fundamental del mismo. Para este trabajo el requisito de alto nivel definido es comparar el tamaño de los modelos de reglas de asociación generados utilizando itemsets maximales frecuentes e itemsets cerrados frecuentes.

1.2 Taxonomía de tipos de software

La taxonomía propuesta presenta cuatro niveles. En el primer nivel se agrupan cuatro categorías diferentes que se etiquetan de la **A** a la **D** y que permiten agrupar a los software teniendo en cuenta el predominio de la gestión de datos o del cómputo (Šmite et al. 2014). Las categorías definidas son las siguientes:

- A: Sistemas con predominio de los datos: en esta categoría se agrupan los sistemas de propósito general desde los orientados a consumo hasta algunos con perfiles más específicos dentro de un negocio.

- B: Software de sistemas: el centro de esta categoría se centra en el nivel de abstracción desde el bajo nivel asociado a los sistemas operativos hasta el más alto asociado a ejemplo, entornos integrados de desarrollo.

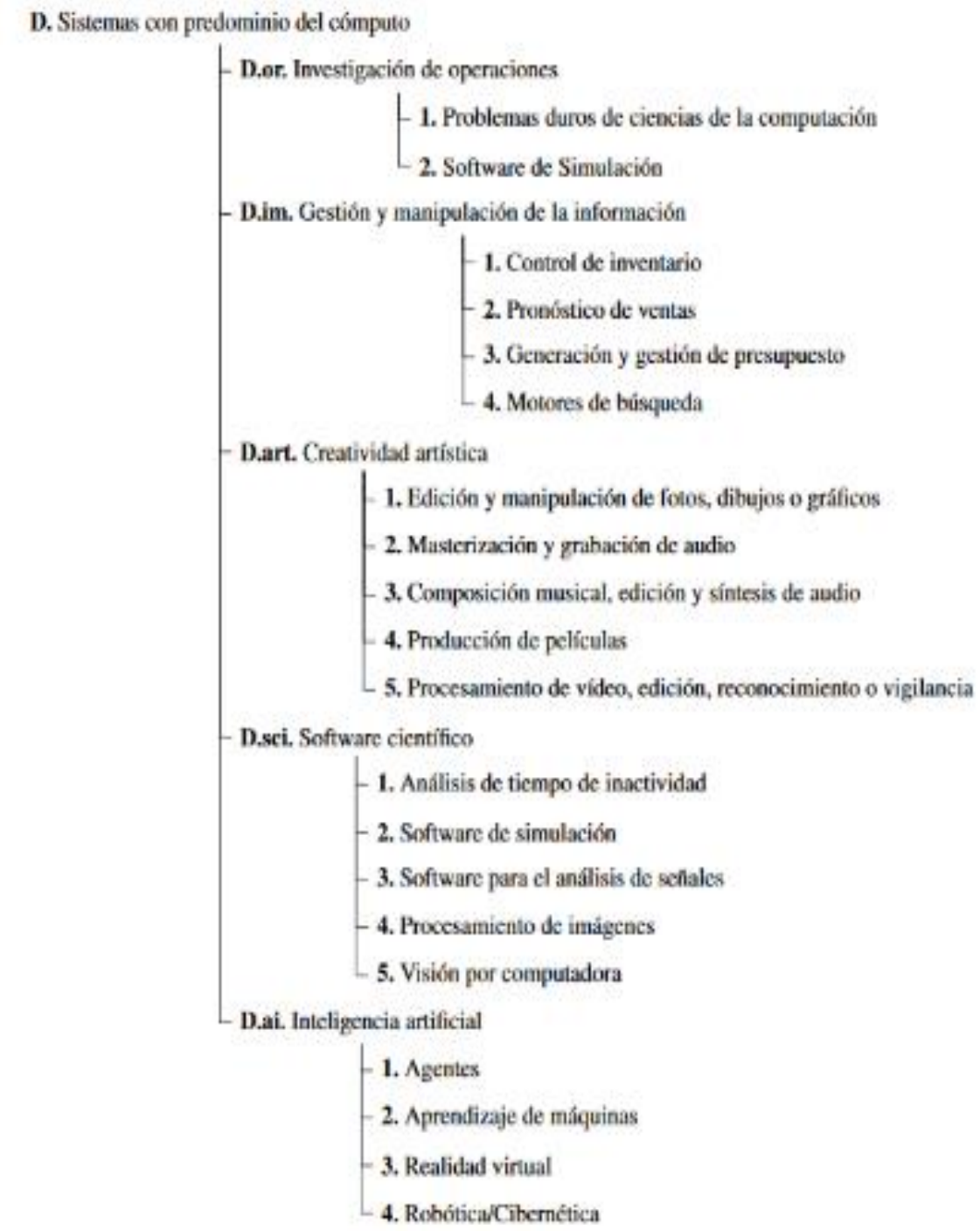
- C: Sistemas con predominio del control: estos se centran en cuánto es necesario controlar el sistema desde pequeños programas embebidos hasta grandes sistemas que gestionan y controlan operaciones complejas como los SCADA.

- D: Sistemas con predominio del cómputo: el eje central de esta línea está en cuán conceptual es el problema desde aspectos prácticos hasta áreas totalmente teóricas como la inteligencia artificial.

El problema planteado en este trabajo clasifica dentro del item D, un sistema donde predomina el cómputo. A continuación, se describe la taxonomía de la rama propuesta.

Según la taxonomía planteada, en el segundo nivel se utilizan dos o tres letras minúsculas que representan un código nemotécnico para la categoría. El tercer nivel se etiqueta con un número mientras que, el cuarto, es etiquetado con una letra minúscula. Es necesario puntualizar que para el software de predominio de cómputo la taxonomía tiene tres niveles como puede apreciarse a continuación.

Ilustración 1: **Taxonomía con predominio de cómputo** *(Samuel Ojeda Pereira 2017)***.**



El cómputo de reglas de asociación es una de las áreas más estudiadas y aplicadas de la minería de datos, una rama multidisciplinaria de las Ciencias de la Computación que a juicio de varios investigadores tiene una estrecha relación con el aprendizaje de máquinas (Antony, Manujesh y Jnanesh 2016). Para la inteligencia artificial, entre el aprendizaje de máquinas y la minería de datos, es posible establecer una relación jerárquica de uso en la que:

- La minería de datos se encarga del descubrimiento de patrones ocultos en los datos, lo que permite explicar algún fenómeno. Permite intuir lo que realmente está pasando en un conjunto de datos.
- El aprendizaje de máquinas utiliza las técnicas de minería de datos y otros algoritmos de aprendizaje para construir modelos acerca de lo que está pasando en un conjunto de datos de forma que se pueda utilizar el modelo para predecir el futuro.
- La inteligencia artificial utiliza los modelos creados con aprendizaje de máquinas y otras formas de razonamiento para permitir un comportamiento inteligente en las computadoras.

De manera general el uso de la taxonomía permitirá estudiar las soluciones que han funcionado y las que no lo han hecho para este tipo particular de desarrollo de software. De esta manera se facilita la tarea de relacionar las herramientas, técnicas y métodos, que garanticen construir un software con la calidad requerida.

1.3 Minería de datos

Según (Hernández Orallo et al. 2004), es el proceso de extracción de conocimiento. La minería de datos o exploración de datos 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 (Matatov, Rokach y Maimon 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 les asignan a los nuevos registros.
- Estimación: similar a la clasificación, salvo que la variable de destino es numérica no 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 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 (Diaz Vera, Molina Fernández y Vila Miranda 2016). 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 del minado de reglas de asociación consiste en encontrar todas las reglas interesantes sobre una colección de datos, las reglas se consideran interesantes cuando tienen una frecuencia de aparición mayor que un umbral definido.

Formalmente una regla de asociación puede definirse como:

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.(Diaz Vera, Molina Fernández y Vila Miranda 2016).

1.5 Principales conceptos asociados a las reglas de asociación

1.5.1 Itemset

Sea I el conjunto de datos de todos los ítems posibles. Llamaremos itemset o patrón a cualquier conjunto $X = \{i_1, \dots, i_k\} = \subseteq \mathrm{I}$. Si el itemset contiene k items también puede ser llamado k-itemset o itemset de cardinalidad k.(Grandinetti 2005).

1.5.2 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 (Medina Pagola 2007).

Dentro de estas medidas se encuentran el soporte.

El soporte, sop(X) de un conjunto de elementos X es la proporción de transacciones en las que aparece un conjunto de elementos con el número total de transacciones. (Pei, Han y Mao 2000).

1.5.3 Frecuencia de un itemset

La frecuencia de un itemset X en D es la probabilidad de que X ocurra en la transacción $\mathrm{T} \in \mathrm{D}$. $Frecuencia(X, D) = P(X) = \frac{Soporte(X,D)}{|D|}$ (Grandinetti 2005).

1.5.4 Itemset frecuente(IF)

Diremos que un itemset es frecuente si su soporte es mayor que un mínimo umbral de soporte absoluto δ_{abs} dado, con $0 \leq \delta_{abs} \geq |D|$. (Grandinetti 2005).

1.5.5 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, Imieliński y Swami 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.

1.5.6 Conjunto de itemsets frecuentes

Sea D una base de datos transaccional sobre un conjunto de ítems I y δ el mínimo umbral de soporte. El conjunto de ítems frecuentes en D con respecto a δ es denotado por: $F(D,\delta) = \{X \subseteq \mathrm{I} | soporte(X,D) \geq \delta\}$ (Grandinetti 2005).

