Mate-tree: un algoritmo para la tarea de clasificación basado en operadores algebraicos y primitivas SQL*[1]

[Mate-tree: an Algorithm for classification task based on algebraic operators and SQL primitives]

Ricardo TIMARÁN PEREIRA[2]

Recibo: 20.02.2012 - Aprobación: 20.05.2012

Resumen

La clasificación basada en árboles de decisión es el modelo más utilizado y popular por su simplicidad y facilidad para su entendimiento. El cálculo del valor de la métrica que permite seleccionar, en cada nodo, el atributo que tenga una mayor potencia para clasificar sobre el conjunto de valores del atributo clase, es el proceso más costoso del algoritmo utilizado. Para calcular esta métrica, no se necesitan los datos, sino las estadísticas acerca del número de registros en los cuales se combinan los atributos condición con el atributo clase. Entre los algoritmos de clasificación por árboles de decisión se cuentan ID-3, C4.5, SPRINT y SLIQ. Sin embargo, ninguno de estos algoritmos se basan en operadores algebraicos relacionales y se implementa con primitivas SQL. En este artículo se presenta Mate-tree, un algoritmo para la

* Modelo para citación de este Reporte de Caso:
TIMARÁN PEREIRA, Ricardo (2012). Mate-tree: un algoritmo para la tarea de clasificación basado en Operadores algebraicos y Primitivas SQL. En: Ventana Informática. No. 26 (ene. – jun., 2012). Manizales (Colombia): Facultad de Ciencias e Ingeniería, Universidad de Manizales. p. 61-76. ISSN: 0123-9678

1 Artículo proveniente del proyecto de investigación *PostgreSQLKDD- Un Sistema de Descubrimiento de Conocimiento fuertemente acoplado con el SGBD PostgreSQL*, ejecutado en el periodo *febrero 2006 – febrero 2007*, e inscrito en el grupo de investigación *GRIAS* de la *Universidad de Nariño*.

2 Doctor en Ingeniería, Master of Science en Ingeniería, Especialista en Multimedia e Ingeniero de Sistemas y Computación. Director grupo de investigación GRIAS, Profesor Asociado, Departamento de Sistemas, Facultad de Ingeniería, Universidad de Nariño, Pasto (Colombia). Correo electrónico: ritimar@udenar.edu.co

tarea de minería de datos clasificación basado en los operadores algebraicos relacionales Mate, Entro, Gain y Describe Classifier, implementados en la cláusula SQL Select con las primitivas SQL Mate by, Entro(), Gain() y Describe Classification Rules, los cuales facilitan el cálculo de Ganancia de Información, la construcción del árbol de decisión y el acoplamiento fuerte de este algoritmo con un SGBD.

Palabras Claves: *Árboles de Decisión, Minería de Datos, Operadores Algebraicos Relacionales, Primitivas SQL, Tarea de Clasificación.*

Abstract

Decision tree classification is the most used and popular model, because it is simple and easy to understand. The calculation of the value of the measure that allows selecting, in each node, the attribute with the highest power to classify on the set of values of the class attribute, is the most expensive process in the used algorithm. To compute this measure, the data are not needed, but the statistics about the number of records in which combine the test attributes with the class attribute. Among the classification algorithms by decision trees are ID-3, C4.5, SPRINT and SLIQ. However, none of these algorithms are based on relational algebraic operators and are implemented with SQL primitives. In this paper Mate-tree, an algorithm for the classification data mining task based on the relational algebraic operators Mate, Entro, Gain and Describe Classifier, is presented. They were implemented in the SQL Select clause with SQL primitives Mate by, Entro(), Gain() y Describe Classification Rules. They facilitate the calculation of the Information Gain, the construction of the decision tree and the tight coupled of this algorithm with a DBMS.

Keywords: *Decision Trees, Data Mining, Relational Algebraic Operators, SQL Primitives, Classification Task.*

Introducción

El proceso de extracción de conocimiento a partir de grandes volúmenes de datos tiene como objetivo descubrir patrones que deben ser válidos, novedosos, interesantes y comprensibles para el usuario (Chen, Han & Yu (1996), Fayyad, Piatetsky-Shapiro & Smyth (1996a), Fayyad, Piatetsky-Shapiro & Smyth (1996b), Cabena et al. (1997)). La minería de datos es la etapa más importante de este proceso (Imielnski & Mannila (1996) y Hernández, Ramírez & Ferri (2005)).

La minería de datos consta de diferentes tareas, cada una de las cuales puede considerarse como un tipo de problema a ser resuelto por un algoritmo de minería de datos, afirman Adamo (2001) y Hernández, Ramírez & Ferri (2005), donde la tarea de clasificación por árboles de decisión es una de ellas.

Se han propuesto varias métricas para este proceso. Para Wang, Iyer & Scott (1998), el cálculo del valor de la métrica que permite seleccionar, en cada nodo, el atributo que tenga una mayor potencia para clasificar sobre el conjunto de valores del atributo clase, es la parte más costosa del algoritmo utilizado. Los algoritmos ID3 (Quinlan, 1986) y C4.5 (Quinlan, 1993) utilizan como métrica, para seleccionar el atributo candidato en cada nodo del árbol, la reducción de la entropía denominada *Ganancia de Información*, mientras que SLIQ (Metha, Agrawal & Rissanen, 1996) y SPRINT (Shafer, Agrawal & Metha, 1996), usan el *índice Gini*. Para el cálculo de estas métricas, no se necesitan los datos en sí, sino las estadísticas acerca del número de registros en los cuales se combinan los atributos condición con el atributo clase. Un operador algebraico relacional para clasificación basado en árboles de decisión debe facilitar estas combinaciones, que conjuntamente con operadores agregados, permita el cálculo de estas métricas.

