Universidad Nacional Autónoma de México

Facultad de Ciencias

Algoritmo basado en el Apriori para la Generación de Reglas de Asociación en Documentos XML

T E S I S

QUE PARA OBTENER EL TÍTULO DE:

Licenciado en Ciencias de la Computación

PRESENTA:

Edward Israel Avendaño Rodriguez

DIRECTORA DE TESIS:
Dra. Amparo López Gaona

México, D.F., 2016

Hoja de datos del jurado

1. Datos del alumno
Avendaño
Rodriguez
Edward Israel
56 77 05 57
Universidad Nacional Autónoma de México
Facultad de Ciencias
Ciencias de la Computación
308012433

2. Datos del tutor
Dra.
Amparo
López
Gaona

3. Datos del sinodal 1
M. en I.
Gerardo
Avilés
Rosas

4. Datos del sinodal 2
Dr.
Gustavo
De La Cruz
Martínez

5. Datos del sinodal 3
Dr.
José de Jesús
Galaviz
Casas

6. Datos del sinodal 4
Dra.
María de Luz
Gasca
Soto

7. Datos del trabajo escrito
Algoritmo basado en el Apriori para la Generación de Reglas de Asociación en Documentos XML
88 p
2016

A mi familia y amigos.

Agradecimientos

Quiero agradecer a mis padres: Ignacio y Rogelia, por todo el apoyo que me han brindado durante todos estos años, todas las enseñanzas y la confianza. No lo habría logrado sin ustedes.

A mis hermanos Alejandro y Nacho (†) , con ustedes crecí y nos divertimos juntos, gracias por la compañía y momentos que han estado a mi lado.

A la UNAM por formarme desde el bachillerato, en ella conocí grandes personas y amigos, además de haber aprendido de los mejores profesores de México.

A mi tutora la Dra. Amparo López Gaona, por haberme dado la confianza de poder realizar este trabajo a su lado. Así como su apoyo y consejos durante la realización del mismo, sin los cuales no podría haberlo concluido.

A cada uno de mis sinodales que me apoyaron revisando este trabajo, gracias por sus comentarios y consejos que me permitieron lograr una versión más completa del mismo.

A todos y cada uno de mis amigos, gracias a todos ustedes por haber estado a mi lado, por todos los consejos y experiencias que hemos vivido durante este tiempo. Sin su apoyo este trabajo no habria sido posible.

Índice general

Índice de figuras

Lista de Algoritmos

Lista de Ejemplos

Introducción

Motivación

La minería de datos es el proceso que consiste en analizar datos desde distintas perspectivas y resumirlos en información útil, como lo son las correlaciones o patrones entre los campos de grandes bases de datos.

Uno de los ejemplos más conocido de la minería de datos es el siguiente:

> Una cadena de supermercado en los Estados Unidos de Norteamerica realizó un estudio tratando de contestar la siguiente pregunta: ¿Qué producto se vende más en compañía de los pañales? Esta cadena usó la minería de datos para encontrar una serie de patrones, y en vez de encontrar un producto relacionado con los bebés como sería leche o talco, descubrieron que el producto más vendido junto a estos fue la cerveza. Además descubrieron que estos eran comprados generalmente los días viernes por la tarde, por hombres entre 25 y 35 años. Esta cadena entonces decidió acercar ambos productos, logrando que se incrementara la venta de cerveza por parte de los que ya la compraban, y de los que no la compraban que empezarán a hacerlo.

Actualmente, la minería de datos es usada mayormente en bases de datos relacionales. Sin embargo, este tipo de bases no son la única manera de almacenar información. También se puede almacenar información en archivos externos, como en este caso serían los documentos XML.

El lenguaje XML se ha convertido rápidamente en un estándar para el intercambio y representación de los datos. A medida que sucedió esto, los datos en este formato se volvieron más comunes, por lo tanto, la posibilidad de obtener conocimiento de estas fuentes de documentos XML disminuyó por la heterogeneidad de los datos y sus estructuras irregulares. Por lo tanto, para obtener conocimiento se requieren de algunas técnicas más avanzadas de procesamiento para analizar una gran cantidad de datos semiestructurados, lo cual, nos permitiria explotar de una manera más general las respuestas de los servicios web y metadatos.

A diferencia del modelo relacional, el lenguaje XML es extensible y semiestructurado. Es decir que sigue una estructura implícita, en la que pueden faltar elementos, o bien, sus componentes pueden cambiar de tipo, además los componentes pueden ser débilmente tipados. Otra ventaja es el poder agregar nuevas etiquetas sin perder compatibilidad.

Esta ventaja ha hecho de XML, un estándar para el intercambio de información en Internet, así como también ha servido como una manera alterna al almacenamiento de la información, que difiere del modelo tradicional, el relacional, por su facilidad de extenderse y poder omitir ciertos datos cuando no se requieran.

Teniendo una gran cantidad de documentos XML es posible usar la minería de datos para encontrar patrones entre ellos. Sin embargo, para poder aplicarla se realiza una transformación de los documentos XML dependiendo de la tarea que se busque realizar. Por ejemplo, en el caso de reglas de asociación se transforma a un documento XML basado en transacciones, con lo cual se pierde su contenido estructural pero permite aplicar algoritmos de minería de datos que buscan reglas de asociación.

La mayor motivación es poder aplicar la minería de datos directamente en XML, sin tener que convertirlo antes al modelo relacional o en un documento donde se pierda la estructura original.

Objetivo

Este trabajo tiene por objetivo aplicar los conceptos de minería de datos, particularmente las reglas de asociación usando el algoritmo Apriori en documentos XML, para obtener relaciones desconocidas entre los datos. Lo novedoso de este

trabajo es que se desea conservar la estructura original de los datos, con lo cual se conserva la informacion semántica de los mismos.

Estructura de la tesis

Este trabajo se divide en dos grandes bloques. En el primer bloque, que abarca los primeros dos capítulos, se muestran conceptos de minería de datos, documentos XML, algoritmos de minería de datos en documentos XML y reglas de asociación.

En el Capítulo 1 se da un panorama sobre la minería de datos, en éste se explica su historia y los distintos tipos de minería. También se presenta una introducción a los documentos XML, cómo se pueden clasificar, los tipos de esquemas e interfaces de manipulación. Además se ve una introducción a la minería de datos en documentos XML.

En el Capítulo 2 se presentan los principales algoritmos para la extracción de reglas de asociación (Apriori y FP-Growth), además se retoman los conceptos de regla de asociación e itemsets; posteriormente se analizan propuestas para la generación de reglas a partir de documentos XML.

Mientras que en el segundo bloque, que abarca el Capítulo 3, se propone un algoritmo de preprocesamiento y uno de posprocesamiento, los cuales, se aplicarán sobre los documentos XML para que sea posible aplicar sobre ellos el algoritmo Apriori. Además se realiza un análisis de los problemas que se presentan para la aplicación de este algoritmo.

También se muestra un caso práctico en donde se aplican estos algoritmos, donde se explican las diversas transformaciones por las que va pasando el documento hasta llegar a su última etapa, la cuál es la salida en forma de reglas de asociación. Asimismo se realiza un análisis de los resultados obtenidos al ejecutar estos algoritmos con diversas entradas, como pueden ser un número mayor de transacciones o un número mayor de elementos dentro de las transacciones.

Finalmente, se exponen las conclusiones obtenidas, así como un panorama de la situación de los formatos de intercambio de información en internet, y que trabajos futuros se podrán realizar a partir de estos.

Capítulo 1

Panorama general de la minería de datos y XML

1.1. Introducción

En los últimos años, se ha visto el apogeo de Internet y los medios digitales, ya sean sitios web, programas, o aplicaciones móviles, y como consecuencia de lo anterior, se tiene un incremento en la generación de datos. Esto ha sido tanto benéfico, como problemático, ya que todos estos datos podrían ser analizados para obtener información valiosa a partir de ellos. Sin embargo, si los analizara una persona y quisiera obtener una conclusión a partir de ellos, sólo podría hacerlo a partir de una pequeña parte de esa inmensidad de datos, ya que sin la ayuda de una computadora es casi imposible poder realizar el análisis de todos los datos.

Aunque lo anterior es un tema reciente, el deseo de analizar los datos y poder obtener información a partir de ellos es un problema que ya lleva bastante tiempo. Llegando incluso a mencionarse como el primer antecedente una investigación en Londres en 1662, la cual fue realizada por John Graunt. Él, basándose en datos que se habían recopilado sobre la peste bubónica a lo largo de los años, trató de realizar un modelo que pudiera predecir el siguiente brote en la ciudad,[9].

Sin embargo, el antecedente que abrió este panorama es el del Taller sobre Descubrimiento de Conocimiento en Bases de Datos[1], en particular la frase "Descubrimiento de Conocimiento en Bases de Datos" (Knowledge Discovery in

Databases o KDD en ingles) es acuñada en este taller,[34]. Esto debido a que había cierta incertidumbre sobre los conceptos de máquinas de aprendizaje aplicados en las bases de datos, ya que se llegaba a pensar que lo unico que se realizaba era el paso de mineria de datos, por lo que se buscó aclarar que el objetivo sería obtener conocimiento útil a partir de los datos y los pasos que involucra. De esta forma el KDD resulta ser un proceso completo, y no sólo la etapa de minería de datos, [27].

La primera interrogante era, en qué se diferenciaba KDD del reconocimiento de patrones, a lo cual la respuesta es que del reconocimiento de patrones se tomaron ciertos métodos que se usan en la etapa de minería de datos, por ejemplo, agrupación, regresiones o reglas de asociación. Dado que el KDD está enfocado al proceso global del descubrimiento de conocimiento en datos, involucra aspectos de almacenamiento y acceso a la información, la escalabilidad de los algoritmos ante grandes cantidades de datos y la interpretación de los resultados.

Los pasos del Descubrimiento de Conocimiento son los mostrados en la Figura 1.1 y explicados a continuación:

1. Limpieza de Datos. Se realiza una depuración de los datos dado que estos pueden estar incompletos o ser inconsistentes. Se busca llenar los valores perdidos, identificar valores atípicos y corregir las inconsistencias en los datos.

2. Integración de Datos. La minería de datos requiere que se unan datos de fuentes heterogéneas. Por lo que la integración de los datos ayuda a reducir y evitar redundancia e inconsistencias dentro del conjunto de datos resultante.

3. Selección de Datos. Se busca obtener una representación reducida del conjunto de datos, el cual es menor en volumen, pero mantiene la integridad de los datos originales.

4. Transformación de Datos. Los datos son transformados o consolidados, de manera que el proceso de minado sea más eficiente, y los patrones encontrados sean más fáciles de entender.

5. Minería de Datos. Un proceso donde métodos de diversas áreas son aplicados para extraer patrones de datos.

6. Evaluación de Patrones. Se identifican los patrones relevantes que representan conocimiento basados en diversas métricas.

7. Presentación de Conocimiento. Se utiliza la visualización y técnicas de representación de conocimiento para presentar el conocimiento minado al usuario final.

Se puede observar en la Figura 1.1 que es importante hacer una buena preparación de los datos para poder aplicar alguna técnica de minería de datos.

La minería de datos se refiere a extraer o "minar" conocimiento de grandes cantidades de datos. Para esto se aplican métodos específicos que extraen informacion útil de los datos. Dependiendo de las necesidades se pueden aplicar técnicas como Clasificación, Agrupación, Predicción, Reglas de Asociación o Redes Neuronales,[21, 32]. La mayoría de los métodos que se usan están basados en técnicas del máquinas de aprendizaje, reconocimiento de patrones y estadística.

Los resultados que arroja la minería de datos son patrones, estos tienen que ser válidos, novedosos, útiles y comprensibles, [40]. Pero hay que recordar que lo que se está buscando es el conocimiento.

Hay una progresión desde los datos sin procesar hasta el conocimiento. El conocimiento es clasificado como inductivo y deductivo. El conocimiento deductivo, como su nombre lo dice, deduce nueva información basada en aplicar ciertas reglas lógicas de deducción a los datos.

La minería de datos, sin embargo, se enfoca más al conocimiento inductivo, el cual descubre nuevas reglas y patrones de los datos proporcionados. El conocimiento puede ser representado de distintas maneras. En un sentido no estructurado puede ser visto como reglas o proposiciones lógicas. En una forma estructurada, se puede representar como árboles de decisión, redes semánticas, redes neuronales o jerarquías de clases.

Existen metodologías alternativas al KDD, como lo son SEMMA (**S**ample, **E**xplore, **M**odify, **M**odel, **A**ssess por sus iniciales) o CRISP-DM (**Cr**oss-**I**ndustry **S**tandard **P**rocess for **D**ata **M**ining).

SEMMA fue desarrollada por SAS Institute, y esta enfocada a su software (Enterprise Miner), y consiste de cinco etapas:

1. Muestreo – Esta etapa consiste en tomar una muestra de los datos, por medio de la extracción de una porcion de una gran conjunto de datos, lo suficientemente grande de modo que sea significativa, y suficientemente pequeña para manipularla de manera rápida.

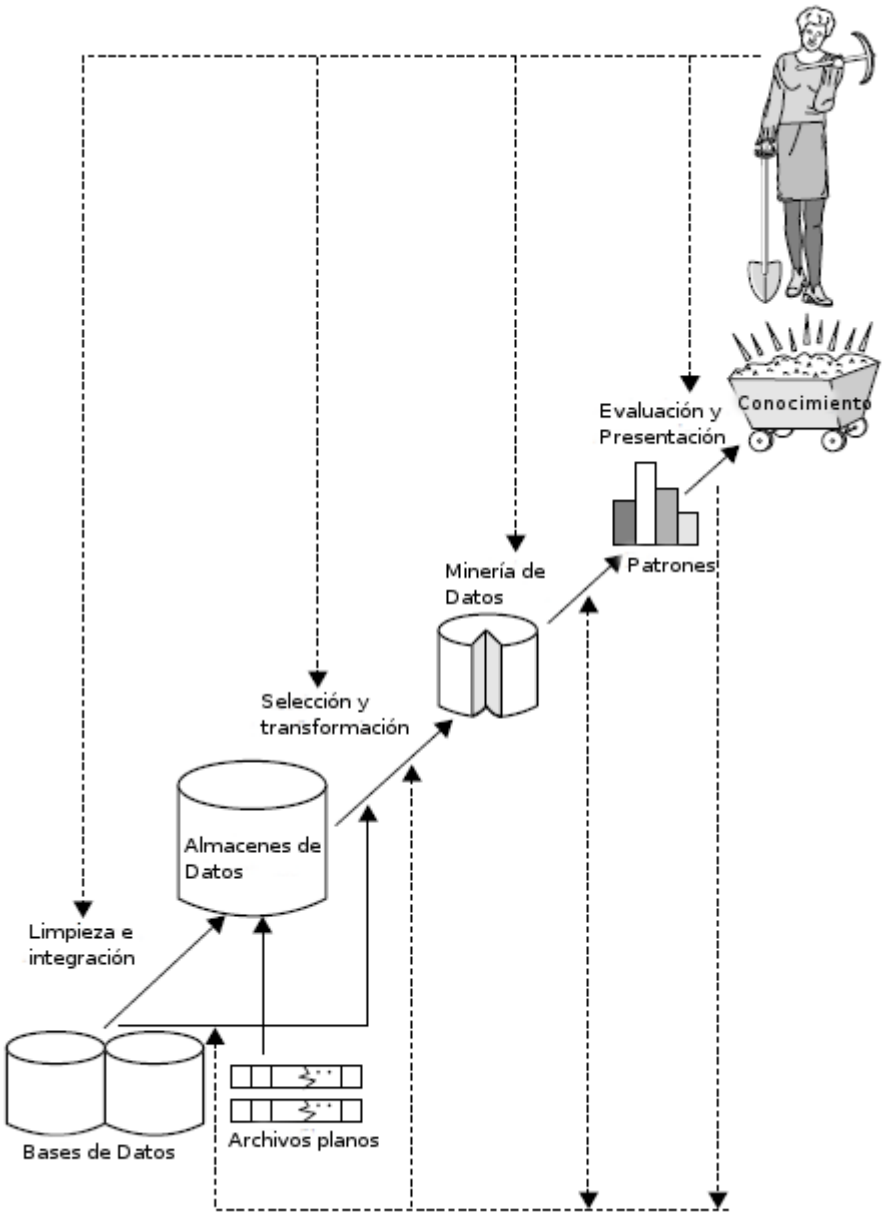

Figura 1.1: Proceso del Descubrimiento de Conocimiento [27]

2. Explorar – Esta etapa consiste en explorar los datos en busqueda de tendencias inesperadas y anomalias para tener mayor conocimiento de los mismos.