1.5.7 Subconjunto

Un conjunto *A* se dice que es **subconjunto** de otro *B*, si cada elemento de *A* es también elemento de *B*, y se denota $A \subseteq B$ (Cantor y Venn 1998).

1.5.8 Superconjunto

Si *A* es un subconjunto de *B*, decimos también que *B* es un **superconjunto** de *A*, también denotado como $B \supseteq A$ (Cantor y Venn 1998).

1.5.9 Superconjunto propio

Un superconjunto propio se define como $B \supset A \iff A \subset B$, lo que significa que B es superconjunto de A siendo $A \neq B$ (Cantor y Venn 1998).

1.5.10 Itemsets maximales frecuentes(IMF)

Un ítem es máximo frecuente si ninguno de sus súperconjuntos es frecuente (Zaki y Hsiao 2002). Dada una colección C de itemset. un itemset $X \in C$ se define como maximal en C si no existe un superconjunto propio de X que esté en C.(Grandinetti 2005)

1.5.11 Itemsets cerrados frecuentes(ICF)

Es un conjunto de elementos frecuente que está cerrado y su soporte es mayor o igual al soporte mínimo. Un conjunto de elementos se cierra en un conjunto de datos si no existe un superconjunto que tenga el mismo conteo de soporte que este conjunto de elementos original (Zaki y Hsiao 2002).

1.5.12 Propiedad de los Itemsets frecuentes cerrados y maximales

Los itemsets frecuentes cerrados son una clase especial de itemsets frecuentes. Todo itemset frecuente maximal es un itemset frecuente cerrado ya que ningún superconjunto propio tiene el mismo soporte. Por lo tanto, se cumple la siguiente relación: $IMF \subseteq ICF \subseteq IF$ (Grandinetti 2005).

Pareciera que a la luz de la propiedad anterior las clases especiales de itemsets no proporcionan ninguna ayuda ya que en el peor caso los tres conjuntos son iguales $(IMF = ICF = IF)$. Sin embargo, aun cuando este caso es posible es muy improbable y en la gran mayoría de los casos tanto los ICF como los IMF son enormemente más pequeños. Aún puede suceder que el conjunto de ICF sea exponencialmente grande (Zou, Chu y Lu 2002), en tal caso el conjunto de IMF es la mejor opción (Grandinetti 2005).

1.6 Desarrollo de software

El problema de la construcción de software es tratado a nivel académico en la disciplina Ingeniería de Software (Kosa et al. 2016). Sin embargo, los principios, prácticas y postulados que conforman el cuerpo de conocimiento de esta disciplina no pueden ser adoptados simplemente porque alguien los ha enunciado antes. La adopción de los postulados de la Ingeniería de Software requiere de contexto y evidencias.
En los últimos años, ha ganado protagonismo, una línea de trabajo enfocada a la realización de investigaciones empíricas en el campo de la Ingeniería de Software (Siegmund, Siegmund y Apel 2016). Las mismas están orientadas a proporcionar el contexto necesario para la adopción de sus postulados. Este elemento ha impulsado el desarrollo de trabajos que pretenden establecer categorías para los tipos de software. La aspiración fundamental es lograr enmarcar el contexto en el que son aplicadas las evidencias empíricas. Uno de los resultados indispensables en esta área es el desarrollo de taxonomías de tipos de software que permitan establecer un vocabulario estructurado al mismo tiempo que define una teoría de acuerdo a la que se organizan las etiquetas del vocabulario (Šmite et al. 2014).

1.6.1 Proceso de desarrollo de software

En el contexto de la ingeniería de software, un proceso no es una prescripción rígida de cómo elaborar software de cómputo. Por el contrario, es un enfoque adaptable que permite que las personas que hacen el trabajo (el equipo de software) busquen y elijan el conjunto apropiado de acciones y tareas para el trabajo. Se busca siempre entregar el software en forma oportuna y con calidad suficiente para satisfacer a quienes patrocinaron su creación y a aquellos que lo usarán (Pressman 2005).

La estructura del proceso establece el fundamento para el proceso completo de la ingeniería de software por medio de la identificación de un número pequeño de actividades estructurales que sean aplicables a todos los proyectos de software, sin importar su tamaño o complejidad. Además, la estructura del proceso incluye un conjunto de actividades sombrilla que son aplicables a través de todo el proceso del software. Una estructura de proceso general para la ingeniería de software consta de cinco actividades(Pressman 2005):

Comunicación. Antes de que comience cualquier trabajo técnico, tiene importancia crítica comunicarse y colaborar con el cliente (y con otros participantes). Se busca entender los objetivos de los participantes

respecto del proyecto, y reunir los requerimientos que ayuden a definir las características y funciones del software.

Planeación. Cualquier viaje complicado se simplifica si existe un mapa. Un proyecto de software es un viaje difícil, y la actividad de planeación crea un "mapa" que guía al equipo mientras viaja. El mapa —llamado plan del proyecto de software— define el trabajo de ingeniería de software al describir las tareas técnicas por realizar, los riesgos probables, los recursos que se requieren, los productos del trabajo que se obtendrán y una programación de las actividades.

Modelado. Ya sea un diseñador de paisaje, constructor de puentes, ingeniero aeronáutico, carpintero o arquitecto, a diario trabaja con modelos. Crea un "bosquejo" del objeto por hacer a fin de entender el panorama general —cómo se verá arquitectónicamente, cómo ajustan entre sí las partes constituyentes y muchas características más—. Si se requiere, refina el bosquejo con más y más detalles en un esfuerzo por comprender mejor el problema y cómo resolverlo. Un ingeniero de software hace lo mismo al crear modelos a fin de entender mejor los requerimientos del software y el diseño que los satisfará.

Construcción. Esta actividad combina la generación de código (ya sea manual o automatizada) y las pruebas que se requieren para descubrir errores en éste.

Despliegue. El software (como entidad completa o como un incremento parcialmente terminado) se entrega al consumidor que lo evalúa y que le da retroalimentación.

Todos los modelos del proceso del software pueden incluir las actividades estructurales generales, pero cada uno pone un distinto énfasis en ellas y define en forma diferente el flujo de proceso que invoca cada actividad estructural (así cómo acciones y tareas de ingeniería de software). De aquí la necesidad de seleccionar un modelo según las características del software a desarrollar, entre estos modelos se encuentran los de proceso descriptivo y los de proceso ágil.

Los **modelos de proceso descriptivo** enfatizan la definición, la identificación y la aplicación detalladas de las actividades y tareas del proceso. Su objetivo es mejorar la calidad del sistema, desarrollar proyectos más manejables, hacer más predecibles las fechas de entrega y los costos, y guiar a los equipos de ingenieros de software cuando realizan el trabajo que se requiere para construir un sistema. Si los modelos prescriptivos se aplican en forma dogmática y sin adaptación, pueden incrementar el nivel de burocracia

asociada con el desarrollo de sistemas basados en computadora y crear inadvertidamente dificultades para todos los participantes(Pressman 2005).

Los **modelos de proceso ágil** ponen el énfasis en la “agilidad” del proyecto y siguen un conjunto de principios que conducen a un enfoque más informal (pero no menos efectivo) del proceso de software. Por lo general, se dice que estos modelos del proceso son “ágiles” porque acentúan la maniobrabilidad y la adaptabilidad. Son apropiados para muchos tipos de proyectos y son útiles en particular cuando se hace ingeniería sobre aplicaciones web(Pressman 2005).

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 define la taxonomía orientada a los software de predominio de cómputo y se describe el proceso de desarrollo de software.

CAPÍTULO 2: ANÁLISIS Y DISEÑO DE LA PROPUESTA DE SOLUCIÓN

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 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 anterior 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.

2.1.1 Valores de la metodología XP

A fin de lograr la comunicación eficaz entre los ingenieros de software y otros participantes, XP pone el énfasis en la colaboración estrecha pero informal (Verbal) entre los clientes y los desarrolladores, para comunicar conceptos importantes, en la retroalimentación continua y en evitar la documentación voluminosa como medio de comunicación.

Para alcanzar la simplicidad, XP restringe a los desarrolladores para que diseñen sólo para las necesidades inmediatas, en lugar de considerar las del futuro. El objetivo es crear un diseño sencillo que se implemente con facilidad en forma de código. Si hay que mejorar el diseño, se rediseñara en un momento posterior.

La retroalimentación se obtiene de tres fuentes: el software implementado, el cliente y otros miembros del equipo. Al diseñar e implementar una estrategia de pruebas eficaz, el software por medio de los resultados de las pruebas da retroalimentación al equipo ágil. XP usa la prueba unitaria como su táctica principal de pruebas. A medida que se desarrolla cada clase, el equipo implementa una prueba unitaria para ejecutar cada operación de acuerdo con la funcionalidad especificada. (Pressman y Troya 1988)

Características fundamentales (Beck y Gamma 2000):

- Desarrollo iterativo e incremental: Se realizan pequeñas mejoras, unas tras otras.
- Pruebas unitarias continuas, frecuentemente repetidas y automatizadas, incluyendo pruebas de regresión. Se aconseja escribir el código de la prueba antes de la codificación.
- Frecuente integración del equipo de programación con el cliente o usuario. Se recomienda que un representante del cliente trabaje junto al equipo de desarrollo.
- Corrección de todos los errores antes de añadir una nueva funcionalidad. Hacer entregas frecuentes.
- Refactorización del código, es decir, rescribir ciertas partes del código para aumentar su legibilidad y mantenibilidad, pero sin modificar su comportamiento. Las pruebas han de garantizar que en la refactorización no se ha introducido ningún fallo.
- Propiedad del código compartido: en vez de dividir la responsabilidad en el desarrollo de cada módulo en grupos de trabajo distintos, este método promueve que todo el personal pueda corregir y extender cualquier parte del proyecto. Las frecuentes pruebas de regresión garantizan que los posibles errores serán detectados.
- Simplicidad en el código: es la mejor manera de que las cosas funcionen. Cuando todo funcione se podrá añadir otra funcionalidad si es necesario. La programación extrema apuesta que es más sencillo hacer algo simple y tener un poco de trabajo extra para cambiarlo si se requiere, que realizar algo complicado y quizás nunca utilizarlo.

