MANUAL DE REFERENCIA

MATE-KDD: Data Mining Tool V.2

Universidad de Nariño
Facultad de Ingeniería
Departamento de Sistemas
Grupo de Investigación
GRIASKDD
2017

TABLA DE CONTENIDO

INTRODUCCIÓN

MATE-KDD Data Mining Tool V. 2, es una herramienta de minería de datos para la tarea de clasificación que cuenta con los algoritmos C4.5, Mate-tree y SLIQ implementados través de funciones definidas por el usuario (FDUs), a partir de datos almacenados en una base de datos en el sistema gestor de bases de datos PostgresSQL., en una arquitectura de acoplamiento mediana.

Ofrece soporte en todas las fases del proceso de Descubrimiento de Conocimiento en Bases de Datos (Selección, preprocesamiento, transformación, minería de datos y evaluación), además que opera directamente dentro del SGBD al aplicar los algoritmos implementados en FDU's, lo que significa, que su arquitectura es medianamente acoplada con el Gestor, esto es favorable en el sentido en que trabaja directamente con una base de datos y aplica el proceso de minería desde el gestor ganando en velocidad de procesamiento. Para manejar la herramienta se cuenta con una interfaz grafica que permite al usuario (analista) interactuar con el Gestor de una forma fácil, se puede configurar el clasificador seleccionando el algoritmo a aplicar, definir el porcentaje de partición de los datos para entrenamiento y prueba para luego aplicar el modelo y obtener unos resultados de forma grafica al poder visualizar el árbol que se genera, y las reglas de una forma detallada y clara.

El proyecto se desarrollo bajo filosofía de Software Libre, la herramienta funciona sobre una plataforma Linux, la distribución que se utilizo fue Ubuntu 14.04.03.

Se debe tener instalado Postgresql-9.4, cuyo código fuente se encuentra incluido en el CD de instalación. En caso de preferir utilizar otra versión de Postgresql es necesario copiar el directorio **"kdd"** al directorio **"contrib"** de la otra versión instalada de Postgresql.

1. INSTALACIÓN DE LA HERRAMIENTA

Para la instalación de MATE-KDD se debe tener en cuenta los pasos que se describen a continuación.

1.1 Instalación del SGBD PostgreSQL

- Se ingresa como usuario root y se desplaza al directorio </usr/local/src/> donde se ubicaran las fuentes de postgresql

  ```
  $ su
  # cd /usr/local/src/
  # cp /cdrom/postgreskdd/postgresql-KDD.tar.gz .
  ```

- Se desempaqueta el archivo y se ingresa en el directorio generado.

  ```
  # tar xvfz postgresql-KDD.tar.gz
  # cd postgresql-7.3.4
  ```

- Antes de iniciar la instalación se configura el código fuente de la siguiente forma:

  ```
  # ./configure
  ```

- Para compilar e instalar, se digitan las siguientes secuencias de comandos:

  ```
  # make
  # make install
  ```

- Se crea el grupo postgres y usuario postgres.

  ```
  # groupadd postgres

  # adduser --home /usr/local/pgsql --ingroup postgres postgres
  ```

- Se crea el directorio de datos y de generación de archivos de seguimiento, y se le asigna permisos de propietario al usuario postgres.

  ```
  # mkdir /usr/local/pgsql/data
  # chown - R postgres:postgres /usr/local/pgsql/
  ```

- Se inician las bases de datos básicas para el correcto funcionamiento de PostgreSQL y se ejecuta el servidor de PostgreSQL. Para esto, es necesario identificarse como usuario postgres.

  # su – postgres

- Se crea las variables de ambiente de postgres en el archivo .profile

  $ pico .profile
  PGUSER=postgres
  PGLIB=/usr/local/pgsql/lib
  PGDATA=/usr/local/pgsql/data
  PATH=$PATH:/usr/local/pgsql/bin
  export PGUSER PGLIB PGDATA PATH

  Guardar
  <crtl><x><y><enter>

- Se reinicia la sesión de postgres
  $exit
  # su – postgres

- Se inicializa los parámetros iniciales de las bases de datos en postgres

  $ initdb

- Se levanta el servicio postgres

  $ postmaster -D /usr/local/pgsql/data&

- Se crea la base de datos pruebadb y se conecta a ella
  $ createdb pruebadb
  $ psql pruebadb
  pruebadb=#

- Se ejecuta el script de prueba llamado jugartenis.sql
  pruebadb=# \i /opt/Matekdd/repositorios/jugartenis.sql

- Una vez instalado y configurado Postgres, se procede a instalar la herramienta MATE_KDD en el directorio **<</opt/>>** como root

  # cd /opt/
  # cp /cdrom/Software/matekdd.tar.gz **.**

- Se descomprime y desempaqueta el archivo y se ingresa en el directorio generado.

  # gunzip matekdd.tar.gz.
  # tar –xvf matekdd.tar
  # cd matekdd

- Antes de ejecutar por primera vez la herramienta se debe cambiar los permisos del directorio **/models** y su contenido.

  # chmod –777 /opt/Matekdd/models/

Para ejecutar la herramienta se ingresa al directorio **<<…/dist>>** en modo grafico

  # cd /opt/Matekdd/dist
  # java –jar Matekdd.jar

1.2 Creación de las Funciones FDUs

- Como root copiar todas las funciónes al directorio /opt/matekdd/Matekdd/functions/ y cambiar de grupo y dueño a postgres

  # cd /opt/Matekdd/functions
  # cp –R /cdrom/Software/Matekdd/functions/ **.**
  # chown – R postgres:postgres /opt/Matekdd/functions

- Como usuario postgres se inicia la terminal interactiva de postgresql y se crean las funciones dentro de la base de datos

  $ psql <nombre_base_datos>

  $ psql pruebadb

  pruebadb=# \i /opt/Matekdd/functions/matekdd.sql

1.3 Obtener y Validar el Modelo C4.5

  # select * from trans_c45_mate('<nombre_tabla>', '<nombre_att_clase>', <rango_ discretizacion>,<num_instacias_prueba>);

select * from c45(‘<nombre_att_clase>’);

select * from test_rules(‘c45’);

select * from translate_rules(<rango_discretizacion>, ’c45’, ’<nombre_att_clase>’);

select * from arm_rules(‘<nombre_att_clase>’, ’c45’);

1.4 Obtener y Validar el Modelo MATE-TREE

select * from trans_c45_mate(‘<nombre_tabla>’, ’<nombre_att_clase>’, <rango_ discretizacion>,<num_instacias_prueba>);

select * from matetree(‘<nombre_att_clase>’);

select * from test_rules(‘mate’);

select * from translate_rules(<rango_discretizacion>, ’mate’, ’<nombre_att_clase>’);

select * from arm_rules(‘<nombre_att_clase>’, ’mate’);

1.5 Obtener y Validar el Modelo SLIQ

select * from trans_sliq(‘<nombre_tabla>’, ’<nombre_att_clase>’, <num_instacias_prueba>);

select * from sliq(‘<nombre_att_clase>’);

select * from test_sliq(‘<nombre_att_clase>’);

select * from arm_rules(‘<nombre_att_clase>’,’sliq’);

2. ASPECTOS DE IMPLEMENTACIÓN DE LA HERRAMIENTA MATEKDD

En esta sección se describe detalladamente la implementación y funcionamiento de los algoritmos de clasificación C4.5, Mate-Tree y SLIQ así como los procesos para la fase de selección, preprocesamiento, transformación y validación, estos fueron codificados en su totalidad como funciones definidas por el usuario en el SGBD PostgreSQL utilizando el lenguaje procedural PL/pgSQL.

2.1 Programación de Funciones en PL/pgSQL

PostgreSQL le proporciona al usuario la capacidad de diseñar e implementar funciones de propósito especializado o general. Dichas funciones se escriben en un lenguaje de programación anfitrión C y en un lenguaje basado en SQL como PL/pgSQL. El lenguaje PL/pgSQL es uno de los más utilizados dentro de PostgreSQL, debido a que guarda cierta similitud con PL/SQL de Oracle y a su facilidad de uso.

SQL es el lenguaje estándar para realizar consultas a un servidor de base de datos. Cada sentencia SQL se ejecuta de manera individual por el servidor, lo cual implica que las aplicaciones cliente deben enviar cada consulta al servidor, esperar a que la procese, recibir los resultados, procesar los datos y después enviar la siguiente sentencia. Al usar PL/pgSQL es posible realizar cálculos, manejo de cadenas y consultas dentro del servidor de la base de datos, combinando el poder de un lenguaje procedimental y la facilidad de uso de SQL, minimizando el tiempo de conexión entre el cliente y el servidor

El lenguaje PL/pgSQL es estructura en bloques. Todas las palabras clave y los identificadores pueden escribirse mezclando letras mayúsculas y minúsculas. Un bloque se define de la siguiente manera:

[<<label>>]
[DECLARE
    declaraciones]

COMENTARIOS, CONSTANTES Y VARIABLES

```
BEGIN
        Sentencias
END;
```

Pueden existir varios bloques o sub-bloques en la sección de sentencias de un bloque. Los sub-bloques pueden ser usados para ocultar las variables a los bloques más externos. Normalmente una de las sentencias es el valor de retorno, usando la palabra clave RETURN.

Las variables declaradas en la sección que antecede a un bloque se inicializan a su valor por omisión cada vez que se entra al bloque, no solamente al ser llamada la función. Por ejemplo:

```
CREATE FUNCTION estafunc() RETURNS INTEGER AS '
DECLARE
        cantidad INTEGER := 30;
BEGIN
        RAISE NOTICE "Cantidad contiene aqui %",cantidad;
        -- Cantidad contiene aqui 30
        cantidad := 50;
        --
        -- Creamos un sub-bloque
        --
        DECLARE
        cantidad INTEGER := 80;
        BEGIN
        RAISE NOTICE "Cantidad contiene aqui %",cantidad;
        -- Cantidad contiene aqui 80
        END;
        RAISE NOTICE "Cantidad contiene aqui %",cantidad;
        -- Cantidad contiene aqui 50
        RETURN cantidad;
END;
' LANGUAGE 'plpgsql';
```

No se debe confundir el uso de las sentencias de agrupamiento BEGIN/END de PL/pgSQL con los comandos de la base de datos que sirven para el control de las transacciones. Las funciones y procedimientos disparadores no pueden iniciar o realizar transacciones y Postgres no soporta transacciones anidadas.