3. Modificación – Esta etapa consiste en modificar los datos por medios de la creación, selección y transformación de las variables para enfocar el proceso de selección de modelos.

4. Modelado – Esta etapa consiste en modelar los datos permitiendo que un software busque de manera automatica una combinación de los datos que puedan predecir de manera confiable la salida deseada.

5. Valoración – Esta etapa consiste en valorar los datos evaluando la utilidad y confiabilidad de los resultados del proceso de mineria de datos y estimar qué tan bien funciona.

Esta metodología esta pensada para ser independiente de la herramienta de minería de datos que se use, pero como se menciono antes esta ligada a su software. Las etapas de SEMMA son muy similares a las del KDD, por lo que se puede ver como una implementación práctica del mismo.

CRISP-DM fue desarrollada por DaimlerChryrler, SPSS y NCR, está consiste de seis etapas:

1. Comprensión del Negocio – Esta etapa inicial se enfoca en entender los objeticos del proyecto y requerimientos desde un punto de vista de negocios. Estó para convertir este conocimiento en una definición de problema de minería de datos y un plan preliminar diseñado para lograr los objetivos.

2. Comprensión de los datos – La etapa de entendimiento de los datos empieza con una colección inicial de datos y continua con actividades que tienen el objetivo de familiarizarse con los datos, para identificar problemas de calidad de los datos, para tener un primer vistazo en los datos o para detectar subconjuntos interesantes para formar hipótesis de información oculta.

3. Preparación de los datos – La etapa de preparación de los datos cubre todas las actividades para construir el conjunto de datos final partiendo de los datos iniciales.

4. Modelado – En esta etapa, varias técnicas de modelado se seleccionan y aplican, y sus parametros son calibrados a valores optimos.

5. Evaluación – En esta etapa, el modelo obtenido se evalúa más a fondo y los pasos realizados para construir el modelo son revisados para tener la certeza de que cumple de manera adecuada los objetivos del negocio.

6. Despliegue – La creación del modelo generalmente no es el final del proyecto. Incluso si el proposito del modelo es incrementar el conocimiento de los datos, el conocimiento obtenido necesitará ser organizado y presentado de un modo que el cliente pueda usarlo.

La secuencia de estas seis etapas no es rigida, y a pesar de que también esta pensada para ser independiente de la herramienta de minería de datos que se quiera usar, esta ligada al software SPSS Modeler, [18].

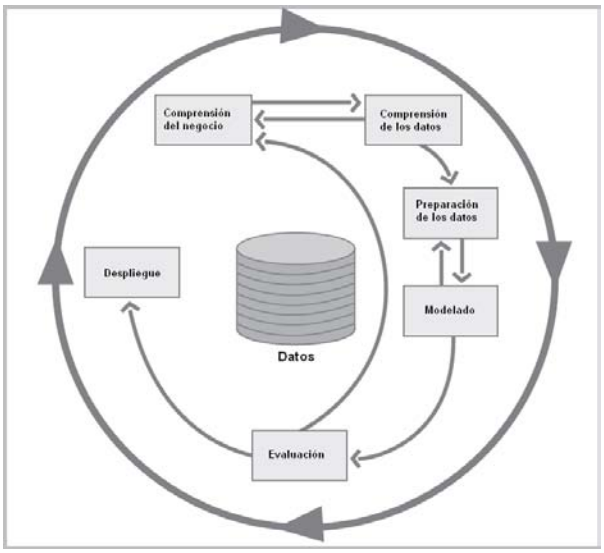

Figura 1.2: Metodología CRISP-DM [18]

La minería de datos se puede dividir según las tareas que se relacionan con la misma, como se ilustra en la Figura 1.3. Algunos ejemplos son: [20]

- Asociación. Se hacen correlaciones entre dos o más elementos para identificar patrones.
- Clasificación. Se usa para describir las características de un grupo de elementos, generando a partir de ellos abstracciones, para poder identificarlos en una clase particular.

- Agrupación (Clustering). A partir de atributos o clases, se busca agrupar datos para obtener una estructura.
- Regresión. Cuando la clase es numérica, y todos sus atributos son numéricos, se puede considerar usar una técnica de regresión, la cual es básica en estadística, para expresar la clase como una combinación lineal de sus atributos.

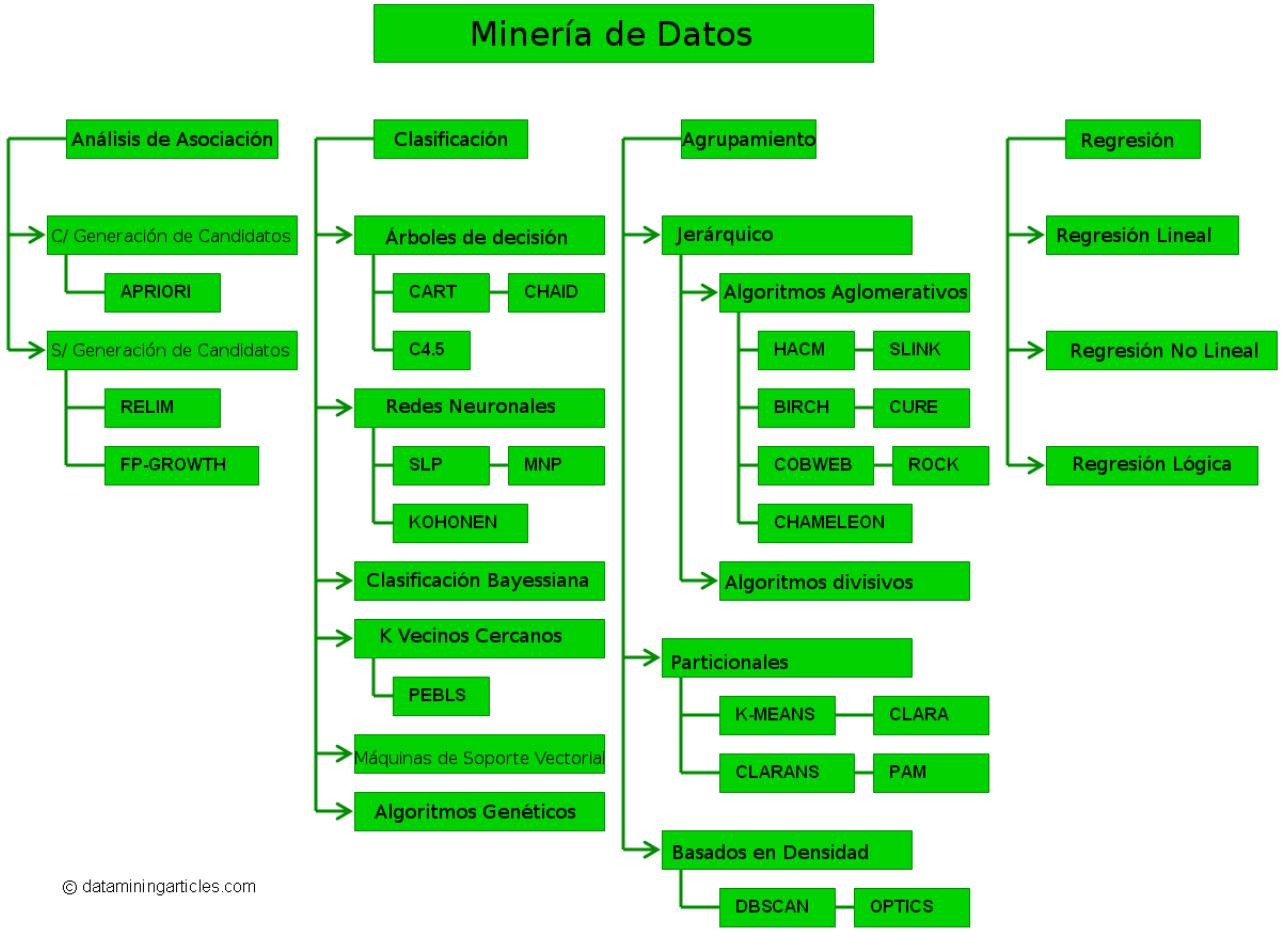

Figura 1.3: Diagrama con los distintos tipos de minería de datos y sus principales algoritmos, [9].

1.1.1. Minería de Patrones Frecuentes

1.1.1.1. Introducción

Una de las aplicaciones más claras de la minería de datos, es el caso del minado de patrones frecuentes. Tomando el ejemplo de la cadena de supermercado que descubrió que los pañales y la cerveza se vendían de manera conjunta, lo que realmente se obtuvo fue un patrón frecuente. En este caso, pensando en las cadenas de supermercado, lo que estas guardan principalmente como información son los tickets de compras y el conocimiento que se busca en este caso son los patrones frecuentes.

Este ejemplo recibe el nombre de análisis del carrito de supermercado (Market Basket) y su objetivo es obtener el conjunto de elementos que son comprados frecuentemente juntos en el supermercado, por medio del análisis de los tickets que se tienen almacenados.

Una vez que se obtienen los conjuntos de elementos frecuentes, estos permitirán extraer reglas de asociación. Las reglas de asociación permiten hacer proposiciones acerca de que tan probable es que dos conjuntos de elementos ocurran al mismo tiempo, [31].

1.1.1.2. Reglas de Asociación

Sea $I = \{I_1, I_2, \ldots, I_m\}$ un conjunto de atributos binarios llamados elementos (items) y sea T una base de datos de transacciones, cada transacción t puede ser representada como un vector, se tiene $t[k] = 1$ si en la transacción t se encontraba el item I_k, y $t[k] = 0$, en otro caso.

Sea X un subconjunto de I, se dice que una transacción t satisface X, si $\forall I_k \in X$, $t[k] = 1$, [15].

Una regla de asociación es una implicación de la forma $X \Rightarrow Y$, donde $X = \{x_1, x_2, \ldots, x_n\}$ y $Y = \{y_1, y_2, \ldots, y_m\}$ son conjuntos de elementos, con $X \subset I$, $Y \subset I$ y $X \cap Y = \emptyset$, [16].

Retomando el ejemplo del supermercado y considerando la notación anterior, ejemplificamos las reglas de asociación, que son ilustradas en la Figura 1.3.

Sea $I = \{$Jugo, Fruta, Agua, Café, Pan$\}$, y en T se encuentra la transacción t con los items $\{$Agua, Café, Pan$\}$. De esta transacción una regla de asociación podría ser

$$\text{Agua, Café} \Rightarrow \text{Pan}$$

con $X = \{$Agua, Cafe$\}$ y $Y = \{$Pan$\}$.

Figura 1.4: Ejemplo de regla de asociación

Al conjunto $X \cup Y$ se le denomina conjunto de elementos (Itemset).

Para que una regla de asociación sea relevante, ésta tiene que satisfacer ciertas métricas, las cuáles son el soporte y la confianza.

El soporte de una regla $X \Rightarrow Y$ es el porcentaje de transacciones en T que contienen a $X \cup Y$, [36]. Y más concretamente:

$$Soporte(X \Rightarrow Y) = P(X \cup Y)$$

El soporte es una métrica útil, ya que si por ejemplo el soporte es muy bajo, puede que la regla está ocurriendo sólo por casualidad y una regla así no será útil.

La confianza de la regla $X \Rightarrow Y$, es el porcentaje de transacciones en T donde X contiene a Y y es calculada con la siguiente fórmula:

$$Confianza(X \Rightarrow Y) = P(X|Y) = \frac{Soporte(X \cup Y)}{Soporte(X)}$$

La confianza es la medida que indica la frecuencia en que aparece Y en las transacciones cuando X se encuentra dentro de la misma.

Dadas las definiciones del soporte y confianza de una regla de asociación, se puede observar que para generar reglas de asociación con una alta confianza, se necesita primero obtener todos los Itemsets frecuentes junto con su soporte.

Formalmente, dado un conjunto de transacciones D, el problema de minar reglas de asociación consiste en generar todas las reglas de asociación que tengan un soporte y una confianza mayor al soporte mínimo y confianza mínima

especificados por el usuario.

Para resolver este problema hay algoritmos muy conocidos, como el Apriori, o el árbol de patrones frecuentes (Frequent-Pattern Tree).

1.1.2. Clasificación

La clasificación es una función de la minería de datos, que coloca elementos de una colección en una categoría o clase, su objetivo es predecir de manera precisa la clase objetivo en cada uno de los elementos, [11]. Ésta comienza con un conjunto de datos, en el cual las clases objetivo ya son conocidas.

Para realizar una clasificación se tiene que aprender de un modelo que describa los distintos tipos de clases para los datos. A esto se le llama Aprendizaje Supervisado. Una vez que el modelo es construido, éste puede ser usado para clasificar nuevos datos.

La clasificación es un proceso de dos etapas, la primera es de aprendizaje, donde se construirá el modelo de clasificación y otra de clasificación, donde el modelo es usado para predecir etiquetas de clase para los datos.

En la etapa de aprendizaje, un clasificador es creado de manera que describa un conjunto de clases, está basado en un algoritmo de clasificación y es construido por medio del "aprendizaje" de un conjunto de entrenamiento, a partir de una base de datos de tuplas y sus etiquetas de clase asociadas.

Una tupla X es representada por un vector n-dimensional de atributos $X = (x_1, x_2, \ldots, x_n)$ con n mediciones de n atributos. Cada tupla se asume que pertenece a una clase predefinida C que está determinada por otro atributo de la base, el cual es llamado Atributo de Etiqueta de Clase. El atributo de etiqueta de clase tiene valores discretos y sin orden.

Las tuplas que de manera individual componen el conjunto de entrenamiento son llamadas tuplas de entrenamiento y son tomadas de manera aleatoria de la base de datos que se está analizando.

La manera más común de representar esta función es con reglas de clasificación, árboles de decisión o fórmulas matemáticas.

Árboles de Decisión

El árbol de decisión es una técnica usada para la clasificación. Éste usa un árbol donde cada hoja tiene una clase asociada y cada nodo interno tiene una función asociada al mismo, [14].

Para clasificar un nuevo elemento, se empieza en la raíz del árbol y se recorre el árbol hasta alcanzar una hoja; en cada nodo interno se evalúa la función asociada con el nuevo elemento. Sin embargo, antes de esto se necesita construir el árbol. Para esto se requiere de un conjunto de entrenamiento, a partir de este conjunto comenzar a particionar el árbol por atributo y por condiciones.