En este artículo se propone *Mate-tree*, un algoritmo para la tarea de minería de datos clasificación basado en el operador algebraico relacional *Mate* que conjuntamente con los operadores agregados *Entro* y *Gain* facilitan el cálculo de la Ganancia de Información y con el operador algebraico relacional *Describe Classifier*, la construcción del árbol de decisión.

La implementación en el lenguaje SQL del operador relacional *Mate* dentro de la cláusula *Select*, como la primitiva *Mate by*, de *Entro* y *Gain* como las funciones agregadas SQL *Entro ()* y *Gain()* y de *Describe Classifier* como el operador SQL *Describe Classification Rules*, facilita la integración de *Mate-tree* al interior de un Sistema Gestor de Base de Datos (SGBD), acoplando la tarea de Clasificación de una manera fuerte, propuesta por Timarán (2001). Esta es una de las ventajas del algoritmo con respecto a los demás. Debido a que el algoritmo es ejecutado conjuntamente con los datos en el SGBD, su ventaja potencial es que resuelve los problemas de escalabilidad y rendimiento de las implementaciones débilmente acopladas. Al implementarse *Mate-tree* en el lenguaje SQL permite la realización de consultas de minería de datos *ad hoc* y el desarrollo de aplicaciones en esta área.

El artículo está organizado en secciones: en la primera se presentan algunos conceptos acerca de la tarea de minería de datos clasificación; en la sección 2 se definen los operadores del álgebra relacional que

forman parte del algoritmo *Mate-tree*; en la sección 3, se presentan las primitivas SQL con las cuales se implementa el algoritmo *Mate-tree;* en la sección 4, se describe el algoritmo *Mate-tree* y su implementación en SQL *Mate-tree*; y en la sección 5 se muestran las pruebas y evaluación de desempeño de los algoritmos *Mate-tree*, *C4.5* y *SLIQ*. Finalmente se presentan las conclusiones y trabajos futuros.

1. Conceptos preliminares

1.1 Tarea de clasificación

La clasificación de datos es el proceso por medio del cual se encuentran propiedades comunes entre un conjunto de objetos de una base de datos y se los clasifica en diferentes clases, de acuerdo al modelo aplicado. Este proceso de aprendizaje supervisado, se realiza en dos pasos:

- En el primer paso, se construye un modelo en el cual, cada tupla, de un conjunto de tuplas de la base de datos, tiene una clase conocida, determinada por uno de los atributos de la base de datos, llamado atributo clase, según Witten & Frank (2000). El conjunto de tuplas que sirve para construir el modelo se denomina conjunto de entrenamiento, siendo, de acuerdo con Han & Kamber (2001), cada tupla de este conjunto un ejemplo de entrenamiento.

- En el segundo paso, se usa el modelo para clasificar. Inicialmente, se estima la exactitud del modelo utilizando un conjunto de tuplas de la base de datos, generalmente diferente al de entrenamiento, cuya clase es conocida, denominado conjunto de prueba, según afirman Witten & Frank (2000). A cada tupla de este conjunto, consideran Han & Kamber (2001), se le denomina ejemplo de prueba.

1.2 Clasificación por árboles de decisión

La clasificación por árboles de decisión es, probablemente, el modelo más utilizado y popular por su simplicidad y facilidad para su entendimiento, de acuerdo con Han & Kamber (2001) y Sattler & Dunemann (2001). El conocimiento obtenido en el proceso de aprendizaje se representa mediante un árbol en el cual cada nodo interior contiene una pregunta sobre un atributo concreto (con un hijo por cada posible respuesta) y cada hoja del árbol se refiere a una decisión (una clasificación). Durante la etapa de construcción del árbol, en forma recursiva, cada conjunto de datos se divide en subconjuntos de acuerdo a un criterio de particionamiento, con el fin de escoger el atributo que mejor separe los ejemplos restantes en clases individuales. Seleccionar el mejor punto de particionamiento es la parte de la construcción del árbol que mayor tiempo consume (Sattler & Dunemann, 2001).

2. Operadores algebraicos relacionales para clasificación

Para lograr eficiencia en las operaciones de minería de datos, un nuevo operador algebraico debe facilitar los procesos de minería de datos computacionalmente más costosos. Los operadores algebraicos propuestos por Timarán & Millán (2006), con los que se extiende el álgebra relacional facilitan estos procesos en el descubrimiento de reglas de clasificación.

2.1 Operador Mate (m)

El operador *Mate* genera, por cada una de las tuplas de una relación, todas los posibles combinaciones de los valores no nulos de los atributos de una lista de atributos denominados Atributos Condición, con el valor no nulo del Atributo Clase. *Mate* realiza estas combinaciones, en una sola pasada sobre la tabla de entrenamiento.