Variables y constantes

Todas las variables, filas y registros usados en un bloque o en sus sub-bloques deben declararse en la sección de declaraciones del bloque.

La excepción es la variable de un ciclo FOR que itera sobre un rango de valores enteros. Las variables en PL/pgSQL pueden ser de cualquier tipo de datos de SQL, como INTEGER, VARCHAR y CHAR. El valor por omisión de todas las variables es el valor NULL de SQL.

A continuación se muestran algunos ejemplos de declaración de variables:

```
user_id INTEGER;
quantity NUMBER(5);
url VARCHAR;
```

Expresiones

Todas las expresiones usadas en las sentencias de PL/pgSQL son procesadas usando el ejecutor del backend. Las expresiones que parecen contener constantes pueden requerir de una evaluación en tiempo de ejecución (por ejemplo, now para el tipo de dato timestamp) así que es imposible para el analizador sintáctico (parser) de PL/pgSQL identificar los valores de las constantes reales diferentes de NULL. Todas las expresiones son evaluadas internamente ejecutando una sentencia SELECT expresión usando el gestor de SPI. En la expresión, las ocurrencias de los identificadores de las variables se sustituyen por parámetros y los valores reales de las variables se pasan al ejecutor en el arreglo de parámetros. Todas las expresiones usadas en una función de PL/pgSQL son preparadas y almacenadas solamente una vez.

La única excepción a esta regla es una sentencia EXECUTE si se requiere analizar una consulta cada vez que es encontrada. La revisión del tipo realizada por el analizador sintáctico principal de Postgres tiene algunos efectos secundarios a la interpretación de valores constantes.

Sentencias

Cualquier cosa no comprendida por el analizador PL/pgSQL tal como se especifica adelante seria enviado al gestor de la base de datos, para su ejecución. La consulta resultante no devolvería ningún dato.

Asignación

Una asignación de un valor a una variable o campo de fila o de registro se escribe:

```
identifier := expression;
```

Sentencias

Si el tipo de dato resultante de la expresión no coincide con el tipo de dato de las variables, o la variable tienen un tamaño o precisión conocido (como char(29)), el resultado sería amoldado implícitamente por el interprete de bytecode de PL/pgSQL, usando los tipos de las variables para las funciones de entrada y los tipos resultantes en las funciones de salida. Nótese que esto puede potencialmente producir errores de ejecución generados por los tipos de las funciones de entrada.

Una asignación de una selección completa en un registro o fila puede hacerse del siguiente modo:

SELECT expressions INTO target FROM ...;

Donde *target* puede ser un registro, una variable de fila o una lista separada por comas de variables y campo de de registros o filas.

Si una fila o una lista de variables se usa como objetivo, los valores seleccionados han de coincidir exactamente con la estructura de los objetivos o se produciría un error de ejecución. La palabra clave FROM puede preceder a cualquier calificador valido, agrupación, ordenación, etc. que pueda pasarse a una sentencia SELECT.

Existe una variable especial llamada FOUND de tipo booleano, que puede usarse inmediatamente después de SELECT INTO para comprobar si una asignación ha tenido exito.

```
SELECT * INTO myrec FROM EMP WHERE empname = myname;
IF NOT FOUND THEN
RAISE EXCEPTION "employee % not found", myname;
END IF;
```

Si la selección devuelve múltiples filas, solo la primera se mueve a los campos objetivo; todas las demás se descartan.

2.2. Creación de las funciones FDU en Postgres.

Debido a que PostgreSQL (al menos hasta la versión 7.3.4) no ofrece soporte para incluir coherentemente funciones externas como funciones agregadas que el *backend* pueda reconocer y tratar directa y eficientemente, incluye entre sus directorios uno por defecto para que el usuario pueda una vez creadas sus funciones establecer aquí la jerarquía necesaria para su correcto funcionamiento, aunque bien se puede evitar este formalismo es recomendable hacer uso de él. Este es el directorio *Contrib* ubicado en *./postgresql/contrib.*

Es necesario para el funcionamiento de una función escrita en un lenguaje procedural basado en SQL como PL/pgSQL, registrar el lenguaje para su uso en una base de datos especifica. Esto se hace accediendo como usuario postgres (o un usuario con privilegios equivalentes) y dando desde el prompt ($) la instrucción:
CREATELANG PLPGSQL <<dbname>>

Donde *dbname* es el nombre de la Base de Datos específica.

Para que una FDU pueda ejecutarse es necesario además del código fuente, construir los archivos *.sql* donde se especifica la instrucción de instalación para Postgres y las fuentes. Así creación de la función se realiza accediendo a la base de datos, ejecutando los programas monitores de Postgres como psql (que permite introducir, editar y ejecutar comandos SQL interactivamente). Luego, se utiliza el comando de psql "\i" para leer peticiones a partir del archivo:
\i /ruta_archivo/nombre_archivo.

Con esta instrucción se crea la función, y se obtiene el mensaje de confirmación 'CREATE FUNCTION'.

A partir de este momento se puede invocar a la función con los parámetros requeridos. Para este proyecto se ha dispuesto de un archivo iniciador.

El desarrollo de las funciones definidas por el usuario se ha realizado por conjuntos de acuerdo a la fase que se busque realizar. De este modo se ha dividido las funciones en seis grupos: primero el grupo para el preprocesamiento, en donde se realiza las tareas de selección de datos. Este grupo es manejado por el usuario. El segundo grupo en donde se depura, discretiza y transforma los datos a minar. El tercer grupo construye el modelo de clasificación empleando el algoritmo C4.5. El cuarto grupo construye el modelo usando el algoritmo Matre-Tree. El quinto grupo construye el modelo con el algoritmo SLIQ y por último el sexto grupo que contiene las funciones para validar e interpretar el modelo generado.

Después de la instalación de la herramienta y si se prefiere trabajar en modo consola, para crear las funciones aquí construidas se utiliza el comando:

\i. /opt/Matekdd/functions/matekdd.sql.

2.3 Implementación de funciones para la fase de preprocesamiento.

Los archivos que contienen las fuentes de estas funciones están ubicadas en el directorio: */opt/Matekdd/functions/Preprocess/*, y son estas funciones las que

apoyan la fase de selección ya que el usuario final puede aplicarlas a una tabla para obtener un grupo de datos con características especificas. Corresponden a:

- process_del.sql: Contiene las fuentes de la función process_del().
- process_add.sql: Contiene las fuentes de la función process_add().
- process_dtz.sql: Contiene las fuentes de la función process_dtz().
- process_nlz.sql: Contiene las fuentes de la función process_nlz().
- process_fcd.sql: Contiene las fuentes de la función process_fcd().
- process_rdm.sql: Contiene las fuentes de la función process_rdm().

Para la utilización de las funciones, se ingresa como usuario Postgres o equivalente, a la base de datos a través de la terminal interactiva psql. Una vez dentro se ejecuta la instrucción:

\i /opt/Matekdd/functions/Process/process_del.sql

\i /opt/Matekdd/functions/Process/process_add.sql

\i /opt/Matekdd/functions/Process/process_dtz.sql

\i /opt/Matekdd/functions/Process/process_nlz.sql

\i /opt/Matekdd/functions/Process/process_fcd.sql

\i /opt/Matekdd/functions/Process/process_rdm.sql

Con estas instrucciones se crea las funciones, y obtiene el mensaje de confirmación ‘CREATE FUNCTION’ para cada una.

A partir de este momento ya es posible invocar a las funciones, haciendo uso de los parámetros requeridos.

2.3.1 Funcionamiento de la FDU process_del(). La función *process_del()* recibe como parámetros de entrada un tipo de dato VARCHAR que contiene el nombre de la tabla que se está trabajando y un tipo de dato VARCHAR que contiene el nombre del atributo a eliminar. Ya que no es recomendable modificar la tabla original esta función hace una copia omitiendo el atributo seleccionado para eliminar.

Si por ejemplo sobre la tabla *"JugarTenis"* (ver tabla 1), se aplica la función *process_del*('Jugar_Tenis','temperatura'), se obtiene como resultado la tabla *'fl_Jugar_Tenis'* (ver tabla 2).

Tabla 1. JugarTenis

Estado	Temperatura	Humedad	Viento	Jugar
Soleado	85	85	Debil	No
Soleado	80	90	Fuerte	No
Nublado	83	78	Debil	Si
Lluvioso	70	96	Debil	Si
Lluvioso	68	80	Debil	Si
Lluvioso	65	70	Fuerte	No
Nublado	64	65	Fuerte	Si
Soleado	72	95	Debil	No
Soleado	69	70	Debil	Si
Lluvioso	75	80	Debil	Si
Soleado	75	70	Fuerte	Si
Nublado	72	90	Fuerte	Si
Nublado	81	75	Debil	Si
Lluvioso	71	80	fuerte	No

Tabla 2. fl_JugarTenis Resultado de process_del()

Estado	Humedad	Viento	Jugar
Soleado	85	Debil	No
Soleado	90	Fuerte	No
Nublado	78	Debil	Si
Lluvioso	96	Debil	Si
Lluvioso	80	Debil	Si
Lluvioso	70	Fuerte	No
Nublado	65	Fuerte	Si
Soleado	95	Debil	No
Soleado	70	Debil	Si
Lluvioso	80	Debil	Si
Soleado	70	Fuerte	Si
Nublado	90	Fuerte	Si
Nublado	75	Debil	Si
Lluvioso	80	fuerte	No

2.3.2 Funcionamiento de la FDU process_add(). La función *process_add()* recibe como parámetros de entrada un tipo de dato VARCHAR que contiene el nombre de la tabla que se está trabajando, un tipo de dato VARCHAR que contiene el nombre del nuevo atributo a adicionar, un tipo de dato VARCHAR que contiene la operación a aplicar, un tipo de dato INTEGER que contiene el numero de un primer atributo, un tipo de dato FLOAT que puede contener el numero de

otro atributo o un numero cualquiera para una operación matemática (suma, resta, multiplicación, división) y un tipo de datos INTEGER que contiene la opción del proceso a aplicar. Entonces, si el valor del último parámetro es cero se realiza una operación entre los dos atributos seleccionados y si el valor es igual a uno se realiza la operación entre un atributo y un valor. El resultado de esta va en un atributo que se adiciona de último a una copia de la tabla original.
Si por ejemplo sobre la tabla *"JugarTenis"* (ver tabla 1), se aplica la función *process_add*('JugarTenis','viv',2,2,'/',1), se obtiene como resultado la tabla *'fl_JugarTenis'* (ver tabla 3.).