2.1.2 Ventajas de la metodología XP

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

- Comienza en pequeño y añade funcionalidades 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.

2.1.3 Fases de la metodología XP

XP consta de 4 fases (Bustamante y Rodríguez 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 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: La fase de codificación se desarrolla en función de cada historia de usuario, además de ser la fase donde se definen las tareas de la ingeniería y los tiempos reales en los que se realizaron cada una de las funcionalidades especificadas, en la cual la implementación, debe realizarse de acuerdo a 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.

En esta fase se realizará la prueba del camino básico, además se realizarán casos de prueba y casos de estudio.

2.2 Descripción de la solución

Una vez establecida la metodología de desarrollo XP como guía para la construcción de un software cuyo objetivo es comparar el tamaño de los modelos de reglas de asociación generados utilizando itemsets maximales y itemsets cerrados, teniendo en cuenta la misma entrada y así obtener una plataforma para el desarrollo de experimentos en el área de reglas de asociación se describen cada una de las fases de la metodología seleccionada y los artefactos generados.

2.2.1 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).

2.2.2 Descripción de los procesos del negocio

Para lograr una mejor comprensión del negocio a continuación se hace una breve descripción de las características y el funcionamiento del proceso principal que interviene en el negocio:

Cargar itemset de las bases de datos, extraer los itemset frecuentes, una vez extraídos los itemset frecuentes se procede a extraer los itemset cerrados e itemset maximales, luego se compara la cantidad de reglas obtenidas por cada uno, mostrando un gráfico de barras que contiene la cantidad de itemsets maximales y cerrados obtenidos de cada uno de los datasets utilizados.

2.2.3 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 (Roger 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 (Grau et al. 2014), 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 (Gonzalo 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 datasets.
- Preprocesamiento de datasets.
- Extraer Itemset frecuentes.
- Extraer Itemset Maximales Frecuentes.
- Extraer Itemset Cerrados Frecuentes.
- Comparar itemsets maximales y cerrados.

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 y Greene 2005).

Los requisitos no funcionales de la plataforma son los siguientes:

- 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.

2.2.4 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 (Cardozzo 2016).

Como ejemplo se muestra la historia de usuario Cargar Datasets, las demás historias de usuario se encuentran en los anexos.

Tabla 1: Historia de usuario 1.

Historia de usuario	
Número: 1	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: Adrian Batista López	
Descripción: Es donde se cargan los datos necesarios para el funcionamiento de la aplicación. Debe cargar los datasets que se encuentran en la carpeta base de la aplicación.	
Observaciones:	

2.2.5 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 (Beck y Fowler 2001).

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.

Tabla 2Estimación de tiempo por historia de usuario

No	Historias de usuario	Estimación(en semanas)
1	Cargar datasets.	1
2	preprocesamiento de los datasets.	1
3	Extraer itemset frecuentes	3
4	Extraer itemset maximales frecuentes	2
5	Extraer itemset cerrados frecuentes	2

6	Comparar resultados	1

2.2.6 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 procede a cargar los datasets y se realiza el preprocesamiento de los dataset
- **Iteración 2:**

  En esta iteración se extraen los itemset frecuentes
- **Iteración 3:**

  En esta iteración se extraen los itemset frecuentes cerrados, los frecuentes maximales y se comparan los resultados obtenidos de acuerdo a la cantidad de itemsets maximales y cerrados obtenidos.

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

Tabla 3: Estimación de tiempo por iteración

Iteración	Tiempo estimado
Iteración 1	2
Iteración 2	3
Iteración 3	5

2.3 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 (Kicillof 2004).

Cuando el evento se anuncia, el sistema invoca todos los procedimientos que se han registrado para él. La interfaz de usuario de la aplicación se mantiene a la espera de algún evento producido por el usuario para realizar sus funcionalidades.

2.4 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 y Fowler 2001).

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 de la 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 (Casas y Reinaga 2009).

A continuación, se muestran las tarjetas CRC generadas:

Tabla 4: Tarjeta CRC de la clase mace

Clase: Mace	
Responsabilidades	**Colaboradores**
maximales cerrados	Apriori

Tabla 5: Tarjeta CRC de la clase apriori

Clase: Apriori	
Responsabilidades	**Colaboradores**
joinset subsets itemset_from_data itemset_suport freq_itemset print_report data_from_csv get_frecuent_itemset	

Tabla 6: Tarjeta CRC de la clase fcm

Clase: Fcm	
Responsabilidades	**Colaboradores**
Iniciar	Mace

2.5 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, Warsaw y Coghlan 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”.

2.6 Fase de implementación

Dentro de la metodología del desarrollo del software, 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.

2.7 Entorno de desarrollo integrado

Con el objetivo de facilitar la implementación se hace necesario un software que posea las herramientas necesarias para el desarrollo de la aplicación.
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 (Salavert y Pérez 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 2018.1.2. PyCharm es un entorno de desarrollo integrado multiplataforma utilizado para desarrollar principalmente en el lenguaje de programación Python por poseer las siguientes características (Flores, Hurtado y Zabala 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 e introducir constante.
- Depurador integrado de Python.

2.8 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). Entre los lenguajes de programación más populares para la solución de problemas de minería de datos se encuentra Java, C++ y Python. Se escoge Python como lenguaje 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.

2.9 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.

Tabla 7: Tabla de ingeniería de la funcionalidad Cargar datasets

Tarea

Número de tarea: 1	Número: 1
Nombre: Cargar datasets.	
Tipo de tarea: Desarrollo	Puntos de estimación: 1
Programador responsable: Adrian Batista López	
Descripción: Se cargan los datasets en el fichero data.csv en la carpeta del programa	

Tabla 8: Tabla de ingeniería de la funcionalidad Preprocesamiento de datasets

Tarea	
Número de tarea: 2	Número: 2
Nombre: Preprocesamiento de datasets	
Tipo de tarea: Desarrollo	Puntos de estimación: 1
Programador responsable: Adrian Batista López	
Descripción: Transforma los datasets en una estructura que sea válida para la aplicación.	

Tabla 9: Tabla de ingeniería de la funcionalidad Extraer itemsets frecuentes

Tarea	
Número de tarea: 3	Número: 3
Nombre: Extraer itemsets frecuentes	
Tipo de tarea: Desarrollo	Puntos de estimación: 2
Programador responsable: Adrian Batista López	
Descripción: Se utiliza el algoritmo apriori para extraer los itemset frecuentes.	

Tabla 10: Tabla de ingeniería de la funcionalidad Extraer itemsets maximales frecuentes

Tarea	
Número de tarea: 4	Número: 4
Nombre: Extraer itemsets maximales frecuentes.	
Tipo de tarea: Desarrollo	Puntos de estimación: 2
Programador responsable: Adrian Batista López	
Descripción: A partir de los itemsets frecuentes se extraen los itemsets maximales frecuentes.	

Tabla 11: Tabla de ingeniería de la funcionalidad Extraer itemsets cerrados frecuentes

Tarea	
Número de tarea: 5	Número: 5
Nombre: Extraer itemsets cerrados frecuentes.	
Tipo de tarea: Desarrollo	Puntos de estimación: 2
Programador responsable: Adrian Batista López	
Descripción: A partir de los itemsets frecuentes se extraen los itemsets cerrados frecuentes.	