Formalmente, sea $A= \{A1,..., An\}$ el conjunto de atributos de la relación R de grado n y cardinalidad m; $LC \subset A$, $LC \neq \phi$ la lista de atributos condición a combinar y n' el número de atributos de LC, $|LC| = n'$, $n'<n$, $Ac \in A$, $Ac \cup L = \phi$ el atributo clase con el que se combinarán los atributos de LC. El operador μ aplicado a la lista de atributos LC, al atributo clase Ac de la relación R se define:

$$\mu_{LC;Ac}(R) = \{ti(M) / M=LC \cup Ac, LC \subset A, / LC / = n', n'<n, Ac \in A, Ac \cap LC=\phi, ti=Xi, 1 \leq i \leq m', m'=(2^{n'}-1)*m, \forall_i \forall_k (X_i=<null, ...,v_i(A_k)... ,null,...,v_i(Ac)>, v_i(A_k) v_i(Ac) \neq null), 1 \leq k \leq n'\}$$

El operador m produce una relación cuyo esquema es $R(M)$, $M=LC \cup Ac$, de grado g, $g= n'+1$ y cuya extensión $r(M)$ de cardinalidad m', $m'= (2n'-1)*m$ es el conjunto de g-tuplas ti, tal que en cada g-tupla, únicamente toman valor los atributos que forman la combinación $X_i \in LC$, ($1 \leq k \leq n'$) y el atributo Ac. Los demás atributos se hacen nulos.

Ejemplo 1. Sea la relación $R(A, B, C,)$ de la Tabla 1. Obtener las diferentes combinaciones de los atributos A, B con el atributo C, es decir, $R1= \mu_{A, B; C}(R)$. El resultado del operador Mate $R1= \mu_{A, B; C}(R)$ se muestra en la misma tabla.

2.2 Operador agregado Entro

El operador agregado *Entro* permite calcular la entropía de una relación R con respecto a un atributo denominado atributo *condición* y un atributo *clase*.

Tabla 1. Relación R y Resultado de la operación R1=$m_{A,B};_C$(R)

	A	B	C
Relación R	a1	b1	c1
	a1	b2	c2
Resultado operación R1=mA,B;C(R)	a1	null	c1
	null	b1	c1
	a1	b1	c1
	a1	null	c2
	null	b2	c2
	a1	b2	c2

Formalmente, sea *A={A_1, ..., A_n,Ac}* el conjunto de atributos de la relación *R* con esquema *R(A)*, extensión *r(A),* grado *n* y cardinalidad *m*. Sea *t* el número de distintos valores del *atributo_clase Ac, Ac∈R(A)* que divide a *r(A)* en *t* diferentes clases, C_i *(1≤i≤.t).* Sea r_i el número de tuplas de *r(A)* que pertenecen a la clase C_i. Sea *q* el número de distintos valores *{v_1 (A_k), v2 (A_k),.., v_q(A_k)}* del *atributo* condición A_k, A_kÎ *R(A),* el cual *particiona* a *r(A)* en *q* subconjuntos {S_1,S_2, …S_q}, donde S_j contiene todos las tuplas de *r(A)* que tienen el valor $v_j(A_k)$ del atributo A_k. Sea s_{ij} el número de tuplas de la clase C_i en el subconjunto S_j. *Entro* (A_k; *Ac; R*), retorna la entropía de *R* con respecto al atributo A_k, que se obtiene de la siguiente manera:

$$Entro\ (A_k; Ac; R)=\{y|y=-\sum p_{ij}\log_2(p_{ij}),\ 1\le i\le .t,\ 1\le j\le q,\ p_{ij}=s_{ij}/|S_j|\ \}$$

donde $p_{ij}= s_{ij}/|S_j$ê es la probabilidad que una tupla en S_j pertenezca a la clase C_i.

La entropía de *R* con respecto al atributo clase *Ac* es:

$$Entro\ (Ac; Ac; R)=\{y|y=-\sum p_i log_2(p_i),\ 1\le i\le .t,\ p_i=r_i/m\}$$

donde p_i es la probabilidad que un tupla cualquier pertenezca a la clase C_i y r_i el número de tuplas de *r(A)* que pertenecen a la clase C_i.

2.3 Operador agregado Gain

El operador agregado *Gain* permite calcular la reducción de la entropía causada por el conocimiento del valor de un atributo de una relación y se define:

$$Gain(A_k; Ac; R)=\{y|y=Entro(Ac; Ac; R)-Entro(A_k;Ac; R)\}$$

donde *Entro* (*Ac; Ac; R*) es la entropía de la relación *R* con respecto al atributo clase *Ac* y *Entro*(A_k;*Ac; R*) es la entropía de la relación *R* con respecto al atributo A_k.

2.4 Operador agregado Describe Classifier (βm)

Describe Classifier (βm) es un operador unario, que toma como entrada la relación resultante de los operadores *Mate*, *Entro* y *Gain* y produce una nueva relación con los valores de los atributos que formarán los diferentes nodos del árbol de decisión. Este operador facilita la construcción del árbol de decisión y por consiguiente la generación de reglas de clasificación.