En un inicio se toma el conjunto de entrenamiento, el cual se puede decir es "impuro", ya que contiene elementos de todas las clases y se quiere llegar hasta las hojas donde se tendrán elementos "puros", ya que sólo pertenecen a una clase. Hay métodos para medir la pureza de un conjunto de entrenamiento S, como pueden ser:

- Ganancia de Información. Se busca el atributo con la mayor "ganancia de información" para ser separado. Sin embargo, tiende a seleccionar atributos con un gran número de valores. Además, usa la entropía como medida de impureza.
- Tasa de Ganancia. Es similar a la ganancia de información, pero aplica una normalización a ésta.
- Índice Gini. Mide la divergencia entre las distribuciones de probabilidad de los valores en los atributos.

1.1.3. Agrupación

Contrario al caso de la clasificación donde se cuenta con un conjunto de entrenamiento, y basándose en éste se pueden clasificar los nuevos elementos, existe otro caso en el que se pueden clasificar a los elementos de acuerdo con sus características. De manera que se pueden ir agrupando las que tengan características similares entre ellas.

A esto se le llama Clustering y consiste en agrupar un conjunto de datos en múltiples grupos o conglomerados de tal modo que los elementos dentro de un

cluster son muy parecidos, pero diferentes a elementos dentro de otros Clusters, [27].

A este proceso también se le llama Análisis de Cluster, pero por simplicidad sólo se refieren a él como clustering y se puede ver como el proceso de particionar un conjunto de datos en subconjuntos. Cada subconjunto es un cluster, de tal modo que los elementos dentro del cluster son similares entre ellos, pero son distintos a los elementos que están en otros clusters. Esta partición no se realiza manualmente, sino que se hace aplicacando un algoritmo de clustering. El clustering es muy útil, ya que puede llevar al descubrimiento de grupos que no eran conocidos anteriormente dentro del conjunto de datos.

Dado que un cluster es una colección de datos que son similares entre ellos, y distintos a otros clusters, un cluster puede ser tratado como una clase implícita. En este sentido, el clustering también es llamado clasificación automática.

Al clustering también se le conoce como aprendizaje no supervisado porque no tiene etiquetas de clase, y por esta razón, el clustering es una forma de aprendizaje por observación, en lugar de una de aprendizaje por ejemplos, como lo es la Clasificación.

El clustering como herramienta de minería de datos tuvo sus orígenes en áreas como la biología, seguridad e inteligencia de negocios.

Hay distintos métodos para el clustering, los cuales se dividen en las siguientes categorías:

- Métodos de Particionamiento. Dado un conjunto de objetos, un método de particionamiento divide los datos en k grupos de tal manera que cada grupo debe contener al menos un elemento.
- Métodos Jerárquicos. Se crea una descomposición jerárquica del conjunto de objetos y pueden ser clasificados como aglomerativos o divisores. El aglomerativo comienza con cada objeto en un cluster separado y, sucesivamente, se mezclan los objetos, hasta llegar a un único cluster. El divisor empieza con todos los objetos en un mismo cluster y en cada paso del método se divide en clusters menores.
- Métodos Basados en Densidad. Consiste en hacer crecer el cluster dado hasta que la densidad del cluster exceda cierto umbral.
- Métodos Basados en Rejillas. Cuantifican el espacio de los objetos en un número finito de celdas que forman una estructura de rejilla.

Algoritmo k-medias (k-means)

Este algoritmo puede ser usado sobre cualquier conjunto de datos donde se pueda definir el promedio.

El algoritmo más conocido de clustering es el de k-means o k-medias, y su nombre viene de representar k clusters C_j por la media c_j de sus elementos o puntos, también llamado centroide, [19].

Sea $D = \{x_1, x_2, \ldots, x_n\}$ un conjunto de puntos donde cada $x_i = x_{i_1}, x_{i_2}, \ldots, x_{i_r}$ es un vector en el espacio $X \subseteq \Re^r$, con r siendo el número de atributos en los datos, o el número de dimensión en el espacio de datos. El algoritmo particiona los datos en k clusters, donde cada cluster tiene un centro de cluster, el cual también es llamado el centroide del cluster. Este centroide, es simplemente el promedio de todos los puntos de datos en el cluster, el cual le da el nombre al algoritmo. Dado que se tienen k clusters, entonces hay k promedios.

Al inicio, el algoritmo escoge al azar k puntos de datos como sus centroides. Entonces calcula la distancia entre cada uno de los centroides y cada punto de datos. Cada punto de datos es asignado al centroide que más cerca esté de él. El centroide y sus puntos de datos entonces representan el cluster. Una vez que todos los puntos de datos están asignados, el centroide de cada cluster es re-calculado, usando los puntos de datos en el cluster actual. Este proceso se repite hasta que se satisface alguna condicion específica de paro, [29].

Las condiciones de paro pueden ser:

1. Que no haya re-asignación de puntos de datos a distintos clusters o que sean mínimas.
2. Que no haya cambios de centroides, o que sean mínimos.
3. Que el decremento en el error de la suma de cuadrados (SSE) sea mínimo,

$$SSE = \sum_{j=1}^{k} \sum_{x \in C_j} dist(x, m_j)^2$$

donde k es el número de clusters necesarios, C_j es el j-esimo cluster, m_j es el centroide del cluster, C_j (el vector promedio de todos los puntos en C_j), y $dist(x, m_j)$ es la distancia entre los puntos de datos x y el centroide m_j.

1.2. Panorama general de XML

Debido a la gran cantidad de información que se genera en internet, las empresas tratan de guardar la mayor cantidad posible de la misma en bases de datos.

La información almacenada en las bases de datos tradicionales es conocida como datos estructurados porque está representado en un formato estricto. Es común que se diseñen esquemas de bases de datos para definir su estructura, una vez teniendo esto, el Sistema Manejador de Bases de Datos se asegura de que los datos siguen la estructura y constantes especificadas en el esquema.

Sin embargo, no todos los datos son recolectados e insertados dentro de bases de datos con un diseño muy estructurado, estos datos pueden tener cierta estructura pero no todos los datos recolectados tendrán la misma estructura, como los datos que resultan de usar servicios web. Algunos atributos pueden ser compartidos entre entidades, pero otros pueden existir sólo en pocas entidades; además, los atributos adicionales pueden ser introducidos en algún elemento nuevo en cualquier momento sin necesidad un esquema predefinido.[36]

Este tipo de datos son llamados "Semiestructurados", se distinguen por el hecho de que el esquema se deriva de los datos en lugar de ser declarado por separado de los datos como en el modelo relacional. Más precisamente sus datos son auto descriptibles, llevan información acerca de cuál es el esquema, y ese esquema puede variar arbitrariamente.

Una base de Datos, con datos semiestructurados, puede verse como una colección de nodos. Cada nodo puede ser interno o una hoja. Los nodos hojas tienen asociados datos, los tipos de estos pueden ser atómicos, como lo pueden ser cadenas o números. Cada arista tiene una etiqueta, la cual indica como el nodo a la cabeza de a la arista se relaciona con el nodo en la cola. Un nodo interno, llamado el nodo raíz, no tiene aristas entrantes y representa la base de datos en su totalidad. Cada nodo tiene que ser alcanzable desde la raíz, de lo contrario la estructura no será un árbol.

El lenguaje XML es la representación más conocida del modelo semiestructurado. XML (eXtensible Markup Language) es un lenguaje basado en etiquetas, diseñado originalmente para marcar documentos, de manera muy similar a lo que se conoce como HTML, . [10]. Mientras que en HTML sus etiquetas son acerca de la presentación de la información contenida en los documentos, en XML las

etiquetas tienen la intención de hablar acerca del significado de las partes del documento.

Las etiquetas en XML son texto rodeados de paréntesis triangulares $< \ldots >$. Las etiquetas generalmente vienen en pares que coinciden, con una etiqueta de apertura y otra de cierre. Entre ambas etiquetas puede haber texto, y cualquier número de pares de etiquetas anidadas. Un par de etiquetas y todo lo que hay dentro de ellas se les llama elementos.

Los atributos en XML proveen información adicional que describe elementos. Es importante diferenciar el término atributo, ya que no es usado como siempre en las bases de datos relacionales.

Hay dos clases de elementos, los complejos y los simples. Los complejos están constituidos de otros elementos, mientras que los elementos simples contienen solo valores. Como se observa, esto sigue la definición de datos semiestructurados que se definió anteriormente, ya que los elementos simples coinciden con los nodos hojas, y los complejos con los nodos internos.

Se puede observar el Ejemplo 1.1, donde lo primero que tiene es la etiqueta inicial `<?xml version="1.0" encoding="utf-8"?>`, la cual es una declaración de XML. Esta servirá para identificar nuestro archivo como un documento XML.

Luego de la declaración sigue el elemento raíz del documento XML, en este caso el elemento raíz es `<Pelicula id="1">` , a su vez se puede observar que este incluye un atributo, el cual indica en este caso el id de la película. En la mayoría de los casos el elemento raíz es un elemento complejo, el Ejemplo 1.1 no es la excepción. Un ejemplo de elemento simple es la clasificación, ya que su contenido es un valor.

Ejemplo 1.1: Un ejemplo de XML podría ser una forma de representar unas colección de películas.

```
<?xml version="1.0" encoding="utf-8"?>
<Pelicula id="1">
      <Clasificacion>PG-13</Clasificacion>
      <Idiomas>
            <Idioma>Ingles</Idioma>
      </Idiomas>
      <Titulo>(500) Days of Summer</Titulo>
      <Paises>
            <Pais>USA</Pais>
```

```
        </Paises>
        <Director>Marc Webb</Director>
        <Estreno>07 Aug 2009</Estreno>
        <Actores>
              <Actor>Joseph Gordon-Levitt</Actor>
              <Actriz>Zooey Deschanel</Actriz>
              <Actriz>Chloe Grace Moretz</Actriz>
        </Actores>
        <Anio>2009</Anio>
        <Generos>
              <Genero>Comedia</Genero>
              <Genero>Drama</Genero>
              <Genero>Romance</Genero>
        </Generos>
        <Duracion>95 min</Duracion>
</Pelicula>
```

XML también permite definir los nombres y jerarquía de las etiquetas en un documento adicional, éste es conocido como un documento de esquema, para darle un significado semántico a los nombres de las etiquetas que puede ser intercambiado con otros usuarios.

Se pueden caracterizar los documentos XML en tres tipos:

- Documentos XML enfocados a datos. Estos documentos tienen muchos elementos de datos pequeños que siguen una estructura específica y por lo tanto pueden ser extraídos de una base de datos estructurada. Estos datos son pasados a documentos XML para poder intercambiarlos. Estos siguen normalmente un esquema predefinido que define el nombre de sus etiquetas.
- Documentos XML enfocados a documentos. Estos documentos tienen grandes cantidades de texto, como pueden ser noticias, artículos o libros. Hay pocos o ningún dato estructurados en esos documentos.
- Documentos XML híbridos. Estos documentos pueden tener partes que contienen datos estructurados y otras partes sin estructura. Por lo tanto, pueden o no tener algún esquema.

Los documentos que no siguen un esquema predefinido para los nombres de sus elementos y su correspondiente estructura son llamados documentos XML sin esquema.

1.2.1. Documentos XML Bien Formados y Válidos

Hay dos clases de documentos XML, los bien formados, que son aquellos documentos que no tienen ningun error de sintaxis, y los válidos que además de ser bien formados no incumplen ninguna de las normas establecidas en su esquema.

1.2.1.1. XMLs Bien Formados

Un documento XML está bien formado si cumple ciertas condiciones, en particular estos deben empezar con una declaración XML, para indicar que versión de XML se usará, debe ser seguido también de la estructura del árbol de datos. Esto quiere decir que debe de haber sólo un elemento raíz, y cada elemento debe de incluir un par de etiquetas de inicio y término dentro de las etiquetas de inicio y fin del elemento padre.

Un documento XML bien-formado es correcto en su sintaxis. Esto permite al documento que pueda ser procesado por procesadores genéricos que recorren el documento y crean la representación interna del árbol.

Un documento bien-formado de XML puede no tener esquema, esto es que puede tener cualquier nombre en sus etiquetas para los elementos dentro del documento. En este caso no hay un conjunto de elementos predefinidos que el programa de procesamiento espere. Esto le da al creador del documento la libertad de especificar nuevos elementos, pero limita la posibilidad de interpretar automáticamente el significado o semántica de los elementos dentro del documento.

1.2.1.2. XMLs Válidos

Un criterio más fuerte para un documento XML es que éste sea válido. En este caso, el documento debe de estar bien formado, y seguir un esquema en particular. Esto quiere decir que los nombres usados en los partes de etiquetas deben seguir la estructura especificada en un documento XML adicional, este puede ser de alguno de dos tipos DTD (*Document Type Definition*) o archivo de esquema XML (XSD). Cualquier documento válido que esté conforme al DTD debe de seguir la estructura especificada. La estructura del DTD debe tener primero una

etiqueta raíz, después los elementos y sus estructuras anidadas son especificadas.

Ejemplo 1.2: Un ejemplo de DTD para las películas.

```
<!ELEMENT Pelicula ( Clasificacion,Idiomas,Titulo,
       Paises,Escritores?,Director,Estreno,
       Actores,Anio,Generos,Premios?,Duracion) >
<!ATTLIST Pelicula id ID #REQUIRED >
<!ELEMENT Clasificacion ( #PCDATA ) >
<!ELEMENT Idiomas ( Idioma+ ) >
<!ELEMENT Idioma ( #PCDATA ) >
<!ELEMENT Titulo ( #PCDATA ) >
<!ELEMENT Paises ( Pais+ ) >
<!ELEMENT Pais ( #PCDATA ) >
<!ELEMENT Escritores ( Escritor+ ) >
<!ELEMENT Escritor ( #PCDATA ) >
<!ELEMENT Director ( #PCDATA ) >
<!ELEMENT Estreno ( #PCDATA ) >
<!ELEMENT Actores ( (Actor|Actriz)+ ) >
<!ELEMENT Actor ( #PCDATA ) >
<!ELEMENT Actriz ( #PCDATA ) >
<!ELEMENT Anio ( #PCDATA ) >
<!ELEMENT Generos ( Genero+ ) >
<!ELEMENT Genero ( #PCDATA ) >
<!ELEMENT Premios ( #PCDATA ) >
<!ELEMENT Duracion ( #PCDATA ) >
```

Se puede observar en Ejemplo 1.2 lo que se mencionó anteriormente, se tiene a la raíz con cada uno de sus elementos. Seguido de esta están todos las definiciones de los elementos que le siguen en jerarquía, en el caso de los simples su contenido dice `#PCDATA` , éste se refiere a *parsed character data*, lo cual se puede ver simplemente como el texto que se encuentra dentro de las etiquetas de inicio y fin del elemento.

También está un atributo en película, éste será obligatorio, ya que tiene la palabra clave `#REQUIRED` . En el caso de los elementos de la raíz, hay dos opcionales, los cuales son Escritores y Premios. Debe de haber escritores en una película, pero puede darse el caso que no se tenga conocimiento de ellos. En el caso de los premios es más claro, ya que no toda película tiene premios, [6].

La DTD de XML es bastante adecuada para especificar estructuras de árboles, con elementos requeridos, opcionales y repetidos, y distintos tipos de atributos se tienen bastantes limitaciones. Por ejemplo, los tipos de datos en el DTD no son

muy particulares. También, el DTD tiene su propia sintaxis especial y requiere procesadores especializados. Otra desventaja muy grande es que todos los elementos de la DTD están forzados a seguir el orden especificado en el documento, por lo que los elementos en desorden no están permitidos.