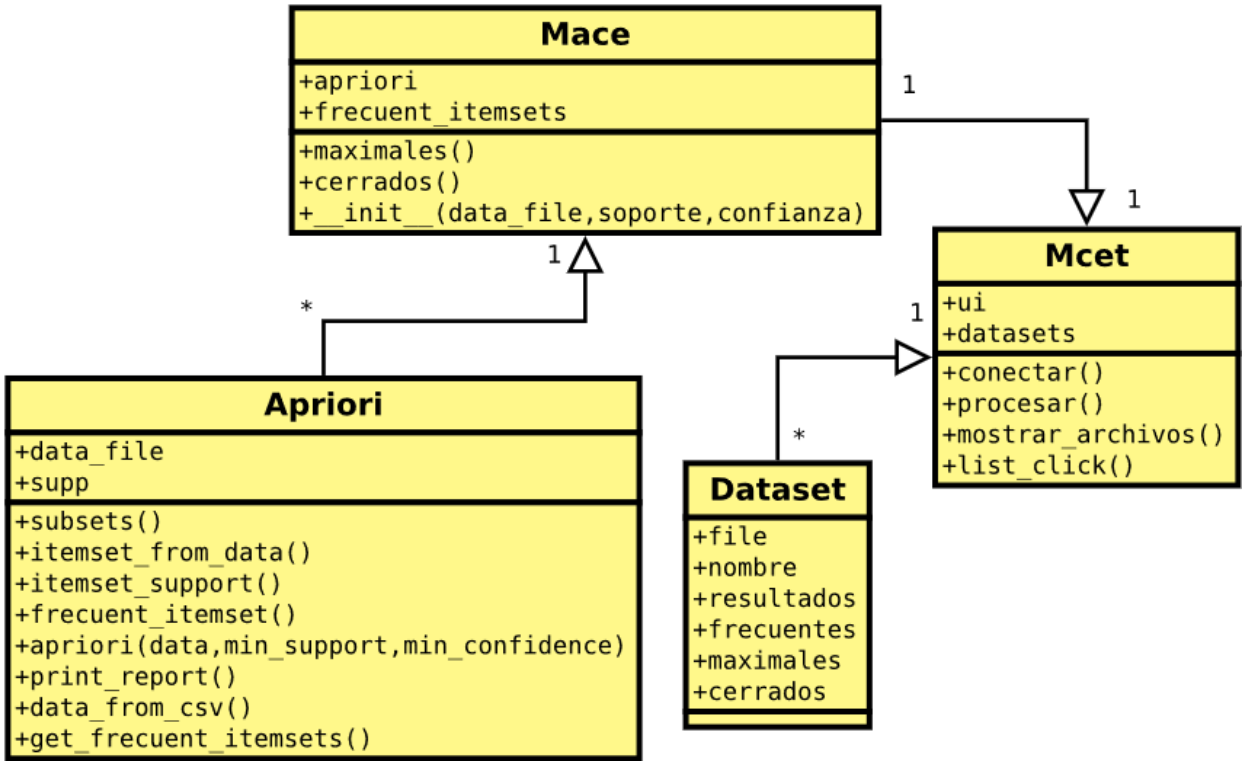
Tabla 12: Tabla de ingeniería de la funcionalidad comparar resultados

Tarea	
Número de tarea: 6	Número: 6
Nombre: Comparar resultados.	
Tipo de tarea: Desarrollo	Puntos de estimación: 1
Programador responsable: Adrian Batista López	
Descripción: A partir de los itemsets cerrados y maximales extraídos se procede a comparar los mismos.	

2.10 Diagrama de clases

A continuación, se muestra el diagrama de clases donde se describen cada una de las clases, sus atributos y sus funcionalidades, además de la interacción entre estas:

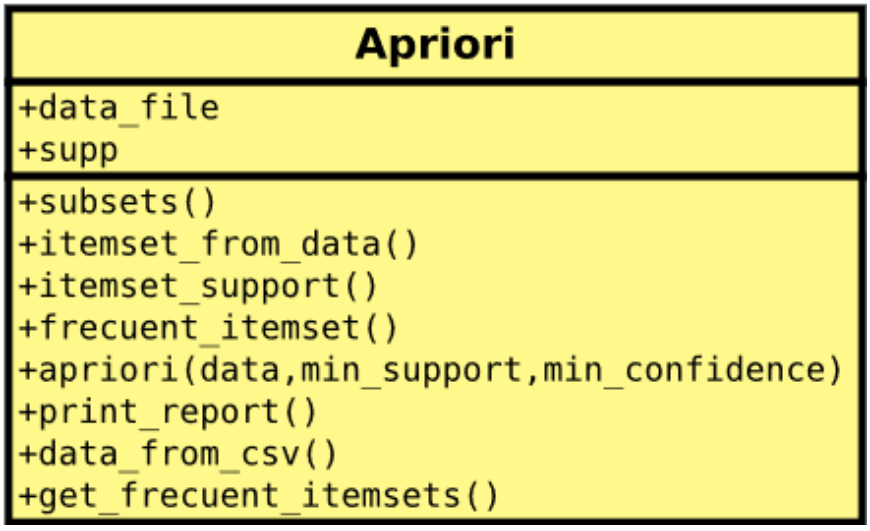
Ilustración 2: Diagrama de clases



2.10.1 Clase Apriori

La clase Apriori se encarga de extraer los itemsets frecuentes de cada dataset.

Ilustración 3: Clase Apriori



Atributos

- data_file: Archivo de entrada que contiene los datasets en formato csv, cada línea está conformada por una transacción.
- supp: Umbral de soporte mínimo para podar los itemsets que no sean frecuentes es decir los itemsets que no superen este umbral serán eliminados.

Métodos

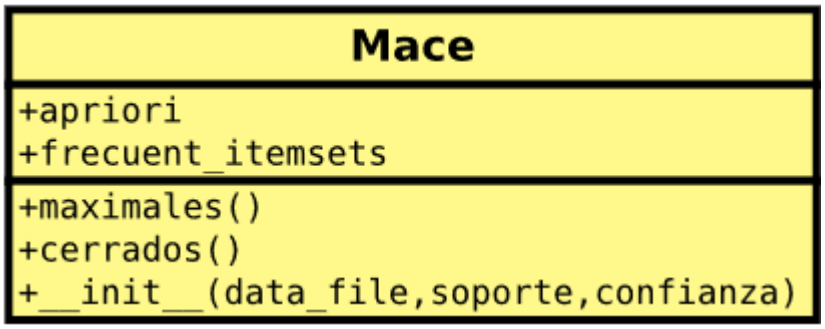
- itemset_suport: Calcula el soporte de los itemsets contenidos en el dataset.
- subsets: Obtiene todos los posibles subconjuntos de longitud n de un itemset.
- itemset_from_data: Obtiene los datasets desde data.
- frecuent_itemset: Obtiene los itemsets frecuentes.
- print_report: Muestra una representación de los resultados en la terminal.
- data_from_csv: Obtiene los datos desde el archivo csv.

- get_frecuent_itemset: Obtiene los itemsets desde el archivo de entrada y obtiene los itemsets frecuentes.

2.10.2 Clase Mace

La clase Mace es la encargada de una vez que se extraen los itemsets frecuentes, extrae a partir de estos los itemsets maximales y cerrados.

Ilustración 4: Clase Mace



Atributos

- arpiori: Instancia de la clase Apriori para calcular los itemsets frecuentes.
- frecuent_itemsets: Resultado del apriori que contiene los itemsets frecuentes.

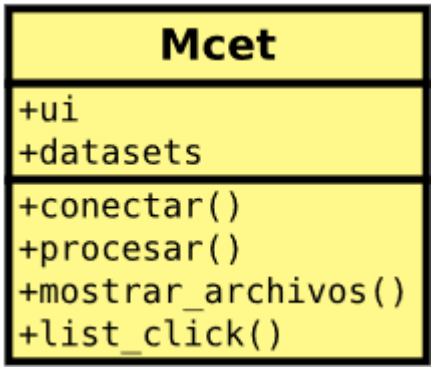
Métodos

- maximales: Obtiene los itemsets maximales frecuentes.
- cerrados: Obtiene los itemsets cerrados frecuentes.

2.10.3 Clase Mcet

La clase Mcet contiene la interfaz de usuario, esta interactúa con la clase Dataset para obtener los archivos de entrada y con la clase Mace para obtener los itemsets maximales y cerrados.

Ilustración 5: Clase Mcet



Atributos

- ui: Instancia de la clase Ui_Dialog que contiene los componentes de la interfaz de usuario generada con el Qtdesigner.
- datasets: Datasets frecuentes obtenidos con el apriori de los cuales se obtendrán los itemsets cerrados y maximales frecuentes.

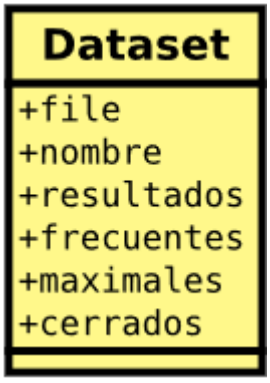
Métodos

- conectar: Conecta cada componente con sus señales y slots correspondientes.
- procesar: Calcula los itemsets frecuentes, los itemsets maximales y los itemsets cerrados.
- mostrar_archivos: Muestra un dialogo para seleccionar un archivo csv.
- list_click: Muestra los resultados según el dataset seleccionado en la lista.

2.10.4 Clase Dataset

La clase Dataset posee toda la información referida a los datasets.