Formalmente, sea $A= \{A_1, .., A_n, E, G\}$ el conjunto de atributos de la relación *R* de grado *n+2* y cardinalidad *m*. El operador βm se define:

$$\beta\mu(R)=\{t_i(Y) \mid Y=\{N,P,A,V,C\}$$

donde,

t_i=<*val(N),null,val(A),null,null*> si t_i es nodo raiz,

t_i=<*val(N),val(P),val(A),val(V),val(C)*> si t_i es nodo hoja

t_i=<*val(N),val(P),val(A),val(V),null*> si t_i es nodo interno.

El operador βm aplicado a *R,* produce una nueva relación con esquema *R(Y), Y={N,P,A,V,C}* donde *N* es el número del nodo, *P* identifica al nodo padre, *A* identifica el nombre del atributo asociado a ese nodo, *V* es un valor para el atributo *A* y *C* es el atributo clase. Su extensión *r(Y),* está formada por un conjunto de tuplas en las cuales si los valores de los atributos son: *N*≠null, *P*=null, *A*≠null, *V*=null y *C*=null corresponde a un nodo raiz; si *N*≠null, *P*≠null, *A*≠null, *V*≠null y *C*≠null corresponde a una hoja o nodo terminal y si *N*≠null, *P*≠null, *A*≠null, *V*≠null y *C*=null corresponde a un nodo interno.

3. Primitivas SQL para clasificación

3.1 Primitiva Mate By

Esta primitiva implementa el operador algebraico *Mate* en la cláusula SQL SELECT*. Mate By* toma los valores de los atributos de una tabla denominados *atributos condición* y por cada registro forma todas las posibles combinaciones de estos atributos con otro atributo de la misma tabla denominado *atributo clase*. Este proceso lo realiza en una sola pasada sobre la tabla.

Dentro de la cláusula SELECT, *Mate By* tiene la siguiente sintaxis:

SELECT <*ListaAtributosTablaDatos*> [INTO <*NombreTablaMate*>]

FROM <*NombreTablaDatos*>

WHERE <*CláusulaWhere*>

MATE BY<*ListaAtributosCondicion*> WITH <*AtributoClase*>

GROUP BY <*ListaAtributosTablaDatos*>

<*ListaAtributosTablaDatos*> : := <*Atributo*>,<*ListaAtributos*>

<*ListaAtributos*>: :=<*Atributo*>, <*ListaAtributos*>

<*ListaAtributos*>: :=<*Atributo*>

<*ListaAtributosCondicion*>: :=<*Atributo*>, <*ListaAtributos*>

<*AtributoClase*>::=<*Atributo*>

3.2 Función agregada SQL Entro ()

Esta función agregada SQL implementa el operador algebraico agregado *Entro* en la cláusula SQL SELECT. SQL *Entro ()* permite calcular, conjuntamente con la primitiva *Mate By*, la entropía de una tabla con respecto a cada una de las combinaciones de los *atributos condición* con el *atributo clase*. SQL *Entro ()* se debe ejecutar conjuntamente con la función agregada *Count ()*. La sintaxis de SQL *Entro ()* en la cláusula SELECT es la siguiente:

SELECT <*ListaAtributosTablaDatos*>, *Count(*)*, *Entro(*)* [INTO <*NombreTablaMate*>]

FROM <*NombreTablaDatos*>

WHERE <*CláusulaWhere*>

MATE BY<*ListaAtributosCondicion*> WITH <*AtributoClase*>

GROUP BY <*ListaAtributosTablaDatos*>

3.3 Función agregada SQL Gain ()

Esta función agregada SQL, implementa el operador algebraico agregado *Gain* en la cláusula SQL SELECT. SQL *Gain()* permite calcular, conjuntamente con la primitiva *Mate by*, la ganancia de información de una tabla con respecto a cada una de las combinaciones de los *atributos condición* con el *atributo clase*. Internamente SQL *Gain()* calcula la entropía del *atributo clase*, por ello se debe ejecutar conjuntamente con las funciones agregadas *Count()* y *Entro()*. La sintaxis de *Gain ()* en la cláusula SELECT es:

SELECT <*ListaAtributosTablaDatos*>, *Count (*)*, *Entro (*), Gain (*)*

[INTO <*NombreTablaMate*>]

FROM <*NombreTablaDatos*>

WHERE <*CláusulaWhere*>

MATE BY<*ListaAtributosCondicion*> WITH <*AtributoClase*>

GROUP BY <*ListaAtributosTablaDatos*>

Ejemplo 2. Sea la tabla SINTOMAS (SID, D_MUSCULAR, TEMPERATURA, GRIPA). Calcular la entropía y ganancia de información de las diferentes combinaciones de los *atributos d_muscular*, *temperatura* con el atributo *gripa* y almacenar el resultado en la tabla *gansintomas*:

SELECT *d_muscular,temperatura,gripa, count(*), Entro(*), Gain(*)*

INTO *gansintomas*

FROM *sintomas*

MATE BY *d_muscular,tempertura* WITH *gripa*

GROUP BY *d_muscular,tempertura,gripa*

3.4 Cláusula SQL Describe Classification Rules

Describe Classification Rules implementa el operador algebraico *Describe Classifier* en una nueva cláusula SQL. Este operador SQL construye el árbol de decisión y genera las reglas de clasificación. El operador *Describe Classification Rules,* permite la construcción del árbol de decisión de manera unificada con el cálculo de la métrica de *particionamiento* con la primitiva *Mate By* y las funciones agregadas *Entro ()* y *Gain ()* en una sola instrucción SQL o en forma separada con sentencias SQL independientes, para luego generar las reglas. *Describe Classification Rules* tiene la siguiente sintaxis:

DESCRIBE CLASSIFICATION RULES

[INTO <*TablaReglasClasificación*>]

FROM <*NombreTablaArbol*>

USING <*NombreTablaMétrica*>

[DO <*SubconsultaCálculoMétrica*>]

<*SubconsultaCálculoMétrica*>::=<SFWMG>

<SFWMG>::=<SELECT FROM WHERE MATE BY GROUP BY>

4. Mate-tree, un algoritmo para clasificación por árboles de decisión

El algoritmo de clasificación *Mate-tree* se basa en los operadores algebraicos relacionales *Mate*, *Entro*, *Gain* y *Describe Classifier* propuestos por Timarán & Millán (2006) y Timarán (2007). *Mate* realiza las combinaciones de los atributos condición con el atributo clase, *Entro* y *Gain* calculan las métricas de Entropía y Ganancia de información y *Describe Classifier* facilita la construcción del árbol de

decisión. Este hecho facilita su integración al interior de cualquier SGBD, acoplando la tarea de Clasificación de una manera fuerte y favorece la aplicación de técnicas de optimización de consultas para mejorar su rendimiento.

4.1 Descripción del algoritmo Mate-tree

En el primer paso el algoritmo calcula la *Entropía* del de la relación *R* con respecto al *atributo clase*. En el subsiguiente paso, se aplica el operador *Mate* con el fin de generar todas las posibles combinaciones de los *atributos condición* con el *atributo clase.* Se calcula el número de ocurrencias de cada combinación y a la relación resultante *R1*, se aplica la funcion agregada *Entro()* que calcula para cada combinación de *atributos condición* y *clase* la entropía de la relación R1 con respecto a estos atributos. Posteriormente, se calcula con la función agregada *Gain()*, la Ganancia de Información obtenida por el particionamiento de la relación R1 por las diferentes combinaciones de *atributos condición* y *clase*. Finalmente, con la relación resultante R2, el operador *describe classifier* construye el árbol de decisión. El algoritmo *Mate-tree* se muestra en la figura 1a.

4.2 Implementación del algoritmo Mate-tree en el lenguaje SQL

El algoritmo *Mate-tree* se implementa en el lenguaje SQL, utilizando la primitiva SQL *Mate By,* el operador SQL *Describe Classification Rules* y la funciones agregadas *Entro ()* y *Gain()* definidos en Timarán & Millán (2006) y Timarán (2007). La implementación del algoritmo *Mate-tree* en el lenguaje SQL se muestra en la figura 1b.

Ejemplo 3. Sea la tabla CLIENTES (EDAD, INGRESOS, ES_ESTUDIANTE, MANEJOCREDITO, COMPRAEQUIPO) cuyos valores han sido *discretizados*. Utilizando el algoritmo SQL *Mate-tree*, construir el árbol de decisión y generar las reglas de clasificación, almacenándolas en la tabla *reglasClasificación.*

En este caso, el algoritmo SQL *Mate-tree* a través del operador *Describe Classification Rules* construye el árbol de decisión en la tabla *nodosarbol* a partir de la tabla *métricaspartición,* donde se encuentran almacenados los valores de las métricas de *Entropía* y *Ganancia de Información* de todas las combinaciones de los atributos condición *edad, ingresos, as_estudiante, manejocredito* con el atributo clase *compraequipo*. Finalmente, de la tabla *nodosarbol* se genera las reglas de clasificación, almacenándolas en la tabla *reglasClasificación*. El resultado se muestra en la figura 1c.

```
//Calcula entropía de R con respecto al atributo clase Ac
Entro(Ac;Ac, R);
Forall lt ∈ R do
begin
// Se aplica el operador Mate
                    R1=μLC,Ac(R)
        // Cuenta las ocurrencias de las diferentes combinaciones
  // de los atributos condición LC y atributo clase Ac
                    Count(R1)
  // calcula entropía
                    Entro(LC,R1)
  // calcula ganancia
    R2=Gain(LC,R1)
  End
  //construye el arbol de decisión con operador Describe Classifier
        Arbol=βμ(R2)
```

a. algoritmo Mate-tree

```
DESCRIBE CLASSIFICATION RULES
        FROM árbol
        USING R2
DO      SELECT LC, Ac, count(*), Entro() AS Entropía,
                Gain() AS Ganancia
INTO R2
                FROM R
                MATE BY LC WITH Ac
GROUP BY LC, Ac
```

b. implementación del algoritmo Mate-tree en el lenguaje SQL

```
DESCRIBE CLASSIFICATION RULES
        INTO reglasClasificación
        FROM nodosarbol
        USING métricaspartición
DO SELECT edad, ingresos, es_estudiante, manejocredito,
      compraequipo, count(*), Entro() AS Entropia, Gain() AS Ganancia
INTO métricaspartición
        FROM clientes
        MATE BY edad, ingresos, as_estudiante, manejocredito WITH compraequipo
GROUP BY edad, ingresos, es_estudiante, manejocredito, compraequipo
```

c. Algoritmo SQL Mate-tree para la clasificación de clientes

Figura 1. Algoritmo Mate-tree

5. Evaluación de desempeño del algoritmo Mate-tree

Para analizar el rendimiento del algoritmo *Mate-tree*, se realizaron varias pruebas en un equipo Intel Pentium IVF de 64 bits a 3,0 Ghz, memoria RAM de 2Gb, disco duro serial ata de 160 Gb con sistema operativo *Fedora core 5*. Se utilizaron los siguientes conjuntos de datos:

- *Zoo.data* del repositorio de la UCI (5 Kb) con 101 registros y 18 atributos con información de 101 animales,
- *Titanic.data* del repositorio de la UCI (120 Kb) con 2201 registros y 4 atributos, con información correspondiente a 2.201 pasajeros de este crucero,
- *Sisben.dat* (1.466 Kb) con 15.000 registros y 10 atributos, pertenecientes a 15.000 afiliados del régimen subsidiado de salud de la ciudad de Pasto (Colombia),
- *Udenar.dat* (1.366 Kb) con 20.328 registros y 26 atributos, con información correspondiente a la parte académica y social de 20.328 estudiantes de la Universidad de Nariño (Colombia). Los algoritmos utilizados para las pruebas fueron *Mate-tree*, C4.5 y SLIQ.

Para realizar las pruebas con los algoritmos *Mate-tree*, C4.5 y SLIQ, se implementaron en la herramienta *Mate-KDD*, un clasificador mediana-

mente acoplado con el SGBD PostgreSQL desarrollado en el laboratorio de KDD de la Universidad de Nariño (Colombia), como funciones definidas por el usuario (FDU) en el lenguaje *procedural* PL/PgSQL.

4.1 Pruebas

Se diseñaron tres tipos de prueba para evaluar el comportamiento de los algoritmos *Mate-tree, C4.5 y SLIQ* en diferentes condiciones.

En la primera prueba se mantiene constante el número de atributos y se incrementa el número de registros. Para esta prueba se utilizaron los conjuntos de datos *Titanic.dat* y *Sisben.dat.* El conjunto de datos *Titanic.data* se duplicó hasta obtener el número de registros que se requerían. Los resultados se muestran en la Figura 2a, estando el tiempo expresado en segundos.

Con el conjunto de datos real *Sisben.dat* se seleccionaron aleatoriamente seis subconjuntos de datos con diferente número de instancias y se midió el costo computacional de los algoritmos. Los resultados se muestran en la Figura 2b.

En la segunda prueba se mantiene constante el número de registros y se varía el número de atributos. Esta prueba se realizó con el conjunto de datos *udenar.dat.* Los resultados de esta prueba se muestran en la Figura 2c.

Teniendo en cuenta que el operador algebraico *Mate* realiza todas las combinaciones entre los *atributos condición* y el *atributo clase*, es el proceso más costoso del algoritmo *Mate-tree*, se diseñó la tercera prueba con el fin de medir el comportamiento del operador *Mate* en dos diferentes arquitecturas: mediana y fuertemente acoplada con el SGBD PostgreSQL. Por esta razón, el operador algebraico *Mate*, conjuntamente con la primitiva SQL *Mate by,* se implementaron en el interior del motor de PostgreSQL (Momjian (2001) y Geschwinde & Schoning (2002)) en una arquitectura fuertemente acoplada (Timarán, 2001) y como una FDU en PL/PgSQL en una arquitectura medianamente acoplada (Timarán, 2001). Para esta prueba se utilizó el conjunto de datos *zoo.data*, el cual se duplicó hasta obtener un máximo de 40.400 transacciones. Los resultados de esta prueba se muestran en la Figura 3.

4.2 Análisis de resultados

Analizando los resultados de las dos primeras pruebas se puede observar que el desempeño del algoritmo *Mate-tree* es mejor en conjuntos de datos con pocos atributos, como sucede con el conjunto de datos *Titanic. data*, pero su rendimiento empieza a degradarse cuando el número de atributos aumenta, como se puede deducir de la segunda prueba. En general, *Mate-tree* y C4.5 tienen un comportamiento similar, pues sus tiempos de ejecución son semejantes, lo que no se puede afirmar del algoritmo SLIQ, que es muy costoso.