Estos contratiempos dieron pie al desarrollo del esquema XML (XSD), un lenguaje más general pero también más complejo para especificar la estructura y elementos de los documentos XML.

1.2.1.3. Esquema XML (XSD)

El propósito del esquema XSD es definir y describir una clase de documentos XML por medio del uso de componentes del esquema para restringir y documentar el significado, uso y relaciones de las partes que lo constituyen: tipos de datos, elementos, y sus contenidos y atributos. Así también los esquemas pueden proveer una especificación adicional a la información del documento, tales como la normalización, atributos por omisión, y valores de los elementos, [8].

Un esquema XSD es un conjunto de componentes tales como la definición de tipos y declaración de elementos. Sirven para evaluar la validez de un elemento bien formado y de sus atributos.

De igual manera que el DTD, el esquema está basado en un modelo de árbol, con elementos y atributos como la principal estructura. Así como también toma conceptos de las bases de datos, como las llaves, referencias e identificadores.

Sin embargo, una clara diferencia con el DTD, es que el XSD está escrito en XML y como tal tiene las ventajas del mismo, en especial la de ser más fácil de entender. Otra ventaja de estar en XML es que no se necesita de un analizador sintáctico para poder leer este documento.

Ejemplo 1.3: Un ejemplo de esquema para el caso de las películas.

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="Pelicula">
    <xs:complexType>
      <xs:sequence>
        <xs:element type="xs:string" name="Clasificacion"/>
        <xs:element name="Idiomas">
          <xs:complexType>
            <xs:sequence>
              <xs:element type="xs:string" name="Idioma"/>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
        <xs:element type="xs:string" name="Titulo"/>
        <xs:element name="Paises">
          <xs:complexType>
            <xs:sequence>
              <xs:element type="xs:string" name="Pais"/>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
        <xs:element type="xs:string" name="Director"/>
        <xs:element type="xs:string" name="Estreno"/>
        <xs:element name="Actores">
          <xs:complexType>
            <xs:sequence>
              <xs:element type="xs:string" name="Actor"/>
              <xs:element type="xs:string" name="Actriz"/>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
        <xs:element type="xs:short" name="Anio"/>
        <xs:element name="Generos">
          <xs:complexType>
            <xs:sequence>
              <xs:element type="xs:string" name="Genero"/>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
        <xs:element type="xs:string" name="Duracion"/>
      </xs:sequence>
        <xs:attribute type="xs:int" name="id"/>
```

```
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Una desventaja que se puede observar rápidamente con su contraparte DTD, es que se tuvo que escribir bastante más para poder expresar la misma información, sin embargo, da mayor información de los datos, ya que se pueden especificar, por ejemplo, los tipos de datos de cada elemento.

1.2.2. Interfaces de XML

El formato XML es una manera de representar los documentos en texto, pero una vez que ha sido leído en memoria, se representa como un árbol. Hay distintas maneras de representar y acceder a este árbol.

El modelo más conocido de datos para almacenar y procesar los árboles XML se llama Modelo Objeto Documento W3C, o en su abreviatura DOM (*Document Object Model*). DOM fue diseñado originalmente para manejar HTML en los navegadores Web, sin embargo, el DOM de XML es una especificación distinta, pero es soportada por todos los navegadores modernos. Este modelo lee todo el documento y construye el árbol en memoria, además de proveer una interfaz para la manipulación y acceso a la estructura del mismo, [3]. Esto permite recorrer el árbol, borrar secciones del árbol, reorganizarlo, entre otras operaciones.

Sin embargo, DOM también tiene desventajas, El ejemplo más claro de esto es que DOM construye el árbol en memoria y si el archivo es muy grande, se requerirá de una gran cantidad de memoria. Así también como se tiene que leer todo el archivo desde un inicio, hasta que no se haya leído todo no se podrá hacer uso de DOM, por lo que esto puede suponer un retraso.

Hay otro modelo llamado SAX, este permite procesar los documentos XML al vuelo, notificando al programa de procesamiento si se encuentran etiquetas de inicio o de fin. Este modelo facilita la manipulación de documentos extensos.

SAX está basado en eventos, el cual notifica cuando hay una etiqueta de inicio, de fin, y texto. En este caso cada persona tiene que decidir que eventos son importantes para cada quien, así también como definir una estructura para estos documentos. SAX se aprovecha de las desventajas de DOM, ya que no guarda todo el documento en memoria, sólo se encarga del análisis sintáctico del

mismo. Así mismo, cuando hay un evento que se definió anteriormente llega el aviso inmediatamente, no se tiene que esperar a toda la lectura del archivo para esto.[7]

1.2.3. XSLT

El lenguaje *XSL Transformations* (XSLT) sirve para transformar documentos XML en otros documentos XML, con el objetivo de que estos cumplan con un formato establecido por el usuario y evitar la necesidad de modificarlos de manera manual, [4]. Con XSLT se pueden agregar o quitar elementos y atributos hacia el archivo de salida, así como reacomodar u ordenar elementos. XSLT puede ser visto como una extensión de *Extensible Stylesheet Language* (XSL). XSL es un lenguaje para formatear un documento XML. XSLT muestra cómo transformar el árbol del XML original en uno que dará origen a un nuevo documento XML, el cuál, puede ser distinto en estructura.

Un ejemplo de una XSLT es tomar el Ejemplo 1.4, el cual es un catalogo de discos y sus canciones, y el archivo XSL (Ejemplo 1.5).

Primero se establece la estructura del documento resultante así como el orden en que son procesados los nodos (Lineas 6-10), luego se toma el atributo artista dentro de la etiqueta Album (Linea 11), establece que la nueva etiqueta se llamara Artista (Linea 12) y su valor sera el nombre del artista que se tomo de la linea 11 (Linea 13) dando como resultado el nuevo XML (Ejemplo 1.6).

Ejemplo 1.4: Un ejemplo de documento XML para ser usado con XPath y XSL.

```
<?xml version="1.0" encoding="utf-8"?>
<Catalogo>
  <Album artista="Daft Punk" titulo="Discovery">
    <Cancion>One More Time</Cancion>
    <Cancion>Aerodynamic</Cancion>
    <Cancion>Digital Love</Cancion>
    <Cancion>Harder, Better, Faster, Stronger</Cancion>
    <Cancion>Superheroes</Cancion>
    <Cancion>Something About Us</Cancion>
  </Album>
  <Album artista="Soda Stereo" titulo="Me Veras Volver">
    <Cancion>Tratame Suavemente</Cancion>
    <Cancion>Nada Personal</Cancion>
```

```
    <Cancion>Persiana Americana</Cancion>
    <Cancion>De Musica Ligera</Cancion>
    <Cancion>Signos</Cancion>
  </Album>
</Catalogo>
```

Ejemplo 1.5: Archivo de ejemplo de XSL.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:output method="xml"
  version="1.0" encoding="UTF-8" indent="yes"/>
<xsl:template match="/">
    <xsl:element name="root">
      <xsl:apply-templates select="Catalogo/Album/@artista"/>
    </xsl:element>
</xsl:template>
<xsl:template match="Catalogo/Album/@artista">
    <xsl:element name="Artista" >
      <xsl:value-of select="."/>
    </xsl:element>
</xsl:template>
</xsl:stylesheet>
```

Ejemplo 1.6: XML resultante de la transformación XSLT.

```
<?xml version="1.0" encoding="UTF-8"?>
<root>
  <Artista>Daft Punk</Artista>
  <Artista>Soda Stereo</Artista>
</root>
```

1.2.4. XPath

El lenguaje XPath sirve para recorrer y seleccionar etiquetas de un documento XML y fue diseñado para ser usado de igual manera por XSLT y XPointer, el cual es una notacion para apuntar a elementos especificos dentro de un documento XML [2, 4, 5]. Obtiene su nombre debido a su uso como una notación de ruta (*path*) para navegar a través de la estructura jerárquica de un documento XML.

Tal como lo hace DOM, XPath modela al documento XML como un árbol de nodos, así como valores atómicos, que pueden ser enteros, cadenas y lógicos. Además una expresión en XPath es un mecanismo para navegar a través del XML y seleccionar nodos del mismo. Una expresión de XPath es parecida a lo que es una consulta en SQL.

Algunos expresiones de XPath a partir de el ejemplo Ejemplo 1.4, podrían ser:

- Seleccionar todos los álbumes

  ```
  /Catalogo/Album
  ```

- Obtener el atributo artista de todos los álbumes

  ```
  /Catalogo/Album/@artista
  ```

- Obtener todas las canciones de todos los álbumes

  ```
  /Catalogo/Album/Cancion
  ```

- Obtener todos los álbumes de Daft Punk

  ```
  /Catalogo/Album[@artista="Daft Punk"]
  ```

1.3. Minería de Datos en documentos XML

El minado de documentos XML es el uso de las técnicas de minería de datos para descubrir de manera automática y extraer información de documentos XML. El hecho de que los datos están representados de manera jerárquica es un reto para la minería. Además de lo anterior se puede tener que los documentos XML estén diseñados con mínimas restricciones y ser flexibles. A pesar de que esto es una de las mayores virtudes de XML, también esto hace que el manejo de documentos sea más complicado.

Para evitar la perdida de información, la minería en XML incluye minar de igual manera la estructura del documento, así como su contenido. Minar la estructura es hacer minería en el esquema del XML. Mientras que el minado del contenido involucra el análisis del contenido y aclaración de la estructura. El análisis de contenido tiene que ver con el análisis de los textos dentro del documento XML. La aclaración de la estructura se ocupa de determinar los documentos similares en función de su contenido.

Minado estructural del XML

Los documentos XML son datos semiestructurados y minarlos puede brindarnos un mayor contexto de los mismos. Dado que XML provee un mecanismo para nombrar las etiquetas, facilita tener un mayor conocimiento de los datos, y si se hace uso de la minería de datos se puede tener una recuperación de información más detallada. El minado estructural de XML es realizar el minado del esquema, incluyendo en este el minado de la intraestructura y de la interestructura.

El minado de la intraestructura se refiere a la estructura dentro del documento XML. En este caso el conocimiento es sobre la estructura interna. La estructura del documento XML, puede ser leída desde el mismo documento, o por medio del esquema. Y como se vio, el esquema proporciona una descripción definitiva de todos los documentos que están basados en él, [37].

Para este caso se puede aplicar los tipos de minería de datos que se vieron anteriormente, los cuales son clasificación, clustering y reglas de asociación.

Un ejemplo claro es el clustering, aquí una posibilidad es usar la minería para identificar elementos similares entre los documentos. La estructura se puede inferir y modelar como un árbol con etiquetas. Cada nodo en el árbol tiene información acerca de ese elemento. Usando un algoritmo de clustering, se toma una colección de árboles y se agrupan basándose en su semejanza semántica y estructural. Estas semejanzas son usadas para generar un nuevo esquema, y a partir de esto se realiza una generalización de los esquemas, el nuevo esquema sería una superclase del conjunto de esquemas de entrenamiento. Así, este conjunto generado puede ser usado para clasificar a los nuevos esquemas.

Otro ejemplo es el descubrimiento de reglas de asociación, en este caso estas pueden describir las relaciones entre las etiquetas que se encuentran a la vez en documentos XML. Como el esquema se puede representar con una estructura de árbol, cada rama del árbol se puede considerar una transacción. Transformando el árbol de la estructura de XML en uno de transacciones, es posible generar reglas de la siguiente forma: si un documento XML contiene la etiqueta `<nombre>` entonces el 90 % de las veces también esta la etiqueta `<apellido>`, [33].

Capítulo 2

Algoritmos para generación de reglas de asociación

2.1. Introducción

En el caso del minado de patrones frecuentes, lo que se busca son las reglas de asociación y éstas son encontradas a partir de itemsets frecuentes. Para poder encontrar estas reglas, se tienen varios algoritmos.

Estos algoritmos tienen un por qué, un claro ejemplo de esto sería que a partir de nuestros items, se almacenaran todas las transacciones en memoria y se contaran todos sus elementos. Se revisarían todas las transacciones y sería necesaria solo una iteración. De esta manera, se podrían generar los itemsets de distintos tamaños, por medio de ciclos y con solo un conteo. Al final, sólo se escogerían los itemsets que están por encima del umbral definido por el usuario, así se tendrían todos los itemsets frecuentes.

Sin embargo, este método fallaría si hubiera demasiados itemsets como para contarlos todos desde memoria. Los siguientes algoritmos están diseñados para reducir el número de itemsets a ser contados, pero, esto implica un costo adicional, ya que en estos casos se aumentan el número de iteraciones a realizarse, en lugar de ser sólo una como se planteaba inicialmente.

2.2. Algoritmo Apriori

Apriori es un algoritmo propuesto por R. Agrawal y R. Srikant en 1994 para hacer minería de datos en los conjuntos de elementos frecuentes para reglas de asociación binarias, [15].

Este algoritmo es iterativo y consiste de dos partes:

1. Se generan todos los itemsets frecuentes, donde un itemset frecuente es un itemset cuyo soporte es mayor o igual que cierto umbral.
2. Se generan todas las reglas de asociación de los itemsets frecuentes de confianza; es decir, aquellas reglas de asociación con una confianza mayor o igual a cierta confianza mínima proporcionados por el usuario. En otras palabras, para cada itemset frecuente I, se encuentran todos los subconjuntos no vacíos de I. Después, para cada subconjunto a encontrado, la salida será una regla de la forma $a \rightarrow (I - a)$, sólo si esta regla tiene confianza mayor o igual a cierta confianza mínima.

Para poder hacer el primer paso se tienen que encontrar los itemsets frecuentes de tamaño 1, por medio de un escaneo a la base de datos, donde se guardara un conteo de repeticiones que tiene cada elemento, se toman aquellos elementos que satisfacen un soporte mínimo. Al conjunto resultante se le nombrara L_1.

Una vez que se tiene L_1, éste se usa para encontrar a L_2, el cual es el conjunto de itemsets con tamaño 2. Éste a su vez es usado para encontrar L_3, y así sucesivamente, hasta que no se puedan encontrar más itemsets.

Para encontrar cada L_k se necesita una revisión completa a la base de datos por cada uno de ellos. Por lo que es necesario optimizar la generación de cada L_k, así que se tiene que reducir el espacio de búsqueda usando la Propiedad Apriori.

Propiedad Apriori: Si un itemset es mayor o igual a un soporte mínimo, entonces cada subconjunto no vacío de este itemset también es mayor o igual a este soporte mínimo. Se ilustra un ejemplo en la Figura 2.1.

Con esta propiedad y el uso del umbral de soporte, se pueden eliminar una gran cantidad de itemsets que no serán frecuentes. El algoritmo Apriori genera todos los itemsets frecuentes por medio de varias iteraciones.

Si el conjunto {Jugo, Fruta, Agua} es frecuente, también lo es {Jugo, Fruta}.

Figura 2.1: Ejemplo de Propiedad Apriori

Algoritmo 1 Algoritmo Apriori