Tabla 3. fl_JugarTenis Resultado de process_add()

Estado	Temperatura	Humedad	Viento	Jugar	viv
Soleado	85	85	Debil	No	42
Soleado	80	90	Fuerte	No	40
Nublado	83	78	Debil	Si	41
Lluvioso	70	96	Debil	Si	35
Lluvioso	68	80	Debil	Si	34
Lluvioso	65	70	Fuerte	No	32
Nublado	64	65	Fuerte	Si	32
Soleado	72	95	Debil	No	66
Soleado	69	70	Debil	Si	34
Lluvioso	75	80	Debil	Si	37
Soleado	75	70	Fuerte	Si	37
Nublado	72	90	Fuerte	Si	36
Nublado	81	75	Debil	Si	40
Lluvioso	71	80	fuerte	No	35

2.3.3 Funcionamiento de la FDU process_dtz(). La función *process_dtz()* recibe como parámetros de entrada un tipo de dato VARCHAR que contiene el nombre de la tabla, un tipo de dato INTEGER que contiene el numero de rangos dispuestos para discretización y un tipo de dato INTEGER que contiene el numero del atributo a discretizar. Así, si el ultimo atributo es igual a cero se realiza la discretización de todos los atributos de tipo numérico que tenga la tabla pero si este parámetro es mayor que cero indicara el numero del atributo que se va a discretizar. La Discretización consiste en reemplazar los valores continuos por el rango al que pertenece. Este cambio se lo realiza en una copia de la tabla. Además se genera una tabla *pg_rangos* en la que se guarda la información correspondiente para la interpretación de las reglas.

Si por ejemplo sobre la tabla *"JugarTenis"* (tabla 1, se aplica la función *process_dtz*('JugarTenis',3,2), se obtiene como resultado la tabla *'fl_Juga_Tenis'* (ver tabla 4.).

Tabla 4. fl_JugarTenis Resultado de process_dtz()

Estado	Temperatura	Humedad	Viento	Jugar
Soleado	[78.000,85.000]	85	Debil	No

Soleado	[78.000,85.000]	90	Fuerte	No
Nublado	[78.000,85.000]	78	Debil	Si
Lluvioso	[64.000,71.000)	96	Debil	Si
Lluvioso	[64.000,71.000)	80	Debil	Si
Lluvioso	[64.000,71.000)	70	Fuerte	No
Nublado	[64.000,71.000)	65	Fuerte	Si
Soleado	[71.000,78.000)	95	Debil	No
Soleado	[64.000,71.000)	70	Debil	Si
Lluvioso	[71.000,78.000)	80	Debil	Si
Soleado	[71.000,78.000)	70	Fuerte	Si
Nublado	[71.000,78.000)	90	Fuerte	Si
Nublado	[78.000,85.000]	75	Debil	Si
Lluvioso	[71.000,78.000)	80	fuerte	No

Y si por ejemplo sobre la tabla *"JugarTenis"* (tabla 1.), se aplica la función *process_dtz*('JugarTenis',3,0), se obtiene como resultado la tabla *'fl_Juga_Tenis'* (ver tabla 5).

Tabla 5. fl_JugarTenis Resultado de process_dtz()

Estado	Temperatura	Humedad	Viento	Jugar
Soleado	[78.000,85.000]	[75.333,85,666)	Debil	No
Soleado	[78.000,85.000]	[85,666,96,000]	Fuerte	No
Nublado	[78.000,85.000]	[75.333,85,666)	Debil	Si
Lluvioso	[64.000,71.000)	[85,666,96,000]	Debil	Si
Lluvioso	[64.000,71.000)	[75.333,85,666)	Debil	Si
Lluvioso	[64.000,71.000)	[65.000,75.333)	Fuerte	No
Nublado	[64.000,71.000)	[65.000,75.333)	Fuerte	Si
Soleado	[71.000,78.000)	[85,666,96,000]	Debil	No
Soleado	[64.000,71.000)	[65.000,75.333)	Debil	Si
Lluvioso	[71.000,78.000)	[75.333,85,666)	Debil	Si
Soleado	[71.000,78.000)	[65.000,75.333)	Fuerte	Si
Nublado	[71.000,78.000)	[85,666,96,000]	Fuerte	Si
Nublado	[78.000,85.000]	[65.000,75.333)	Debil	Si
Lluvioso	[71.000,78.000)	[75.333,85,666)	fuerte	No

A parte de la tabla con los datos discretizados esta la tabla *pg_rangos* (ver tabla 6.)

Tabla 6. pg_rangos

nro	Atributo	Min	Max	equivalente
1	Temperatura	64.000	71.000	[64.000,71.000)
2	Temperatura	71.000	78.000	[71.000,78.000)

3	Temperatura	78.000	85.000	[78.000,85.000]
4	Humedad	65.000	75.333	[65.000,75.333)
5	Humedad	75.333	85.666	[75.333,85,666)
6	Humedad	85.666	96.000	[85,666,96,000]

2.3.4 Funcionamiento de la FDU process_nlz(). La función *process_nlz()* recibe como parámetros de entrada un tipo de dato VARCHAR que contiene el nombre de la tabla un tipo de dato INTEGER que contiene el numero del atributo a normalizar. Así, si el ultimo atributo es igual a cero se realiza la normalización de todos los atributos de tipo numérico que tenga la tabla pero si este parámetro es mayor que cero indicara el numero del atributo que se va a normalizar. La normalización consiste en reemplazar los valores continuos por su equivalente en el rango 0 a 1. Este cambio se lo realiza en una copia de la tabla.

Si por ejemplo sobre la tabla *"Jugar_Tenis"* (tabla 1.), se aplica la función *process_nlz*('JugarTenis',2), se obtiene como resultado la tabla *'fl_Juga_Tenis'* (ver tabla 7.).

Tabla 7. fl_JugarTenis Resultado de process_nlz()

Estado	Temperatura	Humedad	Viento	Jugar
Soleado	1.000	85	Debil	No
Soleado	0.941	90	Fuerte	No
Nublado	0.976	78	Debil	Si
Lluvioso	0.824	96	Debil	Si
Lluvioso	0.800	80	Debil	Si
Lluvioso	0.765	70	Fuerte	No
Nublado	0.753	65	Fuerte	Si
Soleado	0.847	95	Debil	No
Soleado	0.812	70	Debil	Si
Lluvioso	0.882	80	Debil	Si
Soleado	0.882	70	Fuerte	Si
Nublado	0.847	90	Fuerte	Si
Nublado	0.953	75	Debil	Si
Lluvioso	0.835	80	fuerte	No

Y si por ejemplo sobre la tabla *"JugarTenis"* (tabla 1), se aplica la función *process_nlz*('JugarTenis',0), se obtiene como resultado la tabla *'fl_JugarTenis'* (ver tabla 8).

Tabla 8. fl_JugarTenis Resultado de process_nlz()

Estado	Temperatura	Humedad	Viento	Jugar
Soleado	1.000	0.885	Debil	No
Soleado	0.941	0.938	Fuerte	No

Nublado	0.976	0.813	Debil	Si
Lluvioso	0.824	1.000	Debil	Si
Lluvioso	0.800	0.833	Debil	Si
Lluvioso	0.765	0.729	Fuerte	No
Nublado	0.753	0.677	Fuerte	Si
Soleado	0.847	0.990	Debil	No
Soleado	0.812	0.729	Debil	Si
Lluvioso	0.882	0.833	Debil	Si
Soleado	0.882	0.729	Fuerte	Si
Nublado	0.847	0.938	Fuerte	Si
Nublado	0.953	0.781	Debil	Si
Lluvioso	0.835	0.833	fuerte	No

2.3.5 Funcionamiento de la FDU process_fcd(). La función *process_fcd()* recibe como parámetros de entrada un tipo de dato VARCHAR que contiene el nombre de la tabla que se está trabajando y un tipo de dato VARCHAR que contiene la condición para la selección de registros. Esta función hace una copia de la tabla original y en ella copia las instancias que cumplen la condición indicada y retorna el nombre de la tabla creada.

Si por ejemplo sobre la tabla *"Jugar_Tenis"* (tabla 1), se aplica la función *process_fcd*('Jugar_Tenis','humedad>70'), se obtiene como resultado la tabla *'fl_JugarTenis'* (ver tabla 9).

Tabla 9. fl_JugarTenis Resultado de process_fcd()

Estado	Temperatura	Humedad	Viento	Jugar
Soleado	85	85	Debil	No
Soleado	80	90	Fuerte	No
Nublado	83	78	Debil	Si
Lluvioso	70	96	Debil	Si
Lluvioso	68	80	Debil	Si
Soleado	72	95	Debil	No
Lluvioso	75	80	Debil	Si
Nublado	72	90	Fuerte	Si
Nublado	81	75	Debil	Si
Lluvioso	71	80	fuerte	No

2.3.6 Funcionamiento de la FDU process_rd(). La función *process__rdm()* recibe como parámetros de entrada un tipo de dato VARCHAR que contiene el nombre de la tabla y junto con un tipo de dato INTEGER que contiene el numero del tuplas para el nuevo conjunto de datos.

Si por ejemplo sobre la tabla *"Juga_Tenis"* (tabla 1), se aplica la función *process_rdm*('JugarTenis',7), se obtiene como resultado la tabla *'fl_JugarTenis'* (ver tabla 10.).