Ilustración 6: Clase Dataset



Atributos

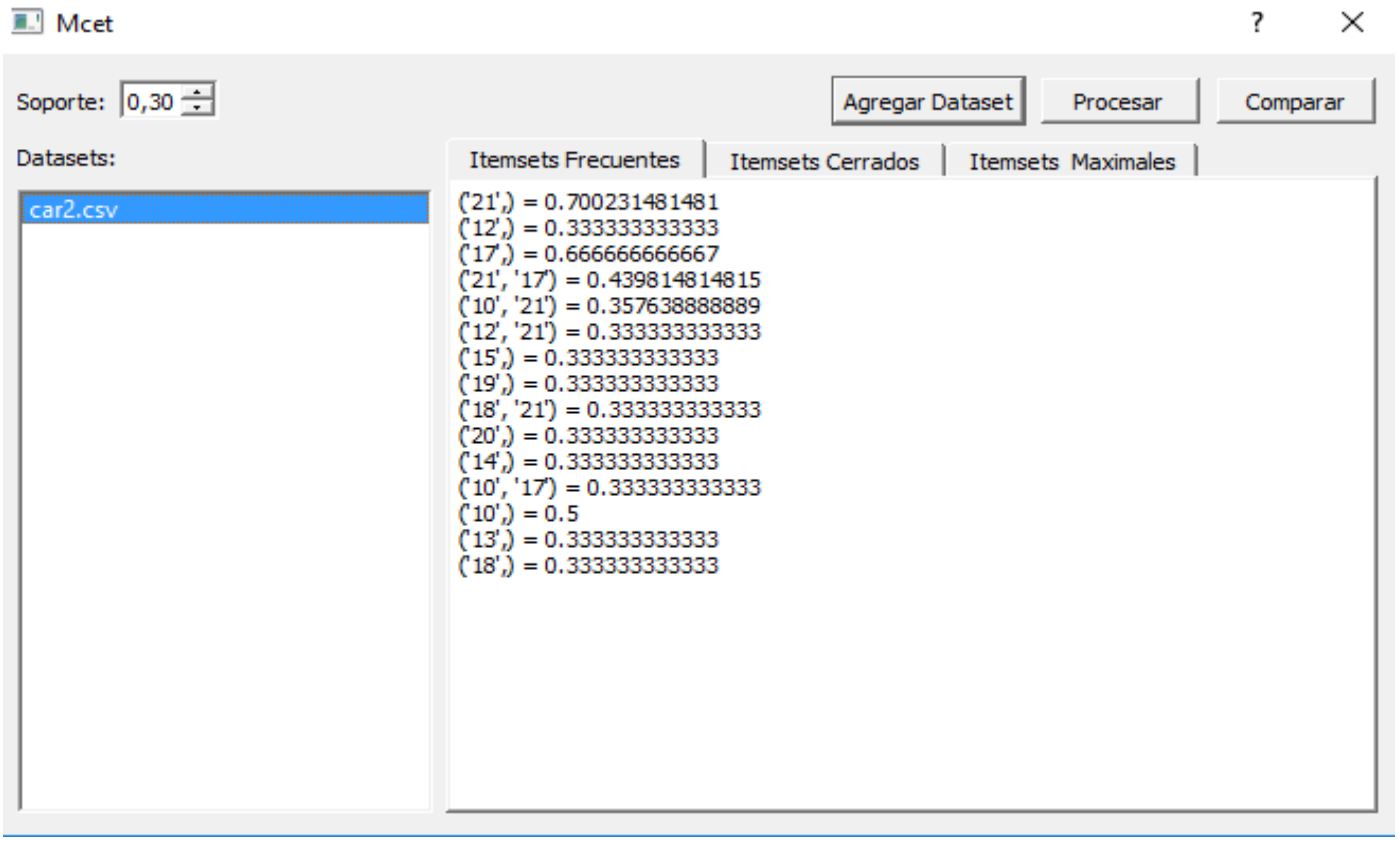
- file: Ruta del archivo csv.
- nombre: Nombre del archivo csv.
- resultados: Contiene la información del resultado obtenido.
- frecuentes: Se encuentran los itemsets frecuentes.
- maximales: Se encuentran los itemsets maximales.
- cerrados: Se encuentran los itemsets cerrados.

2.10.5 Interfaz de usuario

Con el objetivo de comparar los resultados obtenidos al extraer los itemsets maximales y los itemsets cerrados se presenta el prototipo de la interfaz principal donde puede modificarse el soporte, agregar nuevos datasets, procesar los datasets agregados, así como comparar los resultados obtenidos.

A la izquierda se muestran los dataset agregados, y a la derecha los itemsets frecuentes, cerrados y maximales resultado del procesamiento, también se pueden observar los botones que permiten agregar un nuevo dataset, procesar el dataset y comparar el mismo.

Ilustración 7: Interfaz de usuario



Conclusiones del capitulo

Se determinó que se utilizará la metodología XP ya que presenta varias ventajas cuando se trata de un equipo de desarrollo pequeño y existe una probabilidad alta a los cambios en los requisitos.

La recopilación de los requisitos tanto funcionales como no funcionales recogidos para el desarrollo de la solución permitieron definir las características principales del producto.

El desarrollo y confección de los artefactos arrojados por la metodología escogida por el equipo de trabajo, tales como historias de usuarios y tarjetas CRC permitieron documentar el proceso de desarrollo. Se obtuvo la implementación de la aplicación, dando solución a los requisitos funcionales identificados.

CAPÍTULO 3: EVALUACIÓN DE LA SOLUCIÓN PROPUESTA

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, mejorando la satisfacción del cliente y su conformidad con lo realizado. Teniendo dominio sobre todos los procesos involucrados, se aplican diferentes métodos y técnicas.

3.1.2 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 (Gutiérrez et al. 2006).

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.

3.1.3 Pruebas unitarias

Según la metodología XP el desarrollo debe ser basado en pruebas, por lo que se realizaron pruebas unitarias utilizando el módulo doctest, se muestra el ejemplo de la clase Mace:

```
def maximales(self):
    """Debuelve los itemsets maximales.

    >>> m=Mace('car2.csv',0.4,0.4)
    >>> m.maximales()
    {frozenset(['21', '17']): 0.4398148148148148, frozenset(['10']): 0.5}
    """


def cerrados(self):
    """Debuelve los itemsets cerrados.
```

```
            >>> m=Mace('car2.csv',0.4,0.4)
            >>> m.cerrados()
            {frozenset(['17']): 0.6666666666666666, frozenset(['21', '17']):
0.4398148148148148, frozenset(['21']): 0.7002314814814815, frozenset(['10']): 0.5}
            """
import doctest
doctest.testmod()
```

3.1.4 Prueba del camino Básico

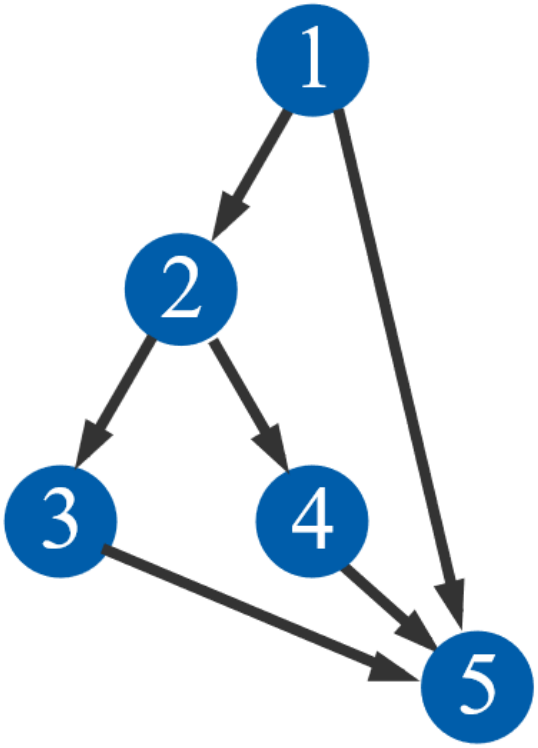
Para la aplicación de las pruebas de caja blanca se hizo uso de la técnica camino básico. El método del camino básico permite obtener una medida de la complejidad de un diseño procedimental, y utilizar esta medida como guía para la definición de una serie de caminos básicos de ejecución, diseñando casos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

Se toma como ejemplo el método mostrar_archivos de la clase Mcet.

```
def mostrar_archivos(self, evt):
  ok = QtGui.QFileDialog.getOpenFileName(self, 'Seleccione el archivo con los datasets')
  if ok:
    if isfile(ok):
      self.datasets.append(Dataset(ok))
      self.ui.listWidget_datasets.addItem(self.datasets[-1].nombre)
    else:
      QtGui.QMessageBox.information(self, 'Error', 'El archivo no existe.')
```

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

Ilustración 8: 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:

Ilustración 9: Casos de prueba

Camino	Condición de ejecución	Prueba	Salida	Resultado de la prueba
1235	Se seleccionó un archivo. Y el	ok = True isfile(ok)=True	Archivo de entrada cargado correctamente.	Prueba satisfactoria

	archivo es válido.			
1245	Se seleccionó un archivo. El archivo no es válido.	ok=True isfile(ok)=False	Se muestra un mensaje. El archivo no existe.	Prueba satisfactoria
15	No se carga ningún archivo.	ok=False	No se carga el archivo.	Prueba satisfactoria

3.2.1 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 (Gutiérrez et al. 2006).

Estas pruebas tienen como propósito detectar (Roger 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.

3.2.2 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 (Canós y Letelier 2012).