```
1: function APRIORI(D)
2:     L_1 = { itemsets tamaño 1 }
3:     for (k = 2; L_{k-1} ≠ ∅; k++) do
4:         C_k = APRIORI-GEN (L_{k-1}) //Nuevos Candidatos
5:         for all transacciones t ∈ D do
6:             C_t = subconjunto(C_k, t) //Candidatos dentro de t
7:             for all candidatos c ∈ C_t do
8:                 c.count++
9:             end for
10:        end for
11:        L_k = { c ∈ C_k | c.count > sopMin }
12:    end for
13:    return ⋃_k L_k
14: end function
15: function APRIORI-GEN(L_{k-1})
16:    C_k = ∅ //inicializa el conjunto de candidatos
17:    for all l_1, l_2 ∈ L_{k-1} con l_1 = {i_1, ..., i_{k-2}, i_{k-1}} y l_2 = {i_1, ..., i_{k-2}, i'_{k-1}}
       y i_{k-1} < i'_{k-1} do //encuentra los itemsets que difieren en el ultimo elemento
18:        c = {i_1, ..., i_{k-1}, i'_{k-1}} //unimos f_1 y f_2
19:        C_k = C_k ∪ {c} //agrega el itemset a los candidatos
20:        for all (k − 1) subconjuntos s de c do
21:            if s ∉ L_{k-1} then elimina c de C_k //borra c si uno de sus subcon-
           juntos no esta
22:            end if
23:        end for
24:    end for
25:    return C_k
26: end function
```

Para poder hacer cada una de las k iteraciones, se necesitan tres pasos:

1. Se empieza con el conjunto resultado de la iteración $k-1$, con esto se genera el itemset de candidatos L_k, los cuales son los posibles itemsets frecuentes.
2. Se revisa la base de datos de transacciones y se hace un conteo del soporte de cada elemento del itemset L_k, con esto se evita la carga de todos los datos en memoria para hacer este procesamiento. En lugar de eso únicamente se tiene una transacción en memoria.
3. Al final de cada iteración se determinan que candidatos son frecuentes, de acuerdo con su soporte.

La salida final del algoritmo es un conjunto F con todos los itemsets frecuentes.

Este algoritmo hace uso de otra función, la generadora de candidatos, esta tiene un paso de unión y otro de podado, [16, 29]. Se presenta un ejemplo en la Figura 2.2 y en la Figura 2.3, en donde se generan los candidatos correspondientes y se comparan con un soporte mínimo de dos, basandose en la lista de transacciones.

1. Paso de Unión. Une dos itemsets frecuentes para producir un posible candidato c. Los dos itemsets frecuentes f_1 y f_2 tienen exactamente los mismos elementos, y sólo difieren en el último. Al final se agrega c a una lista de candidatos.
2. Paso de Podado. Un candidato c del paso 1 puede no ser un candidato final. Este paso determina si todos los $k-1$ subconjuntos de c están en F_{k-1}. Si alguno no está, entonces c no puede ser frecuente, esto por la propiedad Apriori y se borra de la lista de candidatos.

Se puede observar que se realizan 3 iteraciones dado el conjunto de transacciones D (Figura 2.2). En la primera iteración se calcula el soporte que tiene cada itemset, en este caso el soporte mínimo es de 2, por lo que todos los elementos dentro de C_1 lo cumplen y son agregados a L_1. A partir de L_1 se genera C_2, el cual es el conjunto de itemsets de tamaño 2. Después se vuelve a hacer el cálculo del soporte y se filtra C_2 en base al soporte mínimo, en esta iteración si hubo elementos que no cumplen con esta condición y fueron descartados. Por último, se genera C_3 y se calcula el soporte de sus elementos, y son filtrados.

Algunos datos que se saben sobre el algoritmo Apriori son los siguientes:

TID	Lista de Elementos
T100	I1, I2, I5
T200	I2, I4
T300	I2, I3
T400	I1, I2, I4
T500	I1, I3
T600	I2, I3
T700	I1, I3
T800	I1, I2, I3, I5
T900	I1, I2, I3

Figura 2.2: Ejemplo de un conjunto de transacciones[27]

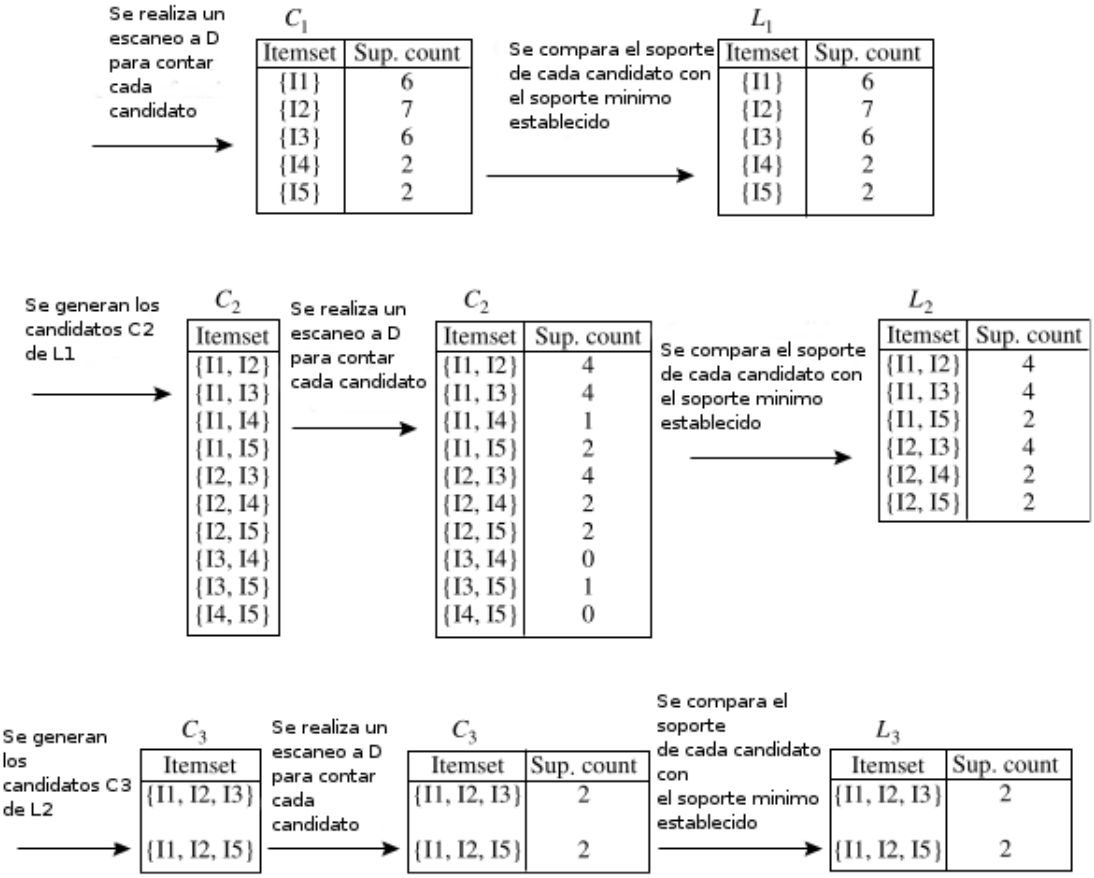

Figura 2.3: Ejemplo del Apriori aplicado en las transacciones de la Figura 2.2 [27]

- En teoría es un algoritmo exponencial. Sea m el número de elementos en I, donde I son todos los items. Entonces el espacio de todos los itemsets es de $\mathcal{O}(2^m)$, dado que cada item puede ser o no ser un itemset. En la práctica, este algoritmo aprovecha la escasez de los datos, así como la propiedad Apriori para hacerlo lo más eficiente posible. La escasez de los datos se refiere a que se manipula una gran cantidad de información, sin embargo, en cada transacción sólo existe una pequeña cantidad de la misma.
- Es escalable, ya que no se almacena toda la información en memoria. Sólo se consulta la información K veces, donde K es el tamaño del itemset más grande.
- Al final de cada iteración se determina que candidatos son frecuentes, basándose en su soporte.

El algoritmo Apriori original puede ser optimizado de diversas maneras, a continación se describen dos:

- Tablas Hash. Una técnica basada en tablas hash se puede usar para reducir el tamaño de los itemsets candidatos. Ya que proporciona un acceso más rápido propio de la estructura, además de que los itemsets pueden ser usados como llaves, mientras que el valor sería el conteo del itemset.
- Reducción de Transacciones. Dado que una transacción que no contenga ningún itemset de tamaño k no puede contener ningún itemset de tamaño $k+1$, por la propiedad Apriori. Entonces esta transacción puede ser marcada o removida, ya que para iteraciones próximas, estas transacciones no serán útiles.

2.3. Algoritmo FP-Growth

Como se observó en el algoritmo Apriori, la generación de candidatos es un proceso costoso, por lo que se propuso otro algoritmo para obtener reglas de asociación sin la necesidad de hacer este proceso.

Para esto se propuso una nueva estructura, la cual es el árbol de patrones frecuentes (FP-Tree), la cual es básicamente un árbol de prefijos, que permite almacenar información comprimida y crucial sobre los patrones, así como el

desarrollo de una técnica basada en este árbol, la cual se le nombro FP-Growth, [25].

Consiste en dos pasos:

1. Se revisa la base de datos de transacciones. Se calcula el soporte de cada item, descartando aquellos que no cumplan soporte mínimo. Una vez hecho esto se ordenan los items frecuentes L en orden descendente.

2. Se crea el nodo raíz del FP-Tree T, y se le etiqueta como null. Y por cada transacción $trans$ en la base de datos transaccional se realiza:

   Seleccionar y ordenar los items frecuentes en $trans$ dependiendo del orden de L. Se denotan la lista de elementos frecuentes ordenados dentro de $Trans$ como $[p|P]$, donde p es el primer elemento de la lista y P es el resto de la lista.

   Una vez teniendo esto de llama a la función insertarArbol($[p|P], T$), la cual depende de los parámetros. Si T tiene un hijo N, de tal manera que $N = p$; es decir, que si ya se tenía a N en el FP-Tree, entonces se incrementa el contador de N en una unidad. En otro caso se crea un nuevo nodo N, se inicializa en 1, su enlace padre está ligado a T y su enlace propio está ligado a los nodos con el mismo nombre, por medio de los nodos de la estructura. Si P no es vacio, entonces se llama insertarArbol(P, N) de manera recursiva.

Usando este algoritmo, se necesitan exactamente dos consultas a la base de datos de transacciones, una para obtener los itemsets frecuentes y otra para construir el FP-Tree. El costo de insertar una transacción $Trans$ dentro del árbol es de $\mathcal{O}(|Trans|)$, donde $|Trans|$ es el número de items frecuentes en $Trans$.

Así como el algoritmo Apriori, el algoritmo FP-Growth empieza obteniendo el soporte para cada item. Como también es un parámetro de entrada el soporte mínimo, se descartan aquellos items que no satisfagan esta condición. La Figura 2.4 ilustra el funcionamiento del mismo, basandose nuevamente en el conjunto de transacciones D (Figura 2.2).

El primer paso es identico al del algoritmo Apriori, el cual nos da el conjunto de itemsets de tamaño uno, asi como su conteo de soportes, donde el soporte minimo es dos. El conjunto de elementos como se ve en la Figura 2.4, esta ordenado de manera descendente en base a su soporte y se le denota L. Despues de esto se crea su nodo raíz etiquetandolo como null, se consulta a D nuevamente y se procesa transacción por transacción.

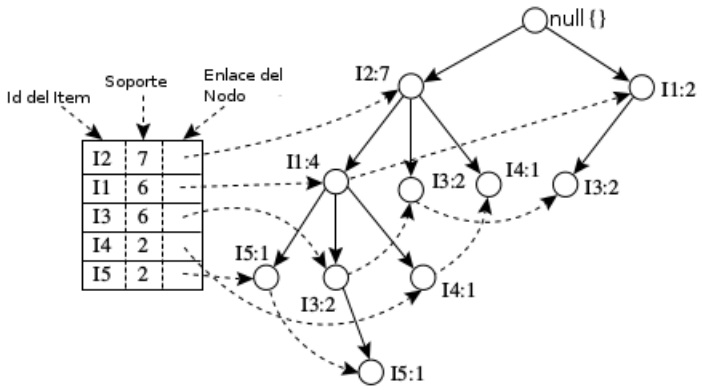

Figura 2.4: Ejemplo de FP-Growth, [27].

Primero se toma a T100, cuyos elementos ($I1$, $I2$, $I5$) son procesados en el orden establecido en L, de manera que el resultado es $I2$, $I1$, $I5$ y se construye la primera ramificacion del árbol, donde el primer nodo es $I2$ con el valor asociado de 1, este a su vez tiene otro nodo hijo que es $I1$ tambiéna con el valor de 1, que a su vez tiene como nodo hijo a $I5$ con valor de 1.

La segunda transaccion T200 ($I2$, $I4$) se va agregando al árbol actual, como el nodo $I2$ esta presente solo se incrementa en una unidad su valor, siendo este ahora 2 y dado que el nodo $I4$ no esta se le agrega como nodo hijo de $I2$ con valor de 1.

Se realiza esto con el resto de transacciones, y al termino de esto se agrega una tabla de elementos con sus ocurrencias en el árbol y se liga al mismo por medio de una cadena de enlaces nodales, esto con el objetivo de facilitar el recorrido del árbol.

2.4. Generación de Reglas de Asociación de los Itemsets Frecuentes

A partir de los Itemsets Frecuentes generados por el Apriori, se pueden crear las reglas de asociación de manera sencilla. Se vio en el capítulo anterior, que las reglas necesitaban dos parámetros, un soporte y una confianza, el algoritmo

Apriori hace uso del soporte, sin embargo, hasta este punto es donde se usará la confianza.

Para generar las reglas de asociación se necesitan de dos pasos:

- Para cada itemset frecuente l, se generan todos sus subconjuntos no vacíos.
- Para cada subconjunto no vacío s de l, la regla $s \Rightarrow (l - s)$ es válida si su confianza es mayor a la confianza mínima establecida.

Como los algoritmos de generación de itemsets frecuentes se encargan de descartar los itemsets que no cumplen con el soporte mínimo, se puede asumir que cada regla generada con los pasos anteriores cumple también con el soporte mínimo.

2.5. Reglas de Asociación en Documentos XML

El minado de reglas de asociación de documentos XML es distinto al de reglas del modelo relacional, ya que la información puede estar estructurada de manera distinta en el formato XML. Generalmente, descubrir reglas de asociación significa encontrar una relación entre las subestructuras de uno o más documentos XML.

También se puede establecer un contexto para las reglas de asociación en XML, las cuales hacen referencia a las secciones del documento que se minaran realmente. Esto debido a que en ocasiones no es necesario hacer minería en toda la información contenida en el documento XML. Por lo que la selección de un contexto, es la capacidad del usuario para definir ciertas limitaciones en el conjunto de transacciones.

En estos documentos, el conjunto de transacciones está dado por una lista de nodos complejos que están formados mediante consultas de los documentos para rutas específicas. Por lo que un solo nodo complejo forma una transacción, y los items son los nodos simples que son hijos del nodo transacción. En el Ejemplo 2.1 se puede observar esta situacion, donde las transacciones son los nodos complejos que estan etiquetados como *Pelicula*, y uno de sus items el nodo etiquetado como *Director*.

A diferencia del modelo relacional, donde una transacción contiene un número fijo de items, el cual es el número de atributos o de columnas en la tabla correspondiente. Una transacción en XML puede tener un número distinto de items, que varía según el nivel de anidamiento en el documento.

Ejemplo 2.1: En los ejemplos que se han venido manejando, las transacciones son todos los nodos complejos con la etiqueta Película y sus items son todos sus nodos hijos.