Tabla 10. fl_Jugar_Tenis Resultado de process_rdm()

Estado	Temperatura	Humedad	Viento	Jugar
Soleado	85	85	Debil	No
Nublado	83	78	Debil	Si
Lluvioso	65	70	Fuerte	No
Nublado	64	65	Fuerte	Si
Lluvioso	75	80	Debil	Si
Soleado	75	70	Fuerte	Si
Lluvioso	71	80	fuerte	No

2.4 Implementación de funciones para la fase de transformación.

Los archivos que contienen las fuentes de estas funciones deben estar ubicadas en el directorio: */opt/Matekdd/functions/Transform*, tienen como propósito depurar, discretizar y transformar los datos para su posterior utilización y corresponden a:

- purify.sql: Contiene las fuentes de la función purify().
- transform.sql: Contiene las fuentes de la función transform().
- filter.sql: Contiene las fuentes de la función filter().

Para la utilización de las funciones, se ingresa como usuario Postgres o equivalente, a la base de datos a través de la terminal interactiva psql. Una vez dentro se ejecuta la instrucción:

\i /opt/Matekdd/functions/Transform/purify.sql

\i /opt/Matekdd/functions/Transform/transform.sql

\i /opt/Matekdd/functions/Transform/filter.sql

Con estas instrucciones se crea las funciones, y obtiene el mensaje de confirmación 'CREATE FUNCTION' para cada una.

A partir de este momento ya es posible invocar a las funciones, haciendo uso de los parámetros requeridos. Cabe resaltar que la función que divide el conjunto de datos en datos de entrenamiento y prueba es la misma función process_rdm() mencionada anteriormente.

2.4.1 Funcionamiento de la FDU purify(). La función *purify()* recibe como parámetro de entrada un tipo de dato VARCHAR que contiene el nombre de la tabla a la cual se le aplicara el proceso y genera como resultado una tabla "*kdd_list"*. Esta función se encarga de depurar la tabla validando los valores nulos y les asigna un valor promedio si es un atributo de tipo numérico y si es un tributo

categórico le asigna el que mayor frecuencia obtenga. En el caso de que no haya necesidad de depurar la tabla simplemente realiza una copia de la tabla.

Si por ejemplo sobre la tabla *"JugarTenis3"* (ver tabla 11) se aplica la función *purify*('JugarTenis3'), se obtiene como resultado la tabla *'kdd_list'* (ver tabla 12) ya depurada.

Tabla 11. Jugar_Tenis3

Estado	Temperatura	Humedad	Viento	Jugar	viv
Soleado	85	85	Debil	No	42
Soleado	80	90	Fuerte	No	40
Nublado	83	78	Debil	Si	41
Lluvioso	70		Debil	Si	35
Lluvioso	68	80	Debil	Si	34
Lluvioso		70	Fuerte	No	32
Nublado	64	65	Fuerte	Si	32
Soleado	72	95	Debil	No	66
Soleado	69	70	Debil	Si	34
Lluvioso		80	Debil	Si	37
Soleado	75	70	Fuerte	Si	37
Nublado	72	90	Fuerte	Si	36
Nublado	81		Debil	Si	40
Lluvioso	71	80	fuerte	No	35

Tabla 12. kdd_list Resultado de purify()

Estado	Temperatura	Humedad	Viento	Jugar	viv
Soleado	85	85	Debil	No	42
Soleado	80	90	Fuerte	No	40
Nublado	83	78	Debil	Si	41
Lluvioso	70	96	Debil	Si	35
Lluvioso	68	80	Debil	Si	34
Lluvioso	65	70	Fuerte	No	32
Nublado	64	65	Fuerte	Si	32
Soleado	72	95	Debil	No	66
Soleado	69	70	Debil	Si	34
Lluvioso	65	80	Debil	Si	37
Soleado	75	70	Fuerte	Si	37
Nublado	72	90	Fuerte	Si	36
Nublado	81	75	Debil	Si	40
Lluvioso	71	80	fuerte	No	35

2.4.2 Funcionamiento de la FDU transform(). La función *transform()* recibe como parámetro de entrada un tipo de dato VARCHAR que contiene el nombre del atributo clase. Recorre la tabla para transformar los valores contenidos. Convierte la tabla *"kdd_list"* en una tabla transformada *"kdd_ent"* que contienen únicamente valores numéricos y organiza los atributo de tal forma que el atributo clase quede al final, Crea también la tabla *'kdd_values'*, que guarda los parámetros de transformación utilizados, necesarios para la conversión cuando sea necesario

expresar los valores resultantes finales del proceso de clasificación en términos de los valores originales.

Por ejemplo, sobre la tabla *"kdd_list"* de la Tabla 5 (debidamente depurada) Al aplicar la función transform('jugar'), se obtiene la tabla *"kdd_ent"* ya transformada (ver tabla 13) y la tabla *'kdd_values'* (ver tabla 14).

Tabla 13. kdd_ent

Estado	Temperatura	Humedad	Viento	Jugar
0	2	1	0	0
0	2	2	1	0
2	2	1	0	1
1	0	2	0	1
1	0	1	0	1
1	0	0	1	0
2	0	0	1	1
0	1	2	0	0
0	0	0	0	1
1	1	1	0	1
0	1	0	1	1
2	1	2	1	1
2	2	0	0	1
2	1	1	1	0

Tabla 14. kdd_values

Nro	Atributo	Valor	Discret	nclase
1	Estado	Soleado	0	
2	Estado	Lluvioso	1	
3	Estado	Nublado	2	3
4	temperatura	[64.000,71.000)	0	
5	Temperatura	[71.000,78.000)	1	
6	Temperatura	[78.000,85.000]	2	3
7	Humedad	[65.000,75.333)	0	
8	Humedad	[75.333,85,666)	1	
9	Humedad	[85,666,96,000]	2	3
10	Viento	Debil	0	
11	Viento	Fuerte	1	2
12	Jugar	No	0	
13	Jugar	Si	1	2

Si se utiliza para la función transform() la tabla *'JugarTenis2'* (tabla 15.). Al aplicar la función transform('jugar'), se obtiene la tabla *"kdd_ent"* 2 (tabla 16.) y la tabla *'kdd_values'* 2 (tabla 17.).

Tabla 15. JugarTenis2

Estado	Temperatura	Humedad	Viento	Jugar
Soleado	Caliente	Alta	Debil	No
Soleado	Caliente	Alta	Fuerte	No
Nublado	Caliente	Alta	Debil	Si
Lluvioso	Templado	Alta	Debil	Si
Lluvioso	Fresco	Normal	Debil	Si
Lluvioso	Fresco	Normal	Fuerte	No
Nublado	Fresco	Normal	Fuerte	Si
Soleado	Templado	Alta	Debil	No
Soleado	Fresco	Normal	Debil	Si
Lluvioso	Templado	Normal	Debil	Si
Soleado	Templado	Normal	Fuerte	Si
Nublado	Templado	Alta	Fuerte	Si
Nublado	Caliente	Normal	Debil	Si
Lluvioso	templado	Alta	fuerte	No

Tabla 16. kdd_ent 2

Estado	Temperatura	Humedad	Viento	Jugar
0	0	0	0	0
0	0	0	1	0
2	0	0	0	1
1	2	0	0	1
1	1	1	0	1
1	1	1	1	0
2	1	1	1	1
0	2	0	0	0
0	1	1	0	1
1	2	1	0	1
0	2	1	1	1
2	2	0	1	1
2	0	1	0	1
2	2	0	1	0

Tabla 17. Tabla kdd_values 2

Nro	Atributo	Valor	Discret	nclase
1	Estado	soleado	0	
2	Estado	lluvioso	1	
3	Estado	nublado	2	3
4	temperatura	Caliente	0	
5	Temperatura	Fresco	1	
6	Temperatura	Templado	2	3
7	Humedad	Alta	0	
8	Humedad	Normal	1	2
9	Viento	Debil	0	

10	Viento	Fuerte	1	2
11	Jugar	No	0	
11	Jugar	Si	1	2

2.4.3 Funcionamiento de la FDU filter(). La función *filter()* recibe como parámetro de entrada un tipo de dato VARCHAR que contiene el nombre del atributo clase. Trabaja con la tabla *"kdd_ent"*- Se encarga de validar las contradicciones que se presenten el conjunto de datos. Mas exactamente si encuentra una condición que pertenezca a más de una clase, calcula la probabilidad de ocurrencia en las clases y deja en la tabla los registros que cumplen con la condición y la clase con mayor probabilidad.

2.5 Implementación del algoritmo C4.5 para la fase de minería

Los archivos que contienen las fuentes de estas funciones están ubicadas en el directorio: */opt/Matekdd/functions/AlgorihtmC45/*, tienen como propósito generar reglas de clasificación aplicando a los datos de una tabla el algoritmo C4.5 y corresponden a:

- c45_algorithm.sql: Contiene las fuentes de la función principal c*45().*
- c45_entro.sql: Contiene las fuentes de la función c45_entr().
- c45_gain.sql: Contiene las fuentes de la función c45_gain().
- c45_tree.sql: Contiene las fuentes de la función c45_tree().
- view_set.sql: Contiene las fuentes de la función view_set().

Para la utilización de la función C***45()***, se ingresa como usuario Postgres o equivalente, a la base de datos a través de la terminal interactiva psql. Una vez dentro se ejecuta la instrucción:

\i /opt/Matekdd/functions/AlgorihtmC45/c45_algorithm.sql

\i /opt/Matekdd/functions/AlgorihtmC45/c45_entro.sql

\i /opt/Matekdd/functions/AlgorihtmC45/c45_gain.sql

\i /opt/Matekdd/functions/AlgorihtmC45/c45_tree.sql

\i /opt/Matekdd/functions//AlgorihtmC45/view_set.sql

Con estas instrucciones se crea las funciones, y obtiene el mensaje de confirmación 'CREATE FUNCTION' para cada una.

A partir de este momento ya es posible invocar a la función c45() haciendo uso de los parámetros requeridos.

2.5.1 Funcionamiento de la FDU c45(). La función c45() recibe como parámetro un tipo de dato VARCHAR que es el nombre del atributo clase del conjunto de datos. Esta es la función principal del algoritmo y trabaja con una tabla *''kdd_ent''* en donde deben estar los datos de entrenamiento debidamente transformados. También requiere de las sub funciones c45_entro(), c45_gain(), c45_tree() y view_set(). Esta función se encarga primero que todo de crear físicamente las tablas *'c45_rules'*, *'c45_nodes'* como las tablas auxiliares que utiliza en el transcurso del proceso de modelado. Seguidamente llama a la función c45_entro() la cual retorna la entropía de la clase y sigue buscando el atributo que va a ser utilizado para el nodo raíz utilizando la función c45_gain(). Y por ultimo llama a la función c45_tree() quien es la que termina de crear el modelo de clasificación.