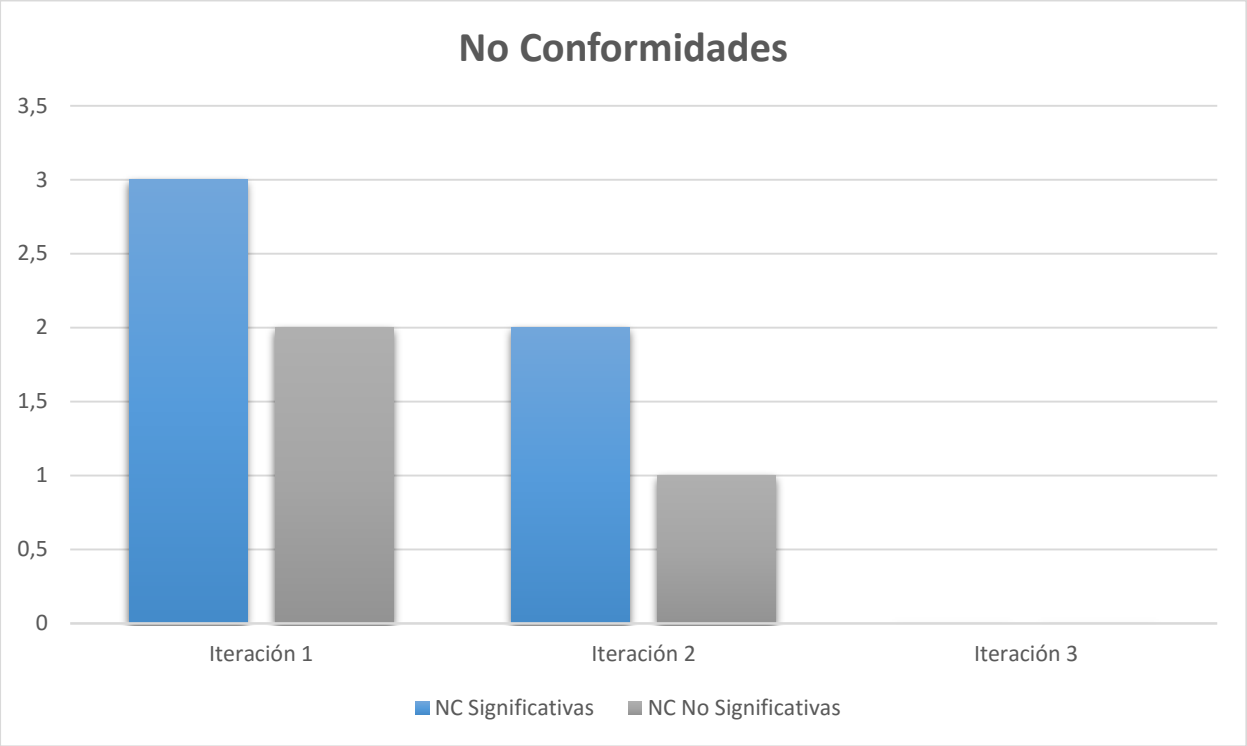
Tabla 13: Descripción del caso de prueba para la historia de usuario: Cargar dataset.

Escenario	Descripción	Observaciones	Respuesta del sistema	Flujo central
1.Cargar dataset.	Es donde se cargan los datos necesarios para el funcionamiento de la aplicación. Debe cargar los datasets que se encuentran en la carpeta base de la aplicación.	Se debe verificar que el archivo de los datasets no esté corrupto.	Se cargan los datasets correspondientes.	Al iniciar el programa, se seleccionan los datasets que se quieren cargar.

Para la aplicación se realizaron 5 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 10: No Conformidades.



3.2.3 Casos de estudios

Se decide realizar a la aplicación casos de estudio con el objetivo de validar las funcionalidades de la misma. El primer caso de estudio consta de un experimento donde se corre el algoritmo apriori y 3 datasets obtenidos del repositorio de datasets de la Universidad de California en Irvine (Dheeru, Dua and Karra Taniskidou, Efi 2017) .

En la siguiente tabla se muestra información sobre los da tasets:

Iris: El conjunto de datos contiene 3 clases de 50 instancias cada una, donde cada clase se refiere a un tipo de planta de iris.

Car y car 2: La base de datos de evaluación de automóviles se derivó de un modelo de decisión jerárquico simple.

Ilustración 11: Datasets del experimento

Nombre	Descripción	Número de instancias	Atributos

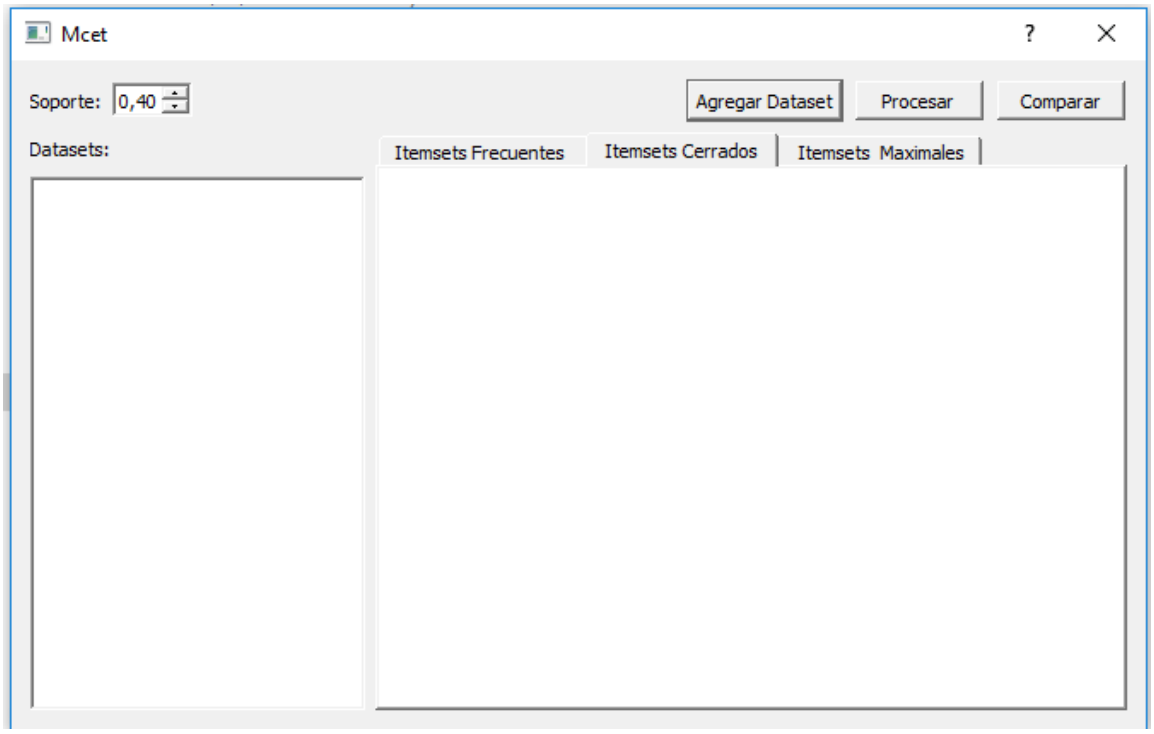
iris	Clasificación de la planta Iris.	150	6
car2	Clasificación de automóviles.	1580	5
car	Clasificación de automóviles.	1728	7

A continuación, se muestra el proceso realizado:

1. Seleccionar Datasets.

Debe seleccionar al menos un dataset.

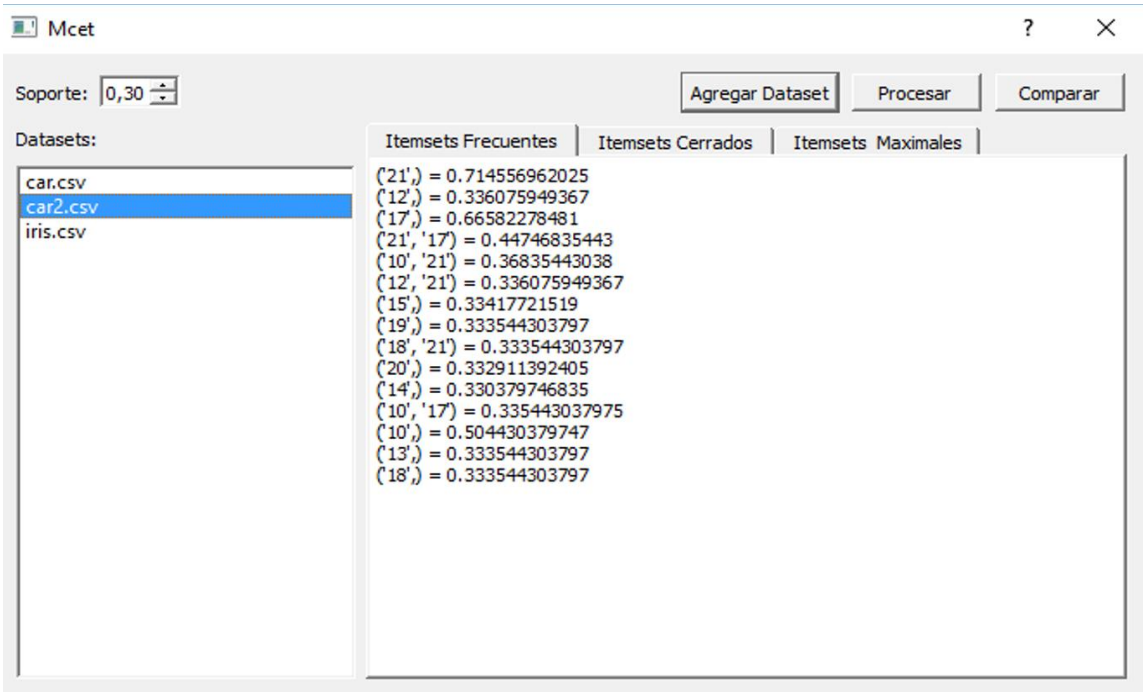
Ilustración 12: Interfaz principal



2. Procesar datasets.

Se procesan los datasets seleccionados, seleccionando la opción procesar para proceder a extraer los itemsets frecuentes, maximales y cerrados de cada dataset.