```
<?xml version="1.0" encoding="utf-8"?>
<Pelicula id="1">
      <Idiomas>
            <Idioma>Ingles</Idioma>
      </Idiomas>
      <Titulo>(500) Days of Summer</Titulo>
      <Paises>
            <Pais>USA</Pais>
      </Paises>
      <Director>Marc Webb</Director>
      ...
</Pelicula>
<Pelicula id="2">
      ...
</Pelicula>
      ...
<Pelicula id="n">
      ...
</Pelicula>
```

Tomando en cuenta que la minería de datos es parte del Descubrimiento de Conocimiento, y que éste a su vez incluye un preprocesamiento, en el caso de los documentos XML se sigue realizando de igual manera, ya que en un inicio para poder aplicar un algoritmo que genere los itemsets frecuentes, es necesario que los documentos tengan cierto formato, como podria ser el mostrado en Ejemplo 2.1 donde cada nodo es una transacción.

Como se vio, XML se representa por medio de un árbol, por lo que uno de los enfoques más claros sería buscar reglas de asociación basadas en subárboles dentro de los documentos XML, en lugar de los items binarios como se hace en los algoritmos tradicionales, [23, 30, 39]. Un ejemplo es el de la Figura 2.5, en donde si una persona de sexo masculino compra un DVD de Star Wars IV, probablemente comprará los DVDs de Star Wars V y Star Wars VI.

Figura 2.5: Ejemplo de una regla de asociación basada en subárboles

También se han propuesto frameworks para obtener las reglas de asociaciones del documento, un ejemplo es XAR-Miner [38], sin embargo, como bien lo menciona este artículo, si se propone una solución muy general, se pueden extraer reglas de asociación muy triviales, pero si se propone una solución muy particular es probable que no se tenga una cantidad suficiente de datos para encontrar reglas de asociación relevantes.

En este caso el framework recibe el documento XML original, pero lo transforma a un árbol XML indexado, o a una base relacional múltiple. Dependiendo del tamaño que se ocupe en memoria el framework decide cuál se usará.

Otra manera de obtener las reglas es por medio de plantillas, esta es por medio de un framework también, y el usuario puede especificar la plantilla que se usará para encontrar las reglas, esto permite especificar qué tipo de relación se está buscando. En el Ejemplo 2.2 se puede observar que se define un nodo raíz (dblp), después al nodo hijo de la raíz se le tomará como comodín y sólo se tomarán en cuenta los valores que estén dentro de la etiqueta "author" buscando asociaciones entre los coautores [17, 22].

Ejemplo 2.2: Plantilla para encontrar coautores.

```
<dblp><*><author>?</author></*></dblp> =>
<dblp><*><author>?</author></*></dblp>
```

En el caso del algoritmo Apriori aplicado en XML, se han utilizado aproximaciones con expresiones en XQuery para extraer reglas de asociación de los datos en XML, sin embargo esta técnica es limitada, ya que para datos XML más complejos, se requiere transformar los datos para poder aplicar las expresiones en XQuery, [26].

En el Ejemplo 2.3 se tiene una representación de itemsets en XML a los cuales se les puede aplicar una expresión XQuery (Ejemplo 2.4). En esta expresión se define que su confianza mínima sera 1, el documento sobre el cual se basará

(large.xml) y aquellas reglas que cumplan con el umbral de confianza se agregarán al documento XML resultante, mostrando cuáles son sus métricas, así como cuál es el antecedente y el consecuente (Ejemplo 2.5).

Ejemplo 2.3: Itemsets con su soporte representados en XML (large.xml).

```
<largeItemsets>
      <largeItemset>
            <items>
                  <item>d</item>
                  <item>b</item>
            </items>
            <support>0.4</support>
      </largeItemset>
      <largeItemset>
            <items>
                  <item>d</item>
                  <item>c</item>
            </items>
            <support>0.4</support>
      </largeItemset>
      <largeItemset>
            <items>
                  <item>b</item>
                  <item>c</item>
            </items>
            <support>0.4</support>
      </largeItemset>
      <largeItemset>
            <items>
                  <item>d</item>
                  <item>c</item>
                  <item>b</item>
            </items>
            <support>0.4</support>
      </largeItemset>
</largeItemsets>
```

Ejemplo 2.4: Codigo XQuery para generar reglas de asociacion en base al Ejemplo 2.3.

```
let $minconf := 1.00
let $src := document(/large.xml)//largeItemset
```

```
for $itemset1 in $src
let $items1 := $itemset1/items/*
      for $itemset2 in $src
      let $items2 := $itemset2/items/*
      where count($items1) > count($items2) and
            count(commonIts($items1, $items2)) =
            count($items2) and $itemset1/support div
            $itemset2/support  $minconf
      return <rule support ={$itemset1/support}
            confidence = {($itemset1/support*1.0) div
                        ($itemset2/support*1.0)}>
                  <antecedent> {$items2} </antecedent>
                  <consequent>
                        {removeItems($items1,$items2)}
                  </consequent>
            </rule>
```

Ejemplo 2.5: Reglas de asociación obtenidas por XQuery.

```
<rules>
      <rule support="0.4" confidence="1.0">
            <antecedent>
                  <item>d</item>
                  <item>b</item>
            </antecedent>
            <consequent>
                  <item>c</item>
            </consequent>
      </rule>
      <rule support="0.4" confidence="1.0">
            <antecedent>
                  <item>d</item>
                  <item>c</item>
            </antecedent>
            <consequent>
                  <item>b</item>
            </consequent>
      </rule>
      <rule support="0.4" confidence="1.0">
            <antecedent>
                  <item>b</item>
                  <item>c</item>
            </antecedent>
```

```
            <consequent>
                  <item>d</item>
            </consequent>
      </rule>
</rules>
```

En general, para poder aplicar el algoritmo Apriori lo que se tiene que hacer una transformación previa al documento XML, algunos generando otro archivo en términos de transacciones e items [35], en otros casos a formatos como CSV [24], pero en pocas ocasiones se mantiene el documento XML original para la obtención de las reglas de asociación a partir de éste.

También se ha propuesto modificación al algoritmo Apriori donde se busca minar patrones frecuentes cortos [28], los cuales buscan limitar el tamaño de los itemsets generados, ya que se pueden obtener itemsets muy grandes, por lo que en este artículo se propone el AprioriSP (Apriori Short Pattern), que es idéntico al algoritmo original, solo que con una entrada adicional que es el tamaño máximo para los itemsets. El inconveniente de este artículo es que vuelve a tomar el archivo basado en transacciones e items.

Ejemplo 2.6: Documento XML usado en los artículos.

```
<transactions>
      <transaction ID=1>
            <items>
                  <item>1</item>
                  <item>25</item>
                  <item>12</item>
            </items>
      </transaction>
      <transaction ID=2>
            <items>
                  <item>7</item>
            </items>
      </transaction>
      <transaction ID=3>
            <items>
                  <item>27</item>
                  <item>18</item>
            </items>
      </transaction>
</transactions>
```

La ventaja que se presenta al usar el Ejemplo 2.6 es que se puede aplicar directamente el algoritmo Apriori, ya que todo esta en términos de transacciones e items, como lo específica el algoritmo.

Pero la más grande desventaja que presenta este tipo de documentos, es que se pierde totalmente la estructura del XML, por lo que el beneficio de hacer minería en este tipo de documentos se pierde. Ya que una de las virtudes de XML es que la estructura da contexto del documento.

Un ejemplo donde se busca preservar la estructura original de los documentos XML, es el de las plantillas, o el de reglas basadas en árboles, y que como se plantea, buscan aprovechar y poder minar tanto el contenido como la estructura.

Capítulo 3

Algoritmo Propuesto

3.1. Introducción

En los capítulos pasados, se han mostrado las implementaciones actuales para obtener reglas de asociación a partir de la generación de itemsets frecuentes, y también se ha presentado un panorama de minería de datos y XML.

El problema radica en cómo aplicar ambos conceptos de manera eficiente y útil. Observando el estado actual sobre XML y reglas de asociación, en su mayoría se usa el algoritmo Apriori, sin embargo, se tienen que reprocesar los documentos a fin de transformarlos a un formato transaccional. Lo anterior derivado de que el algoritmo Apriori funciona sobre las transacciones. Lo único negativo de esto, es que se pierde información estructural.

El formato XML al ser autodescriptivo, por medio de sus etiquetas, permite conocer más sobre el documento que se está observando, ya que es definido en su totalidad por éstas. Además de su conocida jerarquía en elementos anidados.

Si se perdiera la información de las etiquetas, los documentos carecerían de descripción y serían simplemente datos, por lo que se perderían las ventajas sobre las bases de datos relacionales. Entonces se necesita poder hacer la minería de datos directamente sobre los documentos XML.

3.2. Análisis

El problema es aplicar el algoritmo Apriori, en documentos XML sin que se pierda la información estructural.

El algoritmo Apriori necesita una entrada que es una base de datos de transacciones, por lo que se tiene que transformar los documentos XML hacia este formato, una vez transformados poder aplicar el algoritmo, encontrar las reglas de asociación y evaluar los resultados.

Esto ocasiona dos problemas adicionales, el primero es la transformación del documento, y el segundo, que después de aplicar el algoritmo, se puedan reconstruir las reglas de asociación en un documento XML, con sus respectivos parámetros de salida (soporte y confianza). Esto con el objetivo de mantener el formato original.

En este trabajo se propone una forma de solucionar estos problemas, a fin de poder aplicar el algoritmo Apriori de manera directa sobre los documentos XML, sin que tengan que sufrir ninguna alteración en su funcionamiento.

En el primer problema lo que se necesita es abstraer la información útil, sin descartar la que en el algoritmo pueda resultar innecesaria, pero en estructura necesaria. Es decir, no es necesario minar todo el árbol del XML, dado que los elementos que mayor información aportan son los nodos Hoja. Ya que estos al ser elementos simples, como se vio anteriormente, tienen valores.

También se vio que se puede realizar minería sobre la estructura o sobre el contenido. En este caso se necesita de ambos, ya que si solo se hace minería en el contenido, como se hace en la mayoría de los trabajos publicados, [22, 28, 30, 35, 39], lo que se obtiene son reglas de asociación pero sin contexto debido a que estos solo toman valores dentro de los nodos sin tomar en cuenta todo el nodo en su totalidad. Esta es una salida correcta, ya que se generan las reglas de asociación, pero en cambio se pierde la información adicional que proporciona este formato.

Una manera de realizar esta selección de elementos, es tomar todos los elementos que sean simples, en tuplas de la forma (Etiqueta, Valor). Se supone que los documentos usados tienen un esquema XSD.

Tomando el esquema del documento, se debe de hacer una distinción entre los elementos con la misma etiqueta. Un ejemplo es la etiqueta `<nombre>`, ya

que ésta puede encontrarse fácilmente como un elemento hoja dentro de ciertos nodos. Dado que no se puede obtener el origen de los elementos solo a partir de la etiqueta, se debe añadir cierta información adicional. Para solucionar este problema de ambigüedad, se maneja una sustitución de las etiquetas, conforme se va recorriendo la estructura del árbol, se hace una sustitución con un identificador. El identificador sera almacenado dentro de una estructura de datos, ya sea un diccionario o una lista.

Después de la sustitución de etiquetas, se obtiene un documento XML equivalente y libre de ambigüedades, lo que permite continuar con la selección de tuplas. Como se observa en la Figura 3.1, se pueden tener elementos simples, pero aun existir algunos que no será utilizados. Adicionalmente al documento XML, se puede recibir una lista de los elementos hoja a ser tomados en cuenta, permitiendo así que se pueda escoger entre considerar todos los elementos o solo tomar la selección que se establezca.

Figura 3.1: Estructura de un XML en forma de transacciones

Este conjunto se le denotará como H. El siguiente paso es que una vez definidos los items, se deben definir cuáles son las transacciones. En general, puede haber distintas maneras de establecer este punto, uno de ellos es que cada documento XML sea en su totalidad una transacción. En este primer caso no debe existir ningún problema, ya que están definidas por sí mismas. En el caso de que sea un solo documento XML, debe de haber una etiqueta que marque esta separación entre transacciones.

Si se toma de ejemplo nuevamente la Figura 3.1, el elemento `<staff_member>` es el que marca el inicio y fin de cada transacción, pero existe un problema con

este tipo de elementos, no hay manera de definirlos de manera general. Una alternativa para poder identificarlos es con un atributo que funcione como bandera, la manera más común de representar este atributo es con el nombre de id.

El id generalmente hace referencia a la llave primaria de las tuplas en el modelo relacional. Se puede ver a esta tupla como un equivalente a los elementos dentro de la transacción del XML, pero la clara distinción entre ambos, es que la tupla siempre tendrá la misma cantidad de columnas tengan o no valor, y en el documento XML, la cantidad de elementos que puede haber dentro de la transacción podría no ser siempre la misma.

Esto causa una dificultad, ya que como es posible omitir los elementos, también es posible omitir los atributos y se regresaría al punto inicial. En este caso a pesar de tratar de obtener una solución que sea general se necesita de otra entrada, que sería el nombre de la etiqueta que define a las transacciones, la cual es necesaria en el caso de que se tenga un solo documento XML. A esta etiqueta se le denotara como e.

Teniendo como entradas al documento XML X, al conjunto de nodos H y a la etiqueta que marca el inicio de las transacciones e, se puede formular una solución que será explicada en la siguiente sección. Básicamente, es generar una base de datos transaccional la cual será una entrada válida para el algoritmo Apriori que se conoce. Esto sin olvidar las entradas propias de Apriori, las cuales son la confianza y el soporte mínimos para las reglas de asociación.

En un inicio se planteó que hay que reconstruir el árbol a partir de la salida, esto para mantener todo en el formato XML. El algoritmo Apriori devolverá como su salida las reglas de asociación, pero en un formato de texto plano, es decir, $X \Rightarrow Y$. Por lo que hay que convertir la salida en un documento XML, el cual contará con las etiquetas necesarias para darle estructura.

Una estructura posible para una regla es la mostrada en el Ejemplo 3.1, en ésta se muestran ambas partes de la regla, con sus respectivas métricas. Se puede observar que ya cuenta con toda la estructura necesaria. Solo falta recuperar las etiquetas simples y todas las etiquetas complejas que contengan a los nodos hoja resultantes dentro de X y de igual manera con Y. Para esto se requiere de la estructura donde se almacenaron todas las etiquetas, así como una estructura correspondiente de como llegar al nodo hoja.

Ejemplo 3.1: Ejemplo de Regla de Asociación en Formato XML

```
<?xml version="1.0" encoding="utf-8"?>
```

```
<Reglas>
      <Regla>
            <Antecedente>
            ...
            </Antecedente>
            <Consecuente>
            ...
            </Consecuente>
            <Soporte> 0.6 </Soporte>
            <Confianza> 0.9 </Confianza>
      </Regla>
      <Regla>
      ...
      </Regla>
<Reglas>
```

3.3. Solución

Una vez analizada la problemática, se determinó que se deben de hacer ciertas transformaciones al documento XML antes de aplicarle el algoritmo Apriori, así como una vez aplicado el mismo y que se explicaran a continuación.

3.3.1. Preprocesamiento

En el análisis, se determinó que se necesitan tres entradas: un documento XML X, un conjunto H con todos los elementos hoja a considerar y un elemento e, el cual determina el nombre del nodo que contendrá cada transacción.

Un documento XML X se puede ver como un árbol de etiquetas, que comienzan desde un nodo raíz. Se puede ver el algoritmo en función a eventos, como lo hace el parser SAX, en este caso, se toma etiqueta por etiqueta, si se encuentra una etiqueta que sea igual a e se crea una transacción.

A partir de este punto, todos los elementos simples s que están contenidos en H, se introducirán como tuplas (Etiqueta, Valor) dentro de la transacción actual, hasta que se encuentre la etiqueta de cierre de e. Al encontrar esta se

guardará la transacción dentro de una lista de transacciones y así sucesivamente hasta terminar de recorrer el documento.

Algoritmo 2 Algoritmo Pre-Procesamiento