Tanto esta función como c45_tree() utilizan una tabla temporal llamada "pg_columna" que actúa como un vector y lleva el orden y el numero de los atributos que componen una camino del árbol (o sea, una rama).

En la figura 1 se muestra las relaciones de las funciones necesarias en el desarrollo del algoritmo C4.5.

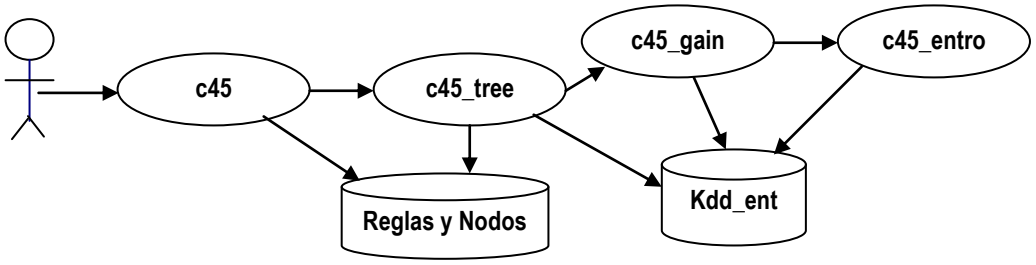


Figura 1. Diagrama de Interrelación de Funciones en C4.5

2.5.2 Funcionamiento de la FDU c45_tree(). La función *c45_tree()* recibe como parámetro un tipo de dato INTEGER que contiene el número de atributos hasta el momento van conformando una condición.

Esta función es llamada por la función principal que es c45() o por sí misma. Empieza insertando el nodo actual en la tabla de nodos *"c45_nodes"* luego valida

que este nodo sea una hoja Terminal, si esto es así, construye la reglas recorriendo la tabla auxiliar *"pg_columna"* y la inserta en la tabla de reglas *"c45_rules"*. Si no es un nodo terminal continúa el análisis primero por la izquierda y luego siguiendo por la derecha, tomando los atributos que no han sido utilizados y aquel atributo que tenga la mayor ganancia (para obtener la ganancia de información utiliza la función *c45_gain()* es el nodo que sigue en el árbol en seguida llama de nuevo a la función *c45_tree().*

2.5.3 Funcionamiento de la FDU c45_entro(). La función *c45_entro()* recibe como parámetro un tipo de dato INTEGER que contiene el numero de la opción solicitada, más claramente si se requiere la entropía de la clase o la entropía de un grupo especifico de atributos, y junto con un tipo de datos VARCHAR que contiene el nombre del atributo clase.

Esta función calcula la entropía aplicando la formula de entropía de Shannon, para el algoritmo C4.5, y retorna el valor de la entropía que es un dato de tipo NUMERIC.

2.5.4 Funcionamiento de la FDU c45_gain(). La función *c45_gain()* recibe como parámetro un tipo de dato FLOAT que contiene un valor de entropía, dato necesario para obtener la ganancia, junto un tipo de dato VARCHAR que contiene el nombre del atributo clase de la tabla. Esta función calcula la ganancia de un atributo utilizando la función entropía() de Shannon,para el algoritmoC4.5.

2.5.5 Funcionamiento de la FDU view_set(). La función *view_set()*. Esta función selecciona un subgrupo de datos relacionados al nodo en turno para su posterior análisis. Crea la vista view1 con la misma estructura de la tabla kdd_ent.

2.5.6. Ejemplo de la Función c45(). Utilizando la tabla *'kdd_ent'* transformada (tabla 13), se aplica la función c45('jugar') y como resultado se obtienen las tablas *'c45_rules'* (tabla 18.) y *'c45_nodes'* (tabla 19).

Tabla 18. Tabla c45_rules

Id	Atributo	valor
1	1	0
1	3	0
1	5	1
2	1	0
2	3	1
2	5	0
3	1	1
3	4	0
3	5	1
4	1	1
4	4	1
4	5	0
5	1	2

5	5	1

Tabla 19. c45_nodes

Nodo	Padre	Atributo	Valor	Hijo	Hemí	Clase
0	-1	1	-1	-1	-1	-1
1	0	1	0	3	2	-1
2	0	1	1	5	-1	-1
3	1	3	0	-1	4	1
4	1	3	1	-1	-1	0
5	4	4	0	-1	6	1
6	4	4	1	-1	7	0
7	0	1	2	-1	-1	1

Las estructuras de las tablas *'c45_rules'* y *'c45_nodes'* se muestran en las tablas 18 y 19 respectivamente. La tabla *'c45_rules'* presenta los nodos relacionados en una misma regla de forma secuencial. Todos los registros que conforman una regla tienen el mismo valor en el atributo id, que indica el número de la regla. El valor constante '-999' indica el caso especial en que el atributo puede ser cualquiera de los posibles valores del atributo clase.

La tabla *'c45_nodes'* especifica las relaciones entre los nodos del árbol, y los valores útiles. Todos los valores están transformados, y por tanto pueden traducirse en resultados reales y coherentes. En este caso el valor constante '-999' tiene el mismo significado que en la tabla *'c45_rules'*.

Esta estructura será igualmente utilizada para mostrar los resultados del modelo obtenido con el algoritmo Mate-Tree que se describe a continuación.

2.6 Implementación del algoritmo Mate-Tree para la fase de minería

Los archivos que contienen las fuentes de estas funciones están ubicadas en el directorio: */opt/Matekdd/functions/AlgorihtmMate/*, tienen como propósito generar reglas de clasificación a partir de los datos de una tabla, aplicando el algoritmo Mate-Tree y corresponden a:

- matetree.sql: Contiene las fuentes de la función principal matetree*()*.
- mate_tree.sql: Contiene las fuentes de la función principal mate_tree*()*.
- mateby.sql: Contiene las fuentes de la función mateby().
- posiciones.sql: Contiene las fuentes de la función controlposicion().
- mate_gain.sql: Contiene las fuentes de la función principal mate_gain*()*.

Para la utilización de la función **m*atetree()***, se ingresa como usuario Postgres o equivalente, a la base de datos a través de la terminal interactiva psql. Una vez dentro se ejecuta las instrucciones:

\i /opt/Matekdd/functions/AlgorihtmMate/mate_gain.sql

\i /opt/Matekdd/functions/AlgorihtmMate/mateby.sql

\i /opt/Matekdd/functions/AlgorihtmMate/posicion.sql

\i /opt/Matekdd/functions/AlgorihtmMate/matetree.sql

\i /opt/Matekdd/functions/AlgorihtmMate/mate_tree.sql

Con estas instrucciones se crea las funciones, y obtiene el mensaje de confirmación 'CREATE FUNCTION' para cada una.

A partir de este momento ya es posible invocar a la función matetree() o cada una de las funciones: mateby(), luego entro() y por ultimo gain(), haciendo uso de los parámetros requeridos.

2.6.1 Funcionamiento de la FDU matetree(). La función *matetree()* recibe como parámetro un tipo de dato VARCHAR que contiene el nombre del atributo clase de la tabla. Esta función tiene como fin generar un modelo de clasificación. Esta es la función principal la cual utiliza las funciones *mateby()* y *entro()* y luego junto las funciones *mate_tree()* y *mate_gain* construye el modelo de árboles de decisión.

En la figura 2 se muestra las relaciones de las funciones necesarias en el desarrollo del algoritmo Mate-Tree.

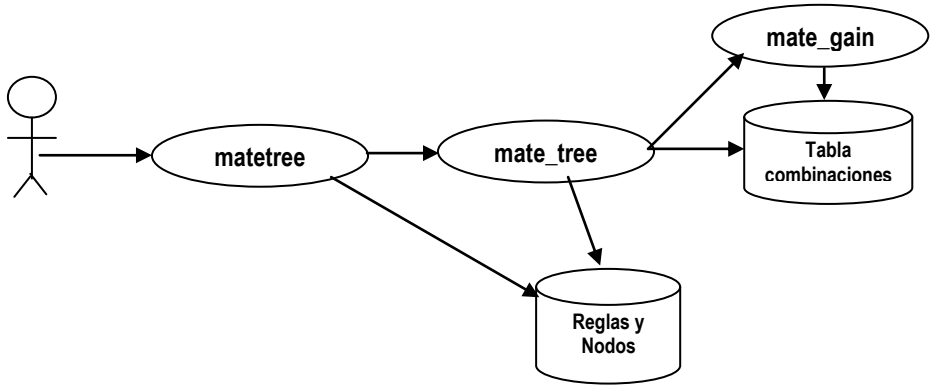


Figura 2. Diagrama de Interrelación de Funciones en Mate-Tree

A continuación se describe el funcionamiento de cada una de las funciones que componen la implementación del algoritmo Mate-Tree, y se muestra el ejemplo paralelamente.

2.6.2 Funcionamiento de la FDU mateby(). La función mateby() recibe como parámetro un tipo de dato VARCHAR que contiene el nombre del atributo clase de la tabla y un ipo de dato INTEGER que contiene el valor minimo de soporte de prepoda. Esta función tiene como fin generar por cada una de las tuplas todas las posibles combinaciones formadas por los valores no nulos de los atributos decisión y el atributo clase de la tabla tratada y además los cuenta, además debe actuar como un proceso de prepoda.

Por ejemplo, sobre la tabla *'kdd_ent'* (tabla 16). Al aplicar la función mateby('jugar') sobre ésta tabla se obtiene la tabla *'mate_clase'* (ver tabla 20).

Tabla 20. mate_clase

Estado	Temperatura	Humedad	Viento	Jugar	Count
0	0	0	0	1	1
0	0	0		1	1
0	0		0	1	1
0	0			1	1
0	1	0	1	1	1
0	1	0		1	1
0	1	2	0	0	1
0	1	2		0	1
0	1		0	0	1
0	1		1	1	1
0	1			0	1
.					
.					
..					