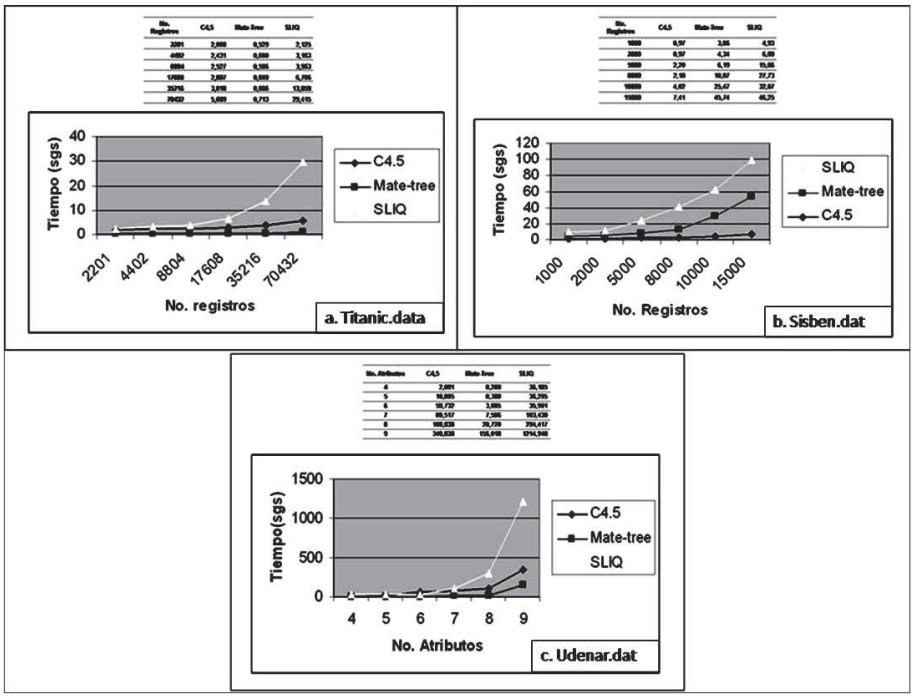

Figura 2. Resultado de las pruebas de rendimiento y comportamiento de los algoritmos con Titanic.data, Sisben.dat y Udenar.dat

No. Atributos	Mate Medianamente Acoplado	Mate Fuertemente Acoplado
4	0,1980	0,0989
5	0,3540	0,2150
6	0,7080	0,5648
7	1,0700	0,7147
8	4,3000	0,7148
9	10,5490	3,2942
10	25,0600	7,1928
11	61,7310	16,0551
12	169,9790	40,4997
13	241,2150	122,2415

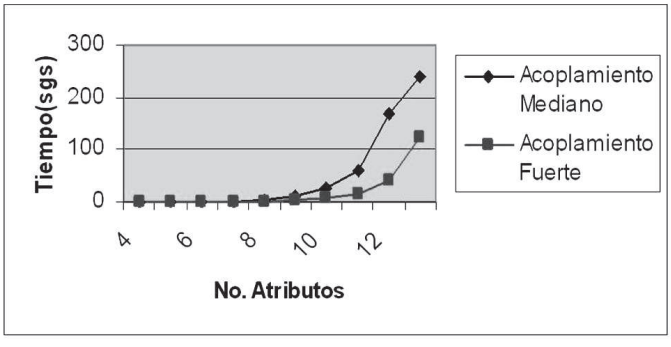

Figura 3. Resultado y comportamiento del operador Mate en diferentes arquitecturas

Como se puede observar en la figura 3, el rendimiento del operador *Mate* es mejor en su versión fuertemente acoplada, cuando el número de atributos aumenta, debido al número de combinaciones que se deben realizar. Sin embargo, *Mate*, en las dos arquitecturas tiende a presentar un comportamiento exponencial. Por esta razón, en el algoritmo *Mate-tree* se debe contemplar una pre-poda que evite combinaciones innecesarias.

5. Conclusiones y trabajos futuros

Mate-tree, es un algoritmo para descubrir reglas de clasificación basado en los operadores algebraicos relacionales *Mate*, *Entro*, *Gain* y *Describe Classifier* que lo diferencia del resto de algoritmos de Clasificación. Este hecho facilita la integración fuerte de este algoritmo con un SGBD, al extender el lenguaje SQL con las primitivas y funciones agregadas *Mate By*, *Entro ()*, *Gain()* y *Describe Classification Rules* que implementan fácilmente a *Mate-tree*, generando rápidamente, a través de consultas *ad-hoc* reglas de clasificación.

Los resultado de las pruebas con el algoritmo *Mate-Tree*, muestran que los tiempos de ejecución están directamente relacionados con el número de atributos y las distintas categorías del atributo clase. El tiempo de ejecución del algoritmo crece significativamente de acuerdo al grado y cardinalidad del conjunto de datos.

Las resultados con las pruebas realizadas con la implementación del operador algebraico *Mate* en una arquitectura medianamente y fuertemente acoplada con el SGBD PostgreSQL, demostraron la eficiencia de esta segunda arquitectura, debido a que *Mate* hace uso de todas las ventajas de estar integrado al motor de PostgreSQL, como el de ejecutarse en el mismo espacio de direccionamiento que los datos, ser un objeto de la base de datos, poder ser optimizable, entre otras.

Como trabajos futuros están, el de contemplar dentro del algoritmo *Mate-.tree* un sistema de prepoda, que evite que el operador *Mate,* haga combinaciones de los atributos *condición* con el atributo *clase* que no intervienen en la construcción del árbol de decisión. Integrar el algoritmo *Mate-tree* al interior del motor de PostgreSQL, acoplando fuertemente todos sus operadores algebraicos y realizar pruebas de desempeño en esta arquitectura.

Bibliografía

ADAMO, J.M. (2001). Data Mining for Association Rules and Sequential Patterns: Sequential and Parallel Algorithms. New York (USA): Springer-Verlag. 253 p. ISBN: 0-387-95048-6.

CABENA, P.; HADJINIAN, P.; STADLER, R.; VERHEES, J. & ZANASI, A. (1997). Discovering Data Mining from Concept to Implementation. New York (USA): Prentice Hall. 224 p. ISBN: 978-0137439805.