```
1: function ConvierteXML(X, H, e)
2:     listTransacciones ← []
3:     trans ← null
4:     for all Etiquetas t ∈ X do
5:         if t = tagInicial(e) then
6:             trans ← [] t = tagFinal(e)
7:             if transisNotEmpty() then
8:                 listTransaccion.add(trans)
9:             end if
10:        end if
11:        if t is nodoHoja and t ⊂ H then
12:            item ← (t.nombreEtiqueta,t.valor)
13:            trans.add(item)
14:        end if
15:    end for
16:    if trans ≠ null then
17:        listTransaccion.add(trans)
18:    end if
19:    return listTransacciones
20: end function
```

Este preprocesamiento toma $\mathcal{O}(n)$ ya que $|X| = n$, es decir, se tienen n etiquetas dentro del documento XML, y aunque se toman en cuenta solamente los nodos hoja, se tienen que recorrer todas las etiquetas que hay dentro el mismo.

Antes de poder aplicar el Algoritmo 2, se debe de verificar que no haya más de una etiqueta hoja con el mismo nombre. Si cada etiqueta no tiene su nombre repetido dentro de otra, es posible aplicar este algoritmo sin tener que sustituir las etiquetas.

En el caso contrario, es decir donde existen al menos dos etiquetas Hoja con el mismo nombre, se debe de realizar una sustitución de las etiquetas, por lo que se deben renombrar las etiquetas del esquema. Para esto se toma el esquema asociado al documento, y en cada una de las etiquetas que hacen referencia a un elemento, se guarda su nombre en un diccionario y se lleva un conteo de los elementos.

Si es el primer elemento que se agrega, se guarda dentro del diccionario con el valor de 1, y el nombre del elemento no cambia. En el caso donde ya exista el elemento dentro del diccionario, a la etiqueta se le agrega al final el valor actual que tenga este elemento dentro del diccionario y se incrementará en una unidad el valor de este elemento dentro del mismo.

Algoritmo 3 Algoritmo Sustitución

```
1: function ConvierteXSD(XSD)
2:     diccionario ← {}
3:     for all Etiquetas t ∈ XSD do
4:         if t is element then
5:             if diccionario.contains(t) then
6:                 t.name = t.name+diccionario.getValue(t.name)
7:                 diccionario.put(t.name,diccionario.getValue(t.name)+1)
8:             end if
9:             if ! diccionario.contains(t) then
10:                diccionario.put(t.name,1)
11:            end if
12:        end if
13:    end for
14:    return (diccionario, XSD)
15: end function
```

El Algoritmo 3 se puede ver aplicado en los ejemplos 3.2 y 3.3, en estos se tienen repetidas las etiquetas **Name** y **Age** tres veces. Al aplicar el Algoritmo 3, ya no existen etiquetas repetidas y se observa como la primera etiqueta no sufrió ningún cambio en su nombre. Sin embargo, a las otras dos se les agregó el sufijo 1 y 2, respectivamente.

Ejemplo 3.2: Ejemplo de Sustitución(Original)

```
<?xml version="1.0" encoding="UTF-8"?>
  <movie>
    <Title>The Dark Knight</Title>
    <Writers>
      <Writer>
        <Name>Jonathan Nolan</Name>
        <Age>45</Age>
      </Writer>
    </Writers>
    <Director>
      <Name>Christopher Nolan</Name>
```

```
    <Age>50</Age>
   </Director>
   <Released>18 Jul 2008</Released>
   <Actors>
    <Actor>
     <Name>Christian Bale</Name>
     <Age>43</Age>
    </Actor>
   </Actors>
  </movie>
```

Ejemplo 3.3: Ejemplo de Sustitución(Cambio)

```
<?xml version="1.0" encoding="UTF-8"?>
  <movie>
   <Title>The Dark Knight</Title>
   <Writers>
    <Writer>
     <Name>Jonathan Nolan</Name>
     <Age>45</Age>
    </Writer>
   </Writers>
   <Director>
    <Name1>Christopher Nolan</Name1>
    <Age1>50</Age1>
   </Director>
   <Released>18 Jul 2008</Released>
   <Actors>
    <Actor>
     <Name2>Christian Bale</Name2>
     <Age2>43</Age2>
    </Actor>
   </Actors>
  </movie>
```

Este renombrado toma $\mathcal{O}(m)$, con $m = |XSD|$, ya que se necesita recorrer todo el esquema. Para transformar el XML actual se necesita de XSLT[4], y hacer una transformación identidad, con lo que se obtendrá el documento con sus etiquetas renombradas. Cuando exista el renombrado, si el conjunto H tiene una etiqueta la cual se encuentra repetida dentro del esquema en un nodo Hoja, todas las que tengan ese nombre serán tomadas en cuenta, ya que no habría manera de distinguirlas.

3.3.2. Algoritmo Apriori

Por la manera en que se realizó el preprocesamiento, se obtiene una lista de transacciones que puede ser directamente aplicable al algoritmo Apriori. Por lo que la ejecución de éste se realiza de manera transparente; es decir, se toman las transacciones, soporte y confianza. Así que su respuesta en este caso son las reglas de asociación resultantes en forma de tuplas (Etiqueta, Valor), en vez de los items que se usan normalmente en este algoritmo.

3.3.3. Posprocesamiento

Una vez generadas las reglas de asociación, únicamente falta recuperar el árbol de cada tupla (Etiqueta, Valor) obtenida dentro de la regla. Para esto se puede usar XPath[5], con la expresión "//Etiqueta", la cual recupera los elementos de nombre Etiqueta. A partir de éste es posible reconstruir el árbol sólo obteniendo el nombre de los nodos Padres con el Algoritmo 4.

Algoritmo 4 Algoritmo Posprocesamiento

```
1: function ObtieneArbol(XML, Etiqueta, Valor)
2:     xpath ="//Etiqueta"
3:     arbol = Valor
4:     nodo = xpath.evaluate(XML).first()
5:     while nodo.getParentNode()!=null do
6:         arbol = < nodo.getNodeName() > arbol </ nodo.getNodeName()>
7:         nodo = nodo.getParentNode()
8:     end while
9:     return (arbol)
10: end function
```

3.4. Aplicación

Una aplicación para este algoritmo es el caso de los servicios web que devuelven una respuesta en XML. Sin embargo, estos funcionan a base de peticiones, y se

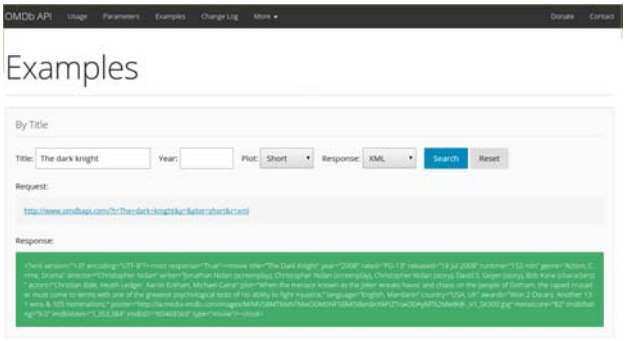

Figura 3.2: The Open Movie Database es una API gratuita que devuelve la información de una película en formato JSON o XML

tienen que hacer una petición a la vez, en lugar de descargar una base de archivos con una sola petición.

Un sitio que ofrece esta posibilidad es OMDb (Open Movie Database Figura 3.2), este sitio ofrece consultas de películas, siendo muy parecido a IMDb (Internet Movie Database), sólo que este último está enfocado a consultas a través del sitio web. Mientras que OMDb únicamente contesta a peticiones, donde simplemente se introduce el nombre de la película que se busca, así como el formato en el que se desea la respuesta.

Ejemplo 3.4: Petición de la película The Dark Knight

```
<?xml version="1.0" encoding="UTF-8"?>
<root response="True">
<movie title="The Dark Knight"
  year="2008"
  rated="PG-13"
  released="18 Jul 2008"
  runtime="152 min"
  genre="Action, Crime, Drama"
  director="Christopher Nolan"
  writer="Jonathan Nolan, Christopher Nolan..."
  actors="Christian Bale, Heath Ledger, Aaron Eckhart..."
  plot="When the menace known as the Joker wreaks havoc ..."
  language="English, Mandarin"
  country="USA, UK"
  awards="Won 2 Oscars. Another 131 wins & 105 nominations."
  metascore="82"
  imdbRating="9.0"
  imdbVotes="1,353,384"
```

```
  imdbID="tt0468569"
  type="movie"/>
</root>
```

Esta página provee de una gran cantidad de información con respuestas en XML, pero la respuesta a la petición es sólo un XML con una sola etiqueta y muchos atributos como se puede ver en el Ejemplo 3.4. Con lo que habría que convertir esta respuesta a un formato más adecuado a nuestras necesidades.

Se puede observar que ya existen ciertas estructuras que sólo se tienen que pasar a etiquetas, por ejemplo, en el caso de Actors, un equivalente sería tener una etiqueta Padre que sea Actors y etiquetas hijas, que serían en dado caso Actor y Actress. De igual manera con Writers, Country y Language.

Ejemplo 3.5: Petición de la película The Dark Knight transformada

```
<?xml version="1.0" encoding="UTF-8"?>
  <movie>
    <Rated>PG-13</Rated>
    <Languages>
      <Language>English</Language>
      <Language>Mandarin</Language>
    </Languages>
    <Title>The Dark Knight</Title>
    <Countries>
      <Country>USA</Country>
      <Country>UK</Country>
    </Countries>
    <Writers>
      <Writer>Jonathan Nolan</Writer>
      <Writer>Christopher Nolan</Writer>
      <Writer>David S. Goyer</Writer>
      <Writer>Bob Kane</Writer>
    </Writers>
    <imdbRating>9.0</imdbRating>
    <Director>Christopher Nolan</Director>
    <Released>18 Jul 2008</Released>
    <Actors>
      <Actor>Christian Bale</Actor>
      <Actor>Heath Ledger</Actor>
      <Actor>Aaron Eckhart</Actor>
      <Actor>Michael Caine</Actor>
    </Actors>
    <Year>2008</Year>
    <Genres>
      <Genre>Action</Genre>
      <Genre>Crime</Genre>
      <Genre>Drama</Genre>
    </Genres>
    <Awards>
      <Oscar>
        <Status>Winner</Status>
      </Oscar>
    </Awards>
    <Runtime>152 min</Runtime>
    <imdbVotes>1,353,384</imdbVotes>
  </movie>
```

Una transformación para los documentos de esta página es la que se vio en el Ejemplo 3.5, con la cual ya se obtiene un documento XML con una estructura adecuada para poder usar los algoritmos anteriores, además de que provee cierto contexto y jerarquía. En este caso se tendrían que hacer una gran cantidad de peticiones, junto a sus respectivas transformaciones.

Para el caso de realizar peticiones se usó la lista de películas de Wikipedia [13], a partir de esta se procedió a realizar las peticiones correspondientes, almacenar los archivos y transformarlos en un XML equivalente. Con esta colección de archivos se realizó un programa en Java que implementará los algoritmos de la Sección 3.3.

Usando un soporte del 60 % y una confianza del 70 % se generaron las reglas de asociación mostradas en el Ejemplo 3.6.

En este caso se obtuvo que con un 98 % de confianza, las películas hechas en Estados Unidos, son en inglés. Pero si las películas en inglés, su país de origen es Estados Unidos, tiene un 82 % de confianza. Lo cual tiene sentido ya que hay varios países de habla inglesa que también producen películas, tales como el Reino Unido o la India.

Se podrían obtener reglas más interesantes filtrando los datos para refinar la búsqueda, pero el objetivo principal era el de usar la minería con nuestros documentos XML.

Esta aplicación involucró un paso extra de preprocesamiento, dado que hubo que cambiar el XML original por uno que tuviera la estructura adecuada. Pero en general si cumple con tener una estructura jerárquica y son válidos o bien formados es suficiente para aplicar el algoritmo directamente.

Ejemplo 3.6: Algunas reglas de asociación resultantes de una lista de películas nominadas y ganadoras del Oscar

```
<?xml version="1.0" encoding="utf-8"?>
<Reglas>
      <Regla>
            <Antecedente>
                  <movie>
                     <Rated>R</Rated>
                  </movie>
            </Antecedente>
            <Consecuente>
                  <movie>
                     <Awards>
                       <Oscar>
                         <Status>Nominated</Status>
                       </Oscar>
                     </Awards>
                   </movie>
            </Consecuente>
            <Soporte> 0.18676 </Soporte>
            <Confianza> 0.65464 </Confianza>
      </Regla>
      <Regla>
            <Antecedente>
                  <movie>
                     <Genres>
                       <Genre>Adventure</Genre>
                     </Genres>
                  </movie>
            </Antecedente>
            <Consecuente>
                  <movie>
                     <Awards>
                       <Oscar>
                         <Status>Winner</Status>
                       </Oscar>
                     </Awards>
                   </movie>
            </Consecuente>
            <Soporte> 0.10343 </Soporte>
            <Confianza> 0.64724 </Confianza>
      </Regla>
```

```
<Regla>
      <Antecedente>
            <movie>
                  <Rated>R</Rated>
               <Genres>
                 <Genre>Drama</Genre>
               </Genres>
            </movie>
      </Antecedente>
      <Consecuente>
            <movie>
               <Awards>
                 <Oscar>
                   <Status>Nominated</Status>
                 </Oscar>
               </Awards>
             </movie>
      </Consecuente>
      <Soporte> 0.15343 </Soporte>
      <Confianza> 0.53780 </Confianza>
</Regla>
<Regla>
      <Antecedente>
            <movie>
                  <Rated>R</Rated>
               <Countries>
                 <Country>USA</Country>
               </Countries>
               <Languages>
                 <Language>English</Language>
               </Languages>
               <Awards>
                 <Oscar>
                   <Status>Nominated</Status>
                 </Oscar>
               </Awards>
            </movie>
      </Antecedente>
      <Consecuente>
            <movie>
               <Genres>
                 <Genre>Drama</Genre>
               </Genres>
```