2.6.3 Funcionamiento de la FDU mate_gain(). La función *mate_gain()* trabaja con la tabla *'mate_clase'* (tabla 20). Calcula la ganancia de información de los atributos decisión que cumplan unas determinadas característica y devuelve este valor la funcion mate_tree().

2.6.4 Funcionamiento de la FDU mate_tree(). La función *mate_tree()* es la que se encarga de seguir los pasos lógicos para encontrar el nodo en turno.

2.6.5. Ejemplo de la Función matetree(). Utilizando la tabla *'kdd_ent'* transformada (tabla 16), se aplica la función matetree('jugar') y como resultado se obtienen las tablas *'mate_rules'* (tabla 21) y *'mate_nodes'* (tabla 22). Estas tablas utilizan la misma estructura que las tablas generadas por C4.5.

Tabla 21. mate_rules

Id	Atributo	valor
1	1	0
1	3	0
1	5	1
2	1	0
2	3	1
2	5	0
3	1	1
3	4	0
3	5	1
4	1	1
4	4	1
4	5	0
5	1	2
5	5	1

Tabla 22. mate_nodes

Nodo	Padre	Atributo	Valor	Hijo	Hem	Clase
0	-1	1	-1	-1	-1	-1
1	0	1	0	3	2	-1
2	0	1	1	5	-1	-1
3	1	3	0	-1	4	1
4	1	3	1	-1	-1	0
5	4	4	0	-1	6	1
6	4	4	1	-1	7	0
7	0	1	2	-1	-1	1

2.7 Implementación del algoritmo SLIQ para la fase de minería

Los archivos que contienen las fuentes de estas funciones están ubicadas en el directorio: */opt/Matekdd/functions/*AlgorithmSliq, tienen como propósito generar reglas de clasificación aplicando a los datos de una tabla el algoritmo SLIQ y corresponden a:

- sliq_algorithm.sql: Contiene las fuentes de la función principal sliq*().*
- sliq_tree.sql: Contiene las fuentes de la función sliq_tree.
- sliq_gini.sql: Contiene las fuentes de la función sliq_gini().

- sliq_combinar.sql: Contiene las fuentes de la función sliq_combinar().

Para la utilización de la función ***sliq()***, se ingresa como usuario Postgres o equivalente, a la base de datos a través de la terminal interactiva psql. Una vez dentro se ejecuta la instrucción:

\i */opt/Matekdd/functions/*AlgorithmSliq *sliq_algorithm.sql*

\i */opt/Matekdd/functions/*AlgorithmSliq *sliq_tree.sql*

\i */opt/Matekdd/functions/*AlgorithmSliq *sliq_gini.sql*

\i */opt/Matekdd/functions/*AlgorithmSliq *sliq_combinar.sql*

Con estas instrucciones se crea las funciones, y obtiene el mensaje de confirmación 'CREATE FUNCTION' para cada una.

A partir de este momento ya es posible invocar a la función sliq() haciendo uso de los parámetros requeridos.

2.7.1 Funcionamiento de la FDU sliq(). La función sliq() recibe como parámetro un tipo de dato VARCHAR que contiene el nombre del atributo clase de la tabla. Esta función trabaja sobre la tabla *"kdd_ent"* que es la tiene los datos de entrenamiento debidamente depurados y ya que este algoritmo trabaja tanto con atributo categóricos como con atributos numéricos no es necesario discretizar ni transformar su valores. Y empieza creando las tablas *"sliq_rules"* y *"sliq_nodes"* para las reglas y los nodos del modelo respectivamente así como las tablas auxiliares que son utilizadas en el proceso, también llama a la función sliq_gini() con el fin de encontrar el nodo raíz y luego llama a la función sliq_tree() la cual se encarga de el resto del árbol de decisión.

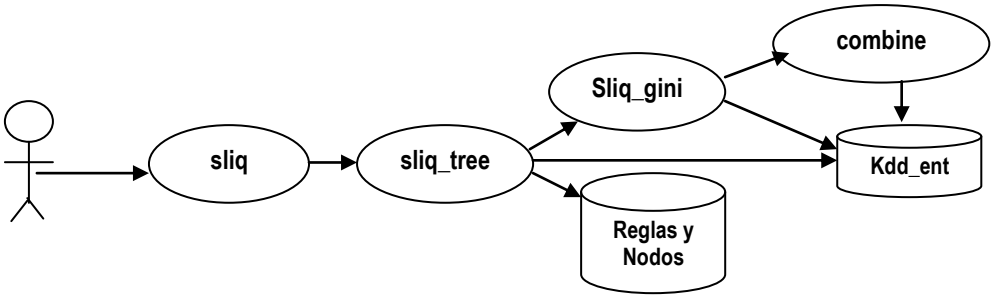


Figura 3. Diagrama de Interrelación de Funciones en Sliq

A continuación se describe el funcionamiento de cada una de las funciones que componen la implementación del algoritmo SLIQ

2.7.2 Funcionamiento de la FDU sliq_tree(). La función *sliq_tree()* recibe como parámetro un tipo de dato VARCHAR que contiene el nombre del atributo clase de la tabla y un tipo de dato INTEGER que contiene la opción a aplicar.

Esta función es llamada por la función principal que es sliq() o por si misma. Empieza insertando el nodo actual en la tabla de nodos *"sliq_nodes"* luego valida que este nodo sea una hoja Terminal, si esto es así, construye la reglas recorriendo la tabla auxiliar *"pg_columna"* y la inserta en la tabla de reglas *"sliq_rules"*. Si no es un nodo terminal continúa el análisis usando la función sliq_gini() y sliq_set().

2.7.3 Funcionamiento de la FDU sliq_gini(). La función *sliq_gini()* recibe como parámetro un tipo de dato VARCHAR que contiene el nombre del atributo clase de la tabla. Esta función tiene como fin encontrar el valor del próximo nodo calculando las ganancias de cada atributo. Así, si el atributo es numérico calcula la ganancia de los valores que estén entre v_i y v_{i+1}, donde los v_i son los valores ordenados de un atributo, si el atributo es categórico se llama a la función *sliq_combinar()* y calcula la ganancia de los subconjuntos diferentes de un atributo (es decir, 2^n donde es el numero de valores del atributo). De estos se selecciona el de menor ganancia y se ubica la condición en el nodo correspondiente.

2.7.4 Funcionamiento de la FDU sliq_combinar(). La función *sliq_combinar()* tiene como fin generar todas las posibles combinaciones entre los valores de un atributo categórico.

2.7.5 Primer ejemplo de la Función sliq(). Ya que este algoritmo no requiere de transformación de datos se lo aplicara sobre la tabla *'Jugar_Tenis'* (tabla 1) que para el ejemplo será la tabla *'kdd_ent'*. Al aplicar la función sliq('jugar') sobre ésta tabla se obtiene la tabla *'sliq_rules'* (tabla 23) y la tabla *"sliq_nodes"* (tabla 24).

Tabla 23. sliq_rules 1

N	Id	Atributo	Valor	Com	1tipo
1	1	Estado	Nublado	=	2
1	1	Jugar	Si	=	2
2	1	Estado	Nublado	!=	2
2	1	Temperatura	77.5	<=	1
2	1	Estado	Lluvioso	=	2
2	1	Viento	Debil	=	2
2	1	Jugar	Si	=	2
3	1	Estado	Nublado	!=	2
3	1	Temperatura	77.5	<=	1

3	1	Estado	Lluvioso	=	2
3	1	Viento	Debil	!=	2
3	1	Jugar	No	=	2
4	1	Estado	Nublado	!=	2
4	1	Temperatura	77.5	<=	1
4	1	Estado	Lluvioso	!=	2
4	1	Humedad	82.5	<=	1
4	1	Jugar	Si	=	2
5	1	Estado	Nublado	!=	2
5	1	Temperatura	77.5	<=	1
5	1	Estado	Lluvioso	!=	2
5	1	Humedad	82.5	>	1
5	1	jugar	No	=	2
6	1	Estado	Nublado	!=	2
6	1	Temperatura	77.5	>	1
6	1	jugar	No	=	2

Tabla 24. Tabla sliq_nodes 1

Nodo	Padre	Atributo	Valor	Com	Hijo	Herm	clase
1	-1	Estado	Nublado	=	-1	-1	Si
2	1	Temperatura	77.5	<=	4	3	-1
3	1	Temperatura	77.5	>	-1	-1	No
4	2	Estado	Lluvioso	=	5		
5	4	Viento	Debil	=			
6	5	Jugar	Si	=			
7	5	Jugar	No	=			
8	4	Humedad	82.5	<=			
9	8	Jugar	Si	=			
10	8	Jugar	No	=			

Las estructuras de las tablas *'sliq_rules'* y *'sliq_nodes'* se muestran en las tablas 23 y 24 respectivamente. La tabla *'sliq_rules'* presenta los nodos relacionados en una misma regla de forma secuencial. Todos los registros que conforman una regla tienen el mismo valor en el atributo *'id'*, que indica el número de la regla. El campo *'tipo'* contiene el tipo de dato a comparar (1 indica tipo numérico y 2 indica tipo categórico) y el campo *'**com**'* indica la comparación del caso.

La tabla *'sliq_nodes'* especifica las relaciones entre los nodos del árbol, y los valores útiles. Estos resultados esta en valores originales por lo cual no se debe hacer ninguna post transformación de las tablas.

2.7.6 Segundo ejemplo de la Función sliq(). Sobre la tabla *'Jugar_Tenis2'* (tabla 15) que para el ejemplo será la tabla *'kdd_ent'*. Al aplicar la función Sliq('jugar') sobre ésta tabla, se obtiene la tabla *'sliq_rules'* (tabla 25.) y la tabla *"sliq_nodes"* (tabla 26).