CHEN, M.; HAN, J. & YU, P. (1996). Data Mining: An Overview from Database Perspective. In: Journal IEEE Transactions on Knowledge Data Engineering, Vol. 8, No. 6 (Dec.). New Jersey (USA): IEEE Educational Activities Department Piscataway. p. 866-883. ISSN: 1041-4347.

FAYYAD, U.; PIATETSKY-SHAPIRO, G. & SMYTH, P. (1996a). From Data Mining to Knowledge Discovery: An Overview. In: Advances in Knowledge Discovery and Data Mining. Menlo Park (USA): AAAI Pres/The MIT Press. 595 p. ISBN: 0-262-56097-6.

FAYYAD, U.; PIATETSKY-SHAPIRO, G. & SMYTH, P. (1996b). The KDD Process for Extracting Useful Knowledge from Volumes of Data. In: Communications of the ACM, Vol. 39, No 11 (Nov.). New York, NY (USA): ACM. p. 27-34. ISSN: 0001-0782.

GESCHWINDE, E. & SCHONING, H.J. (2002). PostgreSQL: Developer's Handbook. Indianapolis (Indiana, USA): Sams Publishing. 751 p. ISBN: 0-672-32260-9.

HAN, J. y KAMBER, M. (2001). Data Mining: Concepts and Techniques. San Diego (USA): Morgan Kaufmann Publishers. 745 p. ISBN: 978-1-55860-901-3.

HERNÁNDEZ, O.J.; RAMÍREZ, Q.M. & FERRI, R.C. (2005). Introducción a la Minería de Datos. Madrid (España): Pearson Prentice Hall. 656 p. ISBN: 84-205-4091-9.

IMIELNSKI, T. & MANNILA, H. (1996). A Database Perspective on Knowledge Discovery. In: Communications of the ACM. Vol. 39, No. 11 (Nov.): New York (USA): ACM. p. 58-64. ISSN: 0001-0782.

METHA, M.; AGRAWAL, R. & RISSANEN, J. (1996). SLIQ: A Fast Scalable Classifier for Data Mining. In: The 5th International Conference on Extending Database Technology EDBT, (25-29/03/1996), Avignon (France): Advances in Database Technology. Proceedings Lecture Notes in Computer Science Springer. p. 18-32. ISBN: 3-540-61057-X.

MOMJIAN, B. (2001). PostgreSQL: Introduction and Concepts. New York (USA): Addison-Wesley. 455 p. ISBN: 0-201-70331-9.

QUINLAN, J.R. (1986). Induction of Decision Trees. In: Machine Learning Journal, Vol. 1, No. 1 (Jan.). Boston (USA): Kluwer Academic Publishers. p. 81-106.

QUINLAN, J.R. (1993). C4.5: Programs for Machine Learning. San Francisco, CA (USA): Morgan Kaufmann Publishers. 299 p. ISBN: 1-55860-238-0.

SATTLER, K. & DUNEMANN, O. (2001). SQL Database Primitives for Decision Tree Classifiers. In: The 10th ACM International Conference on Information and Knowledge Management - CIKM, (5-10/11/2001), Atlanta, Georgia (USA): ACM. Proceedings, p. 379-386. ISBN: 1-58113-436-3.

SHAFER, J.; AGRAWAL, R. & METHA, M. (1996). SPRINT: A Scalable Parallel Classifier for Data Mining. In: The 22th International Conference on Very Large Data Bases – VLDB, (3-6/09/1996), Mumbai, Bombay (India): ACM. Proceedings Morgan Kaufmann, p. 544-555. ISBN: 1-55860-382-4

TIMARÁN, R. (2001). Arquitecturas de Integración del Proceso de Descubrimiento de Conocimiento con Sistemas de Gestión de Bases de Datos: un Estado del Arte. En: Revista Ingeniería y Competitividad. Vol. 3, No. 2 (dic.). Cali (Colombia): Universidad del Valle. ISSN: 0123-3033.

TIMARÁN, R. & MILLÁN, M. (2006). New Algebraic Operators and SQL Primitives for Mining Classification Rules. In: The Second IASTED International Conference on Computational Intelligence - CI 2006, (20-22/11/2006), San Francisco, CA (USA): The International Association of Science and Technology for Development. Proceedings, p. 59 - 63. ISBN: 0-88986-603-1.

TIMARÁN, R. (2007). Extensión del Lenguaje SQL con Nuevas Primitivas para el Descubrimiento de Reglas de Clasificación. En: VI Jornadas Iberoamericanas de Ingeniería del Software e Ingeniería del Conocimiento - JIISIC, (31/01-02/02/2007), Lima (Perú): Facultad de Ciencias e Ingeniería, Pontificia Universidad Católica del Perú. Memorias, p. 19-26. ISBN: 978-9972-2885-1-7.

WANG, M.; IYER, B. & SCOTT, V., J. (1998). Scalable Mining for Classification Rules in Relational Databases. In: International Database Engineering and Application Symposium - IDEAS, (08-10/07/1998), Cardiff, Wales (U.K.). IEEE Computer Society. Proceedings, p. 58-67. ISBN: 0-8186-8307-4.

WITTEN, I. & Frank, E. (2000). Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations. San Francisco, CA (USA): Morgan Kaufmann Publishers. 365 p. ISBN: 1-55860-552-5.