```
                    </movie>
                </Consecuente>
                <Soporte> 0.11667 </Soporte>
                <Confianza> 0.79866 </Confianza>
        </Regla>
</Reglas>
```

3.5. Resultados

Tomando de ejemplo el caso práctico, se realizaron pruebas con distintos volúmenes de transacciones. En este caso el Apriori recibió como parámetros de entrada un soporte del 60 % y una confianza del 70 %, además el número de items dentro de las transacciones pueden ir desde 13 hasta 50.

Observando los resultados se puede notar que el algoritmo escala de manera lineal, mayormente porque escalando el algoritmo Apriori también se comporta lineal como se observa en la Figura 3.5.

Además, cómo se vio en las secciones anteriores, tanto la generación de las transacciones como el renombramiento, sólo consisten en recorrer el documento XML o el esquema una vez, por lo que su complejidad es $\mathcal{O}(m) + \mathcal{O}(n) = \mathcal{O}(\max\{m, n\}) = \mathcal{O}(m)$. Donde m y n son el número de etiquetas del documento y el esquema respectivamente.

Como se puede notar también, el tiempo de ejecución aumenta de manera gradual dependiendo del número de transacciones. Esto puede ser porque el número de itemsets frecuentes se incrementa debido al incremento de las transacciones, además de que encontrar los candidatos dentro de las transacciones toma más tiempo, [16].

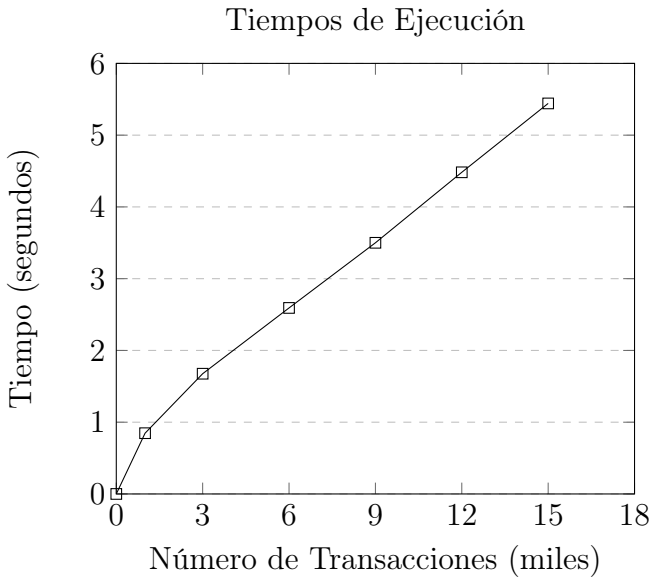

Figura 3.3: Resultados de Tiempos Relativos de Ejecución

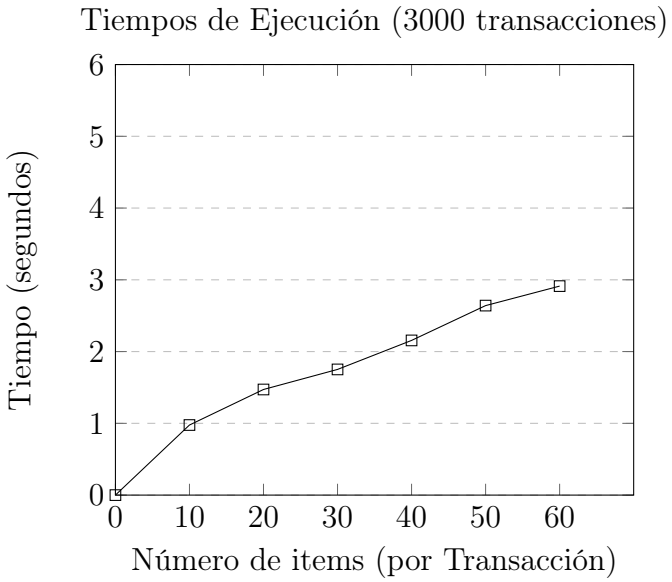

Figura 3.4: Resultados de Tiempos Relativos de Ejecución (número de Items)

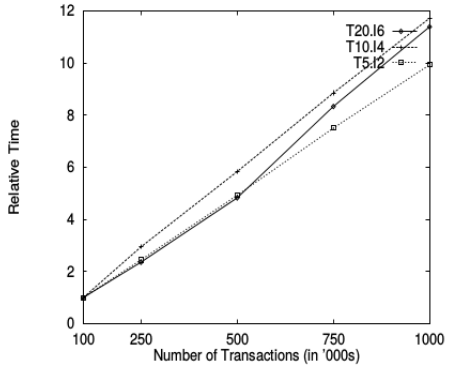

Figura 3.5: Escalabilidad del Apriori

También se realizaron pruebas en proporción al número de items dentro de cada transacción, esto dentro de un intervalo de 10 hasta 60, y considerando 3000 transacciones, la cual es ilustada en la gráfica presentada en la Figura 3.4. Aquí se observa que en comparación con la ejecución inicial en 3000 transacciones, difiere en cuanto a su resultado original que se presenta en la Figura 3.3, esto debido a que en esta última ejecución; el número de item era fijo en cada ejecución. En cambio, en la gráfica inicial, el número de items era variable, ya que en promedio era de 20 items por transacción.

En general, es más adecuada la Figura 3.3, dado que en un documento XML, sus elementos son variables y esa es su principal ventaja. Ya que se le puede agregar nuevos elementos y aun así este seguir siendo operable.

Tomando en cuenta los resultados, se puede observar que en general no afecta el rendimiento del algoritmo Apriori original, cuyo rendimiento se presenta en la Figura 3.5, sino que se mantiene dado el bajo costo de realizar ambos algoritmos. Por lo que el beneficio es muy grande, ya que de esta manera es posible el minar documentos XML de manera directa y genérica.

Adicionalmente se comparó el rendimiento entre tener todos los archivos XML por separado, o teniendo un sólo archivo que contuviera a todas las transacciones. En este caso se obtuvo que el rendimiento teniendo todas las transacciones en un solo archivo es notablemente mayor a que ir leyendo uno por uno. Esto es debido a que el parseo del documento XML es muy lento, y es mucho más rápido leer un solo archivo aunque su tamaño sea muy grande, a leer el equivalente a ese archivo pero divido en 3000 archivos. Por lo que se omite esa prueba, dado que se pierde el contexto del algoritmo a evaluar y lo que realmente se evalúa es la lectura de los archivos.

Conclusiones y Trabajos Futuros

Se desarrolló un algoritmo para poder encontrar reglas de asociación en documentos XML a través del algoritmo Apriori sin transformar los documentos originales, en el cual es posible la generación del árbol a partir de las reglas de asociación obtenidas.

También se mostró un caso práctico donde es aplicable este algoritmo, aunque se tuvo que hacer una transformación previa a esos documentos para que pudiera ser aplicado el algoritmo de generación de transacciones y posteriormente al algoritmo Apriori.

Esto debido a que gran parte de los servicio web proveen respuestas tanto en XML como en JSON[12], sin embargo, hay sitios que simplemente intentan transformar estas respuestas en su equivalente de XML pero relegando todo al uso de atributos y no de etiquetas. Como se puede ver en los ejemplos A.1 y A.2, generalmente ambos formatos tienen una equivalencia.

Esto hace que exista una especie de competencia entre XML y JSON, siendo la principal ventaja del segundo que es más ligero en cuanto al contenido, ya que XML contiene muchas etiquetas.

Una de las dificultades que hubo, fue encontrar una colección de documentos XML, ya que las existentes en ocasiones sólo eran muestras de bases de datos relacionales intentando ser exportadas a XML y no una verdadera recopilación de XML. Por lo que el mayor problema es que se hace un mal uso del XML, ya que se desaprovecha su estructura de árbol y se define toda la información en los atributos. O que en ocasiones no se hace una verdadera recopilación de los XML,

donde si podrían aprovechar totalmente, como es el caso de los servicios web.

Sin embargo, si se presentan archivos con el formato requerido, la ventaja es muy grande, dado que ya es posible hacer minería de datos sobre estos. Sólo son necesario los documentos XML, que el usuario proporcione la etiqueta que defina a las transacciones, un soporte y la confianza para las reglas. Con esto se generarían las reglas resultantes en un formato XML nuevamente para hacer una interpretación de los resultados.

Un posible trabajo para el futuro es integrar JSON con reglas de asociación, se decidió usar XML porque es el formato más popular de intercambio de información. Además de que XML tiene otros usos, ya sea como archivos de configuración o el almacenamiento de metadatos.

El uso del procesamiento paralelo sería una opción adicional para trabajar en el futuro, dado que cada transacción es independiente se podrían destinar n hilos de ejecución, a que cada uno se encargue de transformar las transacciones de manera simultánea, incrementando así la velocidad de la transformación.

Otra posibilidad en el futuro es realizar una implementación similar a la actual pero usando el algoritmo FP-Growth, o extenderlo a otro tipo de minería de datos, ya sea Clustering o Clasificación.

Apéndice A

Documentos XML y JSON

Se muestra una equivalencia entre un documento JSON y un documento XML.

Ejemplo A.1: Ejemplo JSON

```
{"widget": {
   "debug": "on",
   "window": {
      "title": "Sample Konfabulator Widget",
      "name": "main_window",
      "width": 500,
      "height": 500
   },
   "image": {
      "src": "Images/Sun.png",
      "name": "sun1",
      "hOffset": 250,
      "vOffset": 250,
      "alignment": "center"
   },
   "text": {
      "data": "Click Here",
      "size": 36,
      "style": "bold",
      "name": "text1",
      "hOffset": 250,
      "vOffset": 100,
      "alignment": "center",
      "onMouseUp": "sun1.opacity = (sun1.opacity / 100) * 90;"
```

```
    }
}}
```

Ejemplo A.2: Ejemplo XML

```
<widget>
    <debug>on</debug>
    <window title="Sample Konfabulator Widget">
        <name>main_window</name>
        <width>500</width>
        <height>500</height>
    </window>
    <image src="Images/Sun.png" name="sun1">
        <hOffset>250</hOffset>
        <vOffset>250</vOffset>
        <alignment>center</alignment>
    </image>
    <text data="Click Here" size="36" style="bold">
        <name>text1</name>
        <hOffset>250</hOffset>
        <vOffset>100</vOffset>
        <alignment>center</alignment>
        <onMouseUp>
            sun1.opacity = (sun1.opacity / 100) * 90;
        </onMouseUp>
    </text>
</widget>
```

Bibliografía

[1] (1989). Workshop on knowledge discovery in databases. http://www.kdnuggets.com/meetings/kdd89/. [Internet; consultado 27-enero-2015].

[2] (2002). Xml pointer language (xpointer). https://www.w3.org/TR/xptr/. [Internet; consultado 14-enero-2015].

[3] (2005). Document object model (dom). http://www.w3.org/DOM/. [Internet; consultado 14-enero-2015].

[4] (2007). Xsl transformations (xslt). http://www.w3.org/TR/xslt20/. [Internet; consultado 14-enero-2015].

[5] (2011). Xml path language (xpath) 2.0. http://www.w3.org/TR/xpath20/. [Internet; consultado 14-enero-2015].

[6] (2012). Document type definition (dtd). http://www.w3schools.com/dtd/. [Internet; consultado 14-enero-2015].

[7] (2012). Introduction to xml. http://www.ibm.com/developerworks/xml/tutorials/xmlintro/xmlintro.html. [Internet; consultado 14-enero-2015].

[8] (2012). Xml schema definition language (xsd). http://www.w3.org/TR/xmlschema11-1/. [Internet; consultado 14-enero-2015].

[9] (2014). Data mining definition and origins. http://www.dataminingarticles.com/info/data-mining-introduction/. [Internet; consultado 27-enero-2015].

[10] (2014). Extensible markup language (xml). http://www.w3.org/XML/. [Internet; consultado 14-enero-2015].

[11] (2015). Data mining concepts. http://docs.oracle.com/cd/B28359_01/datamine.111/b28129/toc.htm. [Internet; consultado 31-marzo-2015].

[12] (2015). Json (javascript object notation). http://json.org/. [Internet; consultado 31-marzo-2015].

[13] (2015). List of films (wikipedia). http://en.wikipedia.org/wiki/Lists_of_films#Alphabetical_indices. [Internet; consultado 22-marzo-2015].

[14] Abraham Silberschatz, H. F. K. (2011). *Database System Concepts*. McGraw-Hill.

[15] Agrawal, R., Imieliński, T., and Swami, A. (1993). Mining association rules between sets of items in large databases. *SIGMOD Rec.*, 22(2):207–216.

[16] Agrawal, R. and Srikant, R. (1994). Fast algorithms for mining association rules in large databases. In *Proceedings of the 20th International Conference on Very Large Data Bases*, VLDB '94, pages 487–499, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

[17] Ali Mohammadzadeh, R., Soltan, S., and Rahgozar, M. (2006). Template guided association rule mining from xml documents. In *Proceedings of the 15th International Conference on World Wide Web*, WWW '06, pages 963–964, New York, NY, USA. ACM.

[18] Azevedo, A. and Santos, M. F. (2008). Kdd, semma and crisp-dm: a parallel overview. In Abraham, A., editor, *IADIS European Conf. Data Mining*, pages 182–185. IADIS.

[19] Berkhin, P. (2002). Survey of clustering data mining techniques. Technical report.

[20] Brown, M. (2012). Data mining techniques. *developerWorks*.

[21] Fayyad, U., Piatetsky-shapiro, G., and Smyth, P. (1996). From data mining to knowledge discovery in databases. *AI Magazine*, 17:37–54.

[22] Feng, L. and Dillon, T. (2005). Mining interesting xml-enabled association rules with templates. In Goethals, B. and Siebes, A., editors, *Knowledge Discovery in Inductive Databases*, volume 3377 of *Lecture Notes in Computer Science*, pages 66–88. Springer Berlin Heidelberg.

[23] Feng, L., Dillon, T., Weigand, H., and Chang, E. (2003). An xml-enabled association rule framework. In Mařík, V., Retschitzegger, W., and Štěpánková, O., editors, *Database and Expert Systems Applications*, volume 2736 of *Lecture Notes in Computer Science*, pages 88–97. Springer Berlin Heidelberg.

[24] G. Kaur, N. A. (2012). Exploiting hierarchal structure of xml data using association rule analysis. *International Journal of Machine Learning and Computing*, 2(3):274–277.

[25] Han, J., Pei, J., and Yin, Y. (2000). Mining frequent patterns without candidate generation. In *ACM SIGMOD Record*, volume 29, pages 1–12. ACM.

[26] Jacky W.W. Wan, G. D. (2004). Mining association rules from xml data using xquery. *Proc. Australasian Workshop on Data Mining and Web Intelligence (DMWI2004)*, pages 305–317.

[27] Jiawei Han, Micheline Kamber, J. P. (2012). *Data Mining: Concepts and Techniques*. Morgan Kaufmann.

[28] L. Yijun, H. Sheng, Y. F. (2012). A comparison of association rule mining methods for xml data. *International Journal of Advancements in Computing Technology(IJACT)*.

[29] Liu, B. (2007). *Web Data Mining Exploring Hyperlinks, Contents, and Usage Data*. Springer.

[30] M. Mazuran, E. Quintarelli, L. T. (2009). Mining tree-based association rules from xml documents. *Proceedings of the Seventeenth Italian Symposium on Advanced Database Systems*.

[31] Mohammed Zaki, W. M. (2014). *Data Mining and Analysis: Fundamental Concepts and Algorithms*. Cambridge University Press.

[32] Mrs. Bharati, M. R. (2010). Data mining techniques and applications.

[33] Nayak, R. (2008). Xml data mining: Process and applications.

[34] NORTO, M. J. (1999). Knowledge discovery in databases.

[35] Q. Ding, G. S. (2006). Association rule mining from xml data. *International Conference on Data Mining,*, pages 144–150.

[36] Ramez Elmasri, S. B. N. (2011). *Fundamentals of Database Systems*. Addison-Wesley.

[37] Raso, L. (2012). Xml data mining. *developerWorks*.

[38] S. Zhang, J. Zhang, H. L. W. W. (2005). Xar-miner: Efficient association rules mining for xml data.

[39] Vikhe, P. and Gunjal, B. (2013). Extracting tree based association rules from xml document. *International Journal*, 3(6).

[40] Zaki, M. J. and Wong, L. (2003). Data mining techniques.