Tabla 25 sliq_rules 2

N	Id	Atributo	Valor	Com	1tipo
1	1	Estado	Nublado	=	2
1	1	Jugar	Si	=	2
2	1	Estado	Nublado	!=	2
2	1	Temperatura	Caliente	=	2
2	1	Jugar	No	=	2
3	1	Estado	Nublado	!=	2
3	1	Temperatura	Caliente	!=	2
3	1	Estado	Lluvioso	=	2
3	1	Viento	Debil	=	2
3	1	Jugar	Si	=	2
4	1	Estado	Nublado	!=	2
4	1	Temperatura	Caliente	!=	2
4	1	Estado	Lluvioso	=	2
4	1	Viento	Debil	!=	2
4	1	Jugar	No	=	2
5	1	Estado	Nublado	!=	2
5	1	Temperatura	Caliente	!=	2
5	1	Estado	Lluvioso	!=	2
5	1	Humedad	Normal	=	2
5	1	Jugar	Si	=	2
6	1	Estado	Nublado	!=	2
6	1	Temperatura	Caliente	!=	2
6	1	Estado	Lluvioso	!=	2
6	1	Humead	Normal	=	2
6	1	jugar	No	=	2

Tabla 26. sliq_nodes 2

Nodo	Padre	Hijo	Atributo	Valor	Com
1	-1	0	Estado	Nublado	=
2	1	1	Jugar	Si	=
3	1	2	Temperatura	Caliente	=
4	3	1	Jugar	No	=
5	3	2	Estado	Lluvioso	=
6	5	1	Viento	Debil	=
7	6	1	Jugar	Si	=
8	6	2	Jugar	No	=
9	5	2	Humedad	Normal	=
10	9	1	Jugar	Si	=
11	9	2	jugar	No	=

2.8 Implementación de las funciones para la fase de evaluación

Los archivos que contienen las fuentes de estas funciones están ubicadas en el directorio: *i /opt/Matekdd/functions/V*alid, y tienen como propósito probar, traducir e interpretar un modelo de clasificación y corresponden a:

- test_rules.sql: Contiene las fuentes de la función test_rules().
- test_sliq.sql: Contiene las fuentes de la función test_sliq().
- translate_rules.sql: Contiene las fuentes de la función translate_rules().
- translate_inte.sql: Contiene las fuentes de la función translate_inte()..
- arm_rules.sql: Contiene las fuentes de la función arm_rules().

Para la utilización de las funciones, se ingresa como usuario Postgres o equivalente, a la base de datos a través de la terminal interactiva psql. Una vez dentro se ejecuta la instrucción:

\i */opt/Matekdd/functions/V*alid */test_rules.sql*

\i */opt/Matekdd/functions/V*alid */test_sliq.sql*

\i */opt/Matekdd/functions/V*alid */translate_rules.sql*

\i */opt/Matekdd/functions/V*alid */translate_inte.sql*

\i */opt/Matekdd/functions/V*alid */arm_rules.sql*

Con estas instrucciones se crea las funciones, y obtiene el mensaje de confirmación 'CREATE FUNCTION' para cada una.

A partir de este momento ya es posible invocar a las funciones, haciendo uso de los parámetros requeridos.

2.8.1 Funcionamiento de la FDU test_rules(). La función *test_rules()* recibe como parámetro un tipo de dato VARCHAR que contiene el nombre del modelo ('c45'/'mate') a validar (solo se aplica a los modelos de los algoritmos C4.5 y Mate-Tree), y es la cargada de probar el modelo armando las reglas y consultado su coincidencia en la tabla de prueba *'kdd_pru'* y devolver el numero de incidencia de estas reglas en la tabla, es decir, el numero de tuplas o registros que fueron correctamente clasificados.

Retomando el ejemplo del modelo construido con el algoritmo C4.5 en donde se obtienen las tablas de reglas y de nodos *'c45_rules'* (tabla 18) y *'kdd_nodes'* (tabla

19) respectivamente. Si se ejecuta la función test_rules(), el resultado es el mostrado en la figura 4. (Este modelo fue probado con el mismo conjunto de datos que se uso para el entrenamiento).

Figura 4. Resultado de test_rules()

2.8.2 Funcionamiento de la FDU translate_rules(). La función *translate_rules()* recibe como parámetro un tipo de dato INTEGER que contiene el numero de rangos de utilizado para el proceso de discretización junto con un tipo de dato VARCHAR que contiene el nombre del modelo ('c45'/'mate') a traducir (solo se aplica a los modelos de los algoritmos C4.5 y Mate-Tree), y es la encargada de transformar a su estado inicial los valores de las tablas, utilizando las tablas *'kdd_values',* también llama a la función interpretar(), que utilizando la tabla *'kdd_rangos' ,* interpreta los valores anteriormente discretizados.

Continuando con el modelo C4.5 y junto con las tablas de rangos y valores de transformación *'pg_rangos'* de la tabla 6 y *'kdd_values'* de la tabla 17, si se aplica la función *translate_rules()* se obtiene como resultado las misma tabla *'c45_rules'* (tabla 18) y *'c45_nodes'* (tabla 19), pero traducidas a sus valores originales (tablas 27 y 28 respectivamente).

Tabla 27. c45_rules del modelo C4.5 traducida

Id	*Atributo*	*valor*
1	Estado	Soleado
1	Humedad	<= 80.5
1	Jugar	Si
2	Estado	Soleado
2	Humedad	>80.5
2	Jugar	No
3	Estado	Lluvioso
3	Viento	Debil
3	Jugar	Si
4	Estado	Lluvioso
4	Viento	Fuerte
4	Jugar	No
5	Estado	Nublado
5	Jugar	Si

Tabla 28. c45_nodes del modelo C4.5 traducida

Nodo	Padre	Atributo	Valor	Clase
0	-1	Estado	-1	-1
1	0	Estado	Soleado	-1
2	1	Humedad	<=80.5	Si
3	1	Humedad	>80.5	No
4	0	Estado	Lluvioso	-1
5	4	Viento	Debil	Si
6	4	Viento	Fuerte	No
7	0	Estado	Nublado	si

Si se aplica la función translate_rules() sobre la tabla “mate_rules” (tabla 21) del modelo Mate-Tree, se obtiene la tabla “mate_rules” (tabla 29) traducida a sus valores originales.

Tabla 29. Mate_rules del modelo Mate-Tree traducida

Id	Atributo	valor
1	Estado	Soleado
1	Humedad	<= 80.5
1	Jugar	Si
2	Estado	Soleado
2	Humedad	>80.5
2	Jugar	No
3	Estado	Lluvioso
3	Viento	Debil
3	Jugar	Si
4	Estado	Lluvioso
4	Viento	Fuerte
4	Jugar	No
5	Estado	Nublado
5	Jugar	Si

2.8.3 Funcionamiento de la FDU arm_rules(). La función *arm_rules()* recibe como parámetro un tipo de dato VARCHAR que contiene el nombre del atributo clase y junto con un tipo de dato VARCHAR que contiene el nombre del modelo (‘c45’/’mate’) de clasificación. Esta función arma cada una de las reglas como su nombre lo indica y devuelve reglas del tipo *condicion1 AND condicion2.. --> decisión*

Si se aplica la función *arm_rules()* a la tabla de reglas ‘c45_rules’ (tabla 29), se obtiene como resultado las reglas que se muestran en la Figura 5.

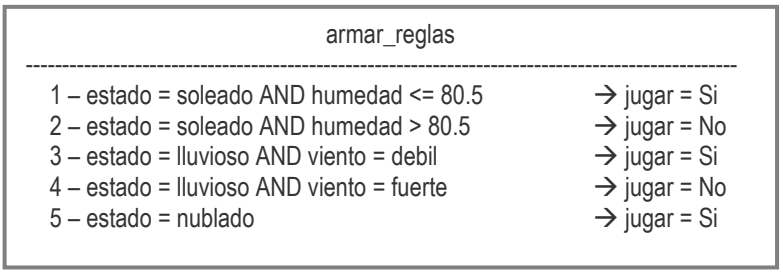


Figura 5. Reglas de Clasificación del modelo C4.5

Si se aplica la función *arm_rules()* a la tabla de reglas 'mate_rules' (tabla 29), se obtiene como resultado las reglas que se muestran en la figura 6.

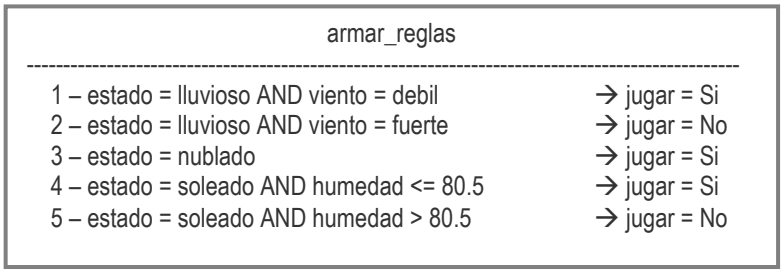


Figura 6. Reglas de Clasificación del modelo Mate-Tree

2.8.4 Funcionamiento de la FDU test_sliq(). La función *test_sliq()* recibe como parámetro un tipo de dato VARCHAR que contiene el nombre del atributo clase, y es la cargada de probar el modelo armando las reglas y consultado su coincidencia en la tabla de prueba *'kdd_pru'* y retorna las reglas del modelo junto con el numero de incidencia de estas reglas en la tabla, es decir, el numero de tuplas o registros que fueron correctamente clasificados.

Retomando el ejemplo del modelo construido con el algoritmo SLIQ en donde se obtuvieron las tablas de reglas y de nodos *'sliq_rules 1'* (tabla 23) y *'sliq_nodes 1'* (tabla 24) respectivamente. Si se ejecuta la función test_sliq() el resultado es el mostrado en la figura 7

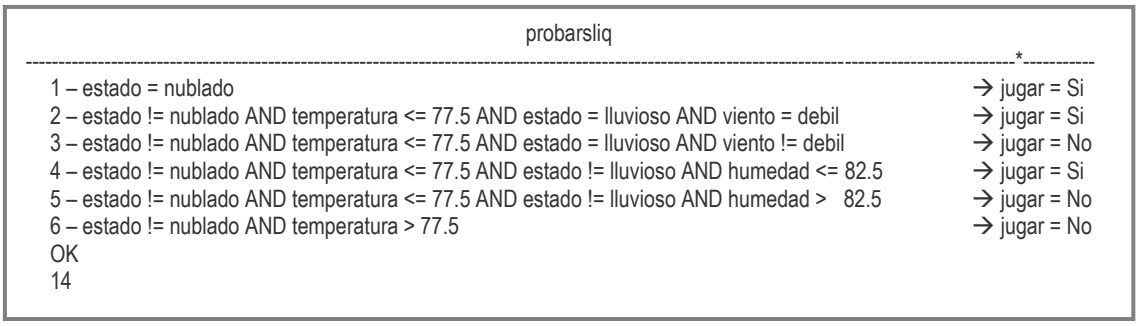


Figura 7. Resultado 1 de test_sliq()

Y si aplica la función *test_sliq()* a la tabla *'sliq_rules 2'* (tabla 25) el resultado es el mostrado en la figura 8.

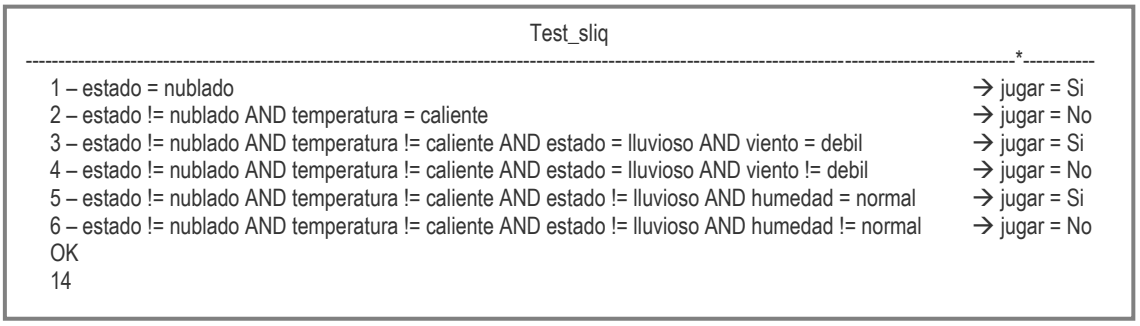
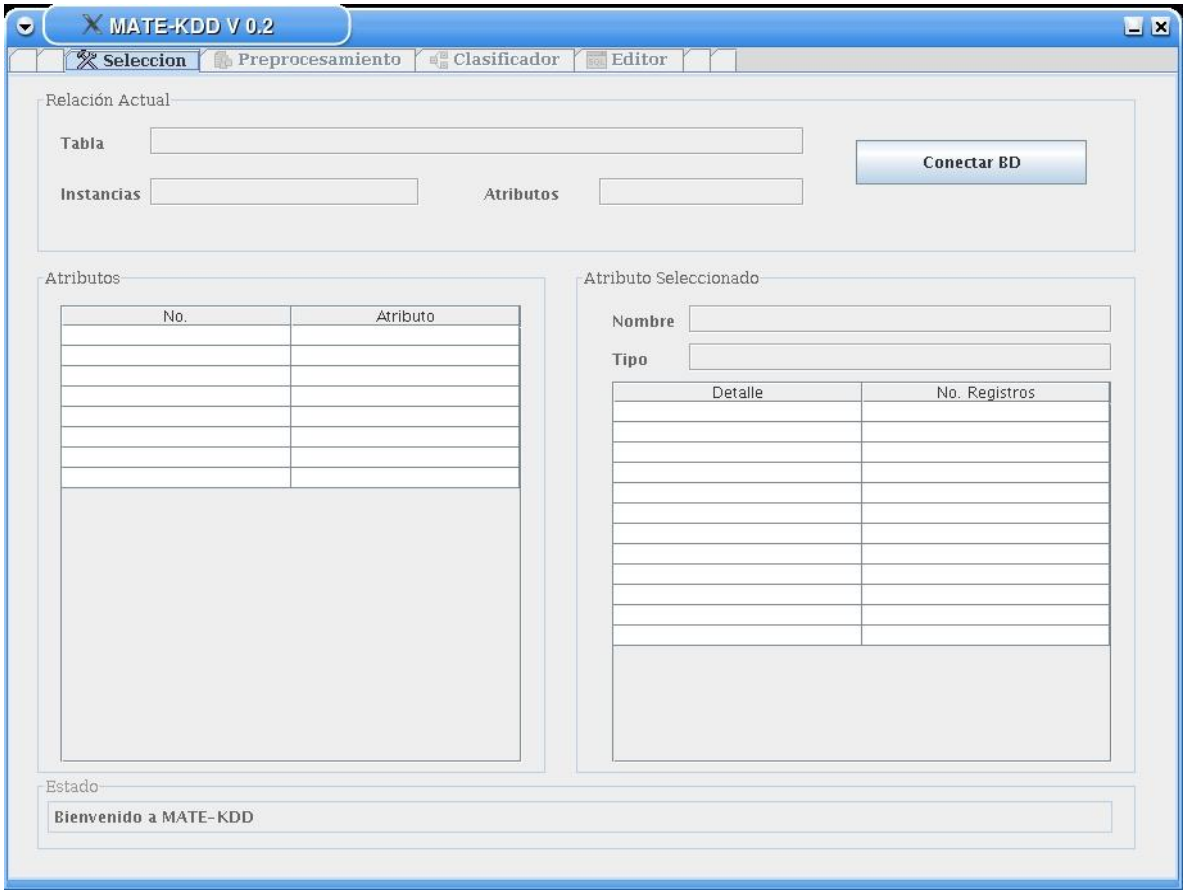


Figura 8. Resultado 2 de test_sliq()

3. AMBIENTE DEL PROGRAMA

Esta herramienta está diseñada para facilitar el proceso de la Minería de datos mediante algoritmos de clasificación como son SLIQ, C4.5 y MATE-TREE.

El diseño, es bastante amigable y predecible para una persona conocedora del proceso de Minería de Datos, ya que la aplicación organiza cada uno de los procesos que se siguen secuencialmente para minar una base de datos, para ello encontramos en una ventana diferente la aplicación de cada fase (selección de la BD, preprocesamiento, modelación con determinado algoritmo y resultados), interfaces fáciles de manejar y perfectamente validadas de tal forma que se puede llevar a cabo el proceso de minería de principio a fin sin dificultad.



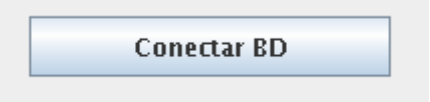
3.1 Pestañas del Programa

En esta parte de la pantalla se visualiza las funciones de la aplicación: Selección, Preprocesamiento, Clasificador y Editor. Cada pestaña permite que naveguemos por la aplicación y en cada item encontramos los pasos principales para minar, empezando por seleccionar la tabla que contiene los datos para procesar, manipulación de la misma, según criterio del analista con elfin de obtener resultados validos, aplicación del modelo para finalmente visualizar las reglas obtenidas y el árbol del modelo. Se implemento una opción importante que permite la interacción directa del analista con el SGBD PostgreSQL, este es el editor.

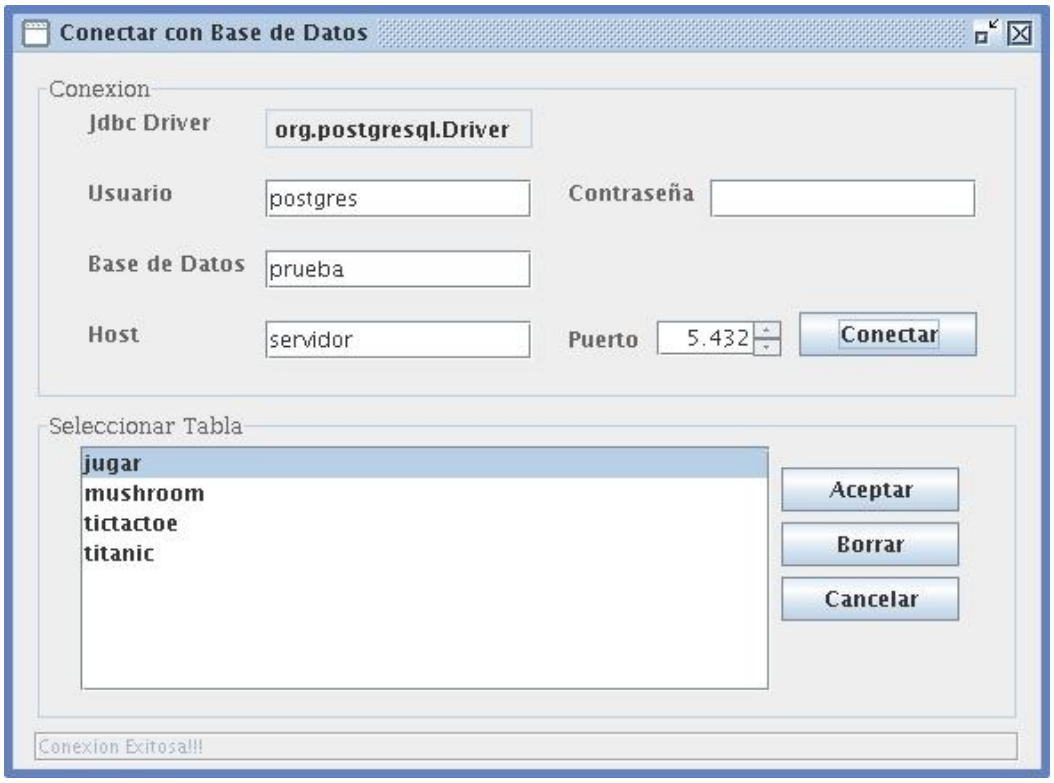
3.1.1 Opción de Selección

En la opción de Selección podemos conectarnos con el SGBD para ubicar la tabla que vamos a minar.

Conexión a la BD



Con este Botón podemos desplegar en la que podemos elegir la Base de Datos que se encuentre Creada en Postgres a la cual accedemos con un nombre de Usuario y una contraseña, el nombre de la base de datos y Host, una vez seguidos estos pasos en el recuadro blanco Rotulado como “Seleccionar Tabla” se visualizan la tablas que tiene creada la base de datos y seleccionamos la deseada para trabajar. También permite borrar tablas que no queramos o ya no necesitemos de nuestra Base de Datos.



Simplemente haciendo clic sobre el botón *“Aceptar”* obtendremos una descripción detallada de la tabla, sus atributos, tipos de datos. Tenemos dos secciones en la ventana principal en la que podemos visualizar la información de los datos con el

fin de conocer sus características para empezar a trabajar y determinar si vamos a aplicar algún cambio en la opción de preprocesamiento.