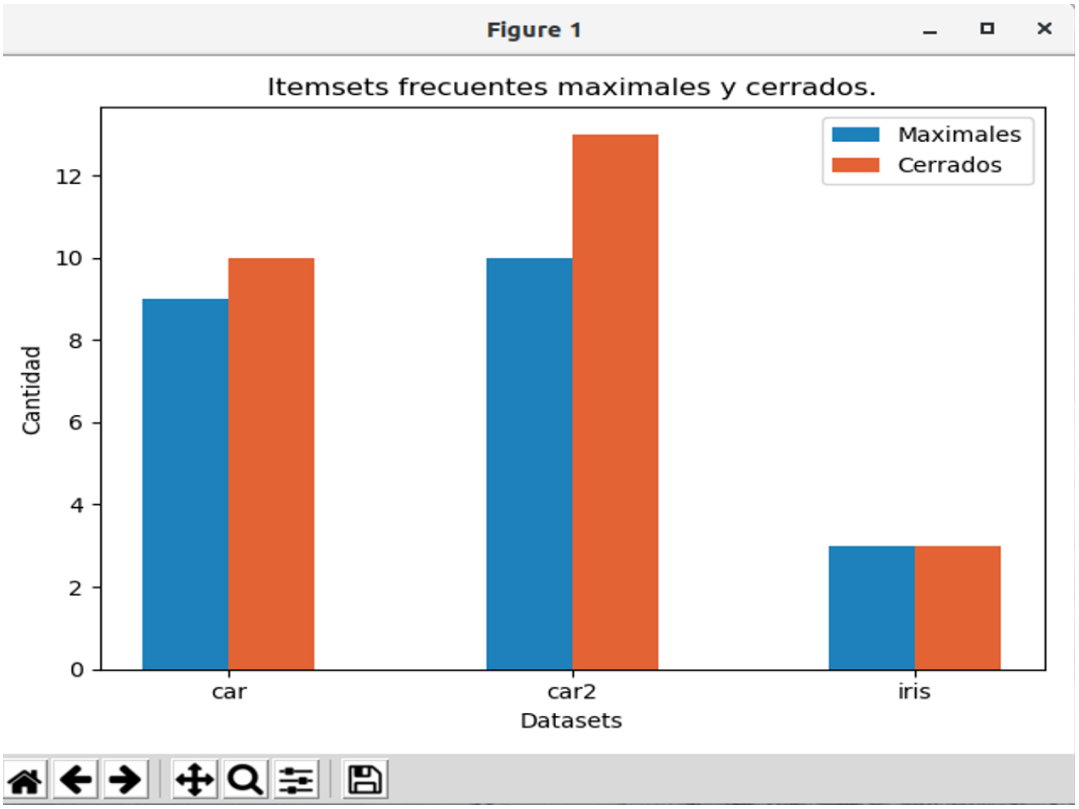
Ilustración 13: Interfaz principal con datases cargados y procesados



3. Comparar datasets.

Se selecciona la opción comparar y se muestra un gráfico de barras comparando la cantidad de itemsets maximales y cerrados obtenidos a partir del procesado de los datasets.

Ilustración 14: Comparación de los resultados.



3.2.4 Casos de estudio

A continuación, se muestra un ejemplo de los resultados obtenidos a partir de obtener los itemsets cerrados y maximales de forma manual y se comparan estos resultados con la aplicación desarrollada. Utilizando el siguiente dataset ficticio.

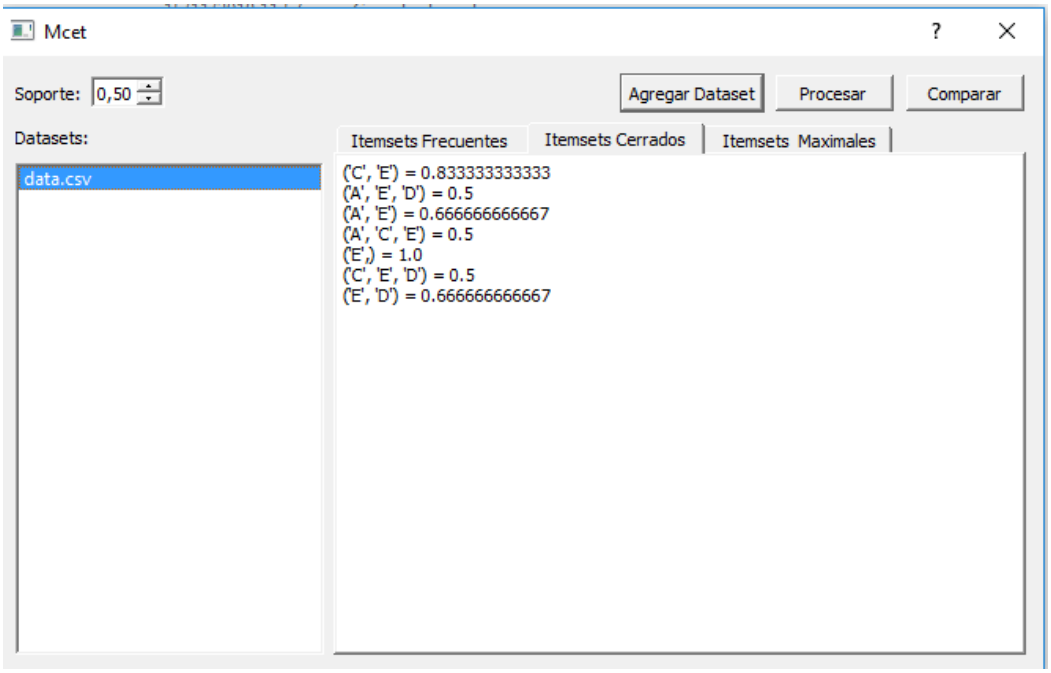
A,*B*,*C*,*E*
A,*C*,*D*,*E*
B,*C*,*E*
A,*C*,*D*,*E*
C,*D*,*E*
A,*D*,*E*

Itemsets cerrados obtenidos manualmente:
{E} = 6
{A, E} = 4
{C, E} = 5
{D, E} = 4

{A, C, E} = 3
{A, D, E} = 3
{C, D, E} = 3

Itemsets cerrados obtenidos por la aplicación

Ilustración 15: Itemsets cerrados obtenidos.

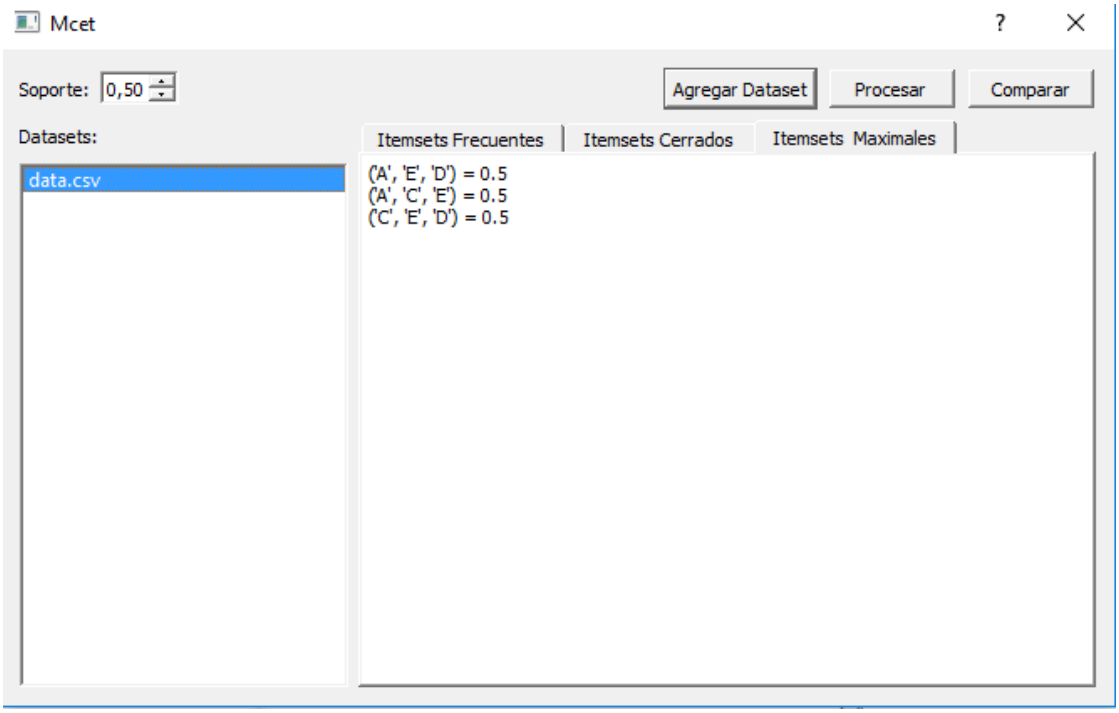


Itemsets maximales obtenidos manualmente:

{A, C, E} = 3
{A, D, E} = 3
{C, D, E} = 3

Itemsets maximales obtenidos por la aplicación:

Ilustración 16: Itemsets maximales.



Conclusiones del capítulo

Luego de realizada 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, se demostró 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. Después de realizada las pruebas de aceptación el cliente evidenció su satisfacción con el producto desarrollado.

CONCLUSIONES

Como resultado de la presente investigación se obtuvo una aplicación que permite comparar el tamaño de los modelos de reglas de asociación generados utilizando itemsets maximales e itemsets cerrados, teniendo en cuenta la misma entrada. 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 obtuvo la implementación de la aplicación, dando solución a los requisitos funcionales identificados. Se demostró 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. Después de realizada las pruebas de aceptación el cliente evidenció su satisfacción con el producto desarrollado.

ANEXOS

Historias de usuario

Tabla 14 Historia de Usuario 2

Historia de usuario	
Número: 2	Nombre: Preprocesamiento de 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: Adrian Batista López	
Descripción: Transforma los datasets en una estructura que sea válida para la aplicación.	
Observaciones:	

Tabla 15: Historia de Usuario 3

Historia de usuario	
Número: 3	Nombre: Extraer Itemset frecuentes
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: Adrian Batista López	
Descripción: Es donde se utiliza el algoritmo Apriori para extraer los itemsets frecuentes del dataset original.	
Observaciones:	

Tabla 16: Historia de Usuario 4

Historia de usuario	
Número: 4	Nombre: Extraer Itemset maximales frecuentes
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: Adrian Batista López	
Descripción: Despues de extraer los itemset frecuentes, de estos se extraen los itemset maximales frecuentes.	
Observaciones:	

Tabla 17: Historia de Usuario 5

Historia de usuario	
Número: 5	Nombre: Extraer Itemset cerrados frecuentes
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: Adrian Batista López	
Descripción: Despues de extraer los itemset frecuentes, de estos se extraen los itemset cerrados frecuentes.	
Observaciones:	

Tabla 18: Historia de Usuario 6

Historia de usuario

Número: 6	Nombre: Comparar resultados.
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: Adrian Batista López	
Descripción: Es donde se comparan los itemsets maximales y cerrados de acuerdo a cuantos fueron extraídos por cada dataset.	
Observaciones:	

BIBLIOGRAFÍA

AGRAWAL, R., IMIELIŃSKI, T. y SWAMI, A., 1993. Mining association rules between sets of items in large databases. *Acm sigmod record*. S.l.: ACM, pp. 207-216. ISBN 0-89791-592-5.

ANTONY, P.J., MANUJESH, P. y JNANESH, N.A., 2016. Data mining and machine learning approaches on engineering materials—a review. *Recent Trends in Electronics, Information & Communication Technology (RTEICT), IEEE International Conference on*. S.l.: IEEE, pp. 69-73. ISBN 1-5090-0774-1.

BECK, K. y FOWLER, M., 2001. *Planning extreme programming*. S.l.: Addison-Wesley Professional. ISBN 0-201-71091-9.

BECK, K. y GAMMA, E., 2000. *Extreme programming explained: embrace change*. S.l.: addison-wesley professional. ISBN 0-201-61641-6.

BUSTAMANTE, D. y RODRÍGUEZ, J., 2014. Metodología Actual Metodologia XP. *Informe. Barinas: Universidad Nacional Experimental de los Llanos occidentales Ezequiel Zamora, Facultad de informática*,

CANÓS, J.H. y LETELIER, M.C.P.P., 2012. Metodologías ágiles en el desarrollo de software. ,

CANTOR, G. y VENN, J., 1998. Teoría de conjuntos. ,

CARDOZZO, D.R., 2016. *Desarrollo de Software: Requisitos, Estimaciones y Análisis*. S.l.: IT Campus Academy. ISBN 1-5300-8861-5.

CARRION, J., 2017. Diferencia entre dato información y conocimiento. ,

CASAS, S.I. y REINAGA, H.H., 2009. Aspectos Tempranos: un enfoque basado en Tarjetas CRC. *Avances en Sistemas e Informática*, vol. 6, no. 1, pp. 85-92.

DHEERU, DUA AND KARRA TANISKIDOU, EFI, 2017. *UCI Machine Learning Repository* [en línea]. 2017. S.l.: University of California, Irvine, School of Information and Computer Sciences. Disponible en: http://archive.ics.uci.edu/ml.

DIAZ VERA, J., MOLINA FERNÁNDEZ, C. y VILA MIRANDA, M.-A., 2016. Reducción de Redundancia en Reglas de Asociación. *Revista Cubana de Ciencias Informáticas*, vol. 10, no. 1, pp. 55-70.

FÉLIX, M. y CARLOS, L., 2002. Data mining: torturando a los datos hasta que confiesen. *Coordinador del programa de Data mining*,

FERNANDO, B., 2006. *Reglas de asociación*. 2006. S.l.: DECSAI.

FLORES, Á., HURTADO, J. y ZABALA, B., 2018. Sistema inteligente web. *Caribeña de Ciencias Sociales*, no. mayo.

FUENTES HERRERA, I.E., 2016. *Descubrimiento de conocimiento en entornos hospitalarios a partir de registros médicos para la toma de decisiones*. S.l.: Universidad Central “Marta Abreu” de Las Villas.

GONZALO, M., 2008. *Ingeniería de requisitos*. 2008. S.l.: Universidad Complutense de Madrid.

GRANDINETTI, W.M., 2005. Tesis de Magister en Ciencias de la Computación. ,

GRAU, R., LAUENROTH, K., BEREZA, B., VAN VEENENDAAL, E. y VAN DER ZEE, S., 2014. Requirements engineering and agile development-collaborative, just enough, just in time, sustainable. *International Requirements Engineering Board*,

GUTIÉRREZ, J.J., ESCALONA, M.J., MEJÍAS, M. y TORRES, J., 2006. Pruebas del Sistema en Programación Extrema. *Departamento de Lenguajes y Sistemas Informáticos. Universidad de Sevilla*,

HAN, J., PEI, J. y KAMBER, M., 2011. *Data mining: concepts and techniques*. S.l.: Elsevier. ISBN 0-12-381480-4.

HERNÁNDEZ ORALLO, J., FERRI RAMIREZ, C. y RAMIREZ QUINTANA, M.J., 2004. *Introducción a la Minería de Datos*. S.l.: Pearson Prentice Hall,. ISBN 84-205-4091-9.

JOYANES AGUILAR, L., 2003. *Fundamentos de programación: algoritmos y estructura de datos y objetos*. S.l.: s.n. ISBN 84-481-3664-0.

KICILLOF, C.R.-N., 2004. Lenguajes de Descripción de Arquitectura (ADL). ,

KOSA, M., YILMAZ, M., O'CONNOR, R. y CLARKE, P., 2016. Software engineering education and games: a systematic literature review. *Journal of Universal Computer Science*, vol. 22, no. 12, pp. 1558-1574.

LAROSE, D.T., 2005. An introduction to data mining. *Traduction et adaptation de Thierry Vallaud*,

MATATOV, N., ROKACH, L. y MAIMON, O., 2010. Privacy-preserving data mining: A feature set partitioning approach. *Information Sciences*, vol. 180, no. 14, pp. 2696-2720.

MEDINA PAGOLA, J.E., 2007. *Generación de conjunto de ítems y reglas de asociación*. S.l.: s.n.

PEI, J., HAN, J. y MAO, R., 2000. Closet: An efficient algorithm for mining frequent closed itemsets. *ACM SIGMOD workshop on research issues in data mining and knowledge discovery*. S.l.: s.n., pp. 21-30.

PRESSMAN, R.S., 2005. *Software engineering: a practitioner's approach*. S.l.: Palgrave Macmillan. ISBN 0-07-301933-X.

PRESSMAN, R.S. y TROYA, J.M., 1988. Ingeniería del software. ,

PRIETO, A., LLORIS, A. y TORRES, J.C., 1995. *Introducción a la Informática*. S.l.: McGraw-Hill,.

ROGER, S., 2010. Pressman, Ingeniería del software un enfoque práctico séptima edición. *México DF: McGraw Hill*, pp. 1-74.

SALAVERT, I.R. y PÉREZ, M.D.L., 2000. *Ingeniería del software y bases de datos: tendencias actuales*. S.l.: Univ de Castilla La Mancha.

SAMUEL OJEDA PEREIRA, 2017. *Extracción de correlaciones entre el test vocacional CHASIDE y la carrera de ingenieria en ciencias informáticas.* S.l.: s.n.

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

SIEGMUND, J., SIEGMUND, N. y APEL, S., 2016. How reviewers think about internal and external validity in empirical software engineering. *Software Engineering 2016*,

ŠMITE, D., WOHLIN, C., GALVIŅA, Z. y PRIKLADNICKI, R., 2014. An empirically based terminology and taxonomy for global software engineering. *Empirical Software Engineering*, vol. 19, no. 1, pp. 105-153.

SOMMERVILLE, I., 2005. Integrated requirements engineering: A tutorial. *IEEE software*, vol. 22, no. 1, pp. 16-23.

STELLMAN, A. y GREENE, J., 2005. *Applied software project management*. S.l.: O'Reilly Media, Inc. ISBN 0-596-55382-X.

VAN ROSSUM, G., WARSAW, B. y COGHLAN, N., 2001. PEP 8: style guide for Python code. *Python. org*,

ZAKI, M.J. y HSIAO, C.-J., 2002. CHARM: An efficient algorithm for closed itemset mining. *Proceedings of the 2002 SIAM international conference on data mining*. S.l.: SIAM, pp. 457-473.

ZOU, Q., CHU, W.W. y LU, B., 2002. SmartMiner: A depth first algorithm guided by tail information for mining maximal frequent itemsets. *Data Mining, 2002. ICDM 2003. Proceedings. 2002 IEEE International Conference on*. S.l.: IEEE, pp. 570-577. ISBN 0-7695-1754-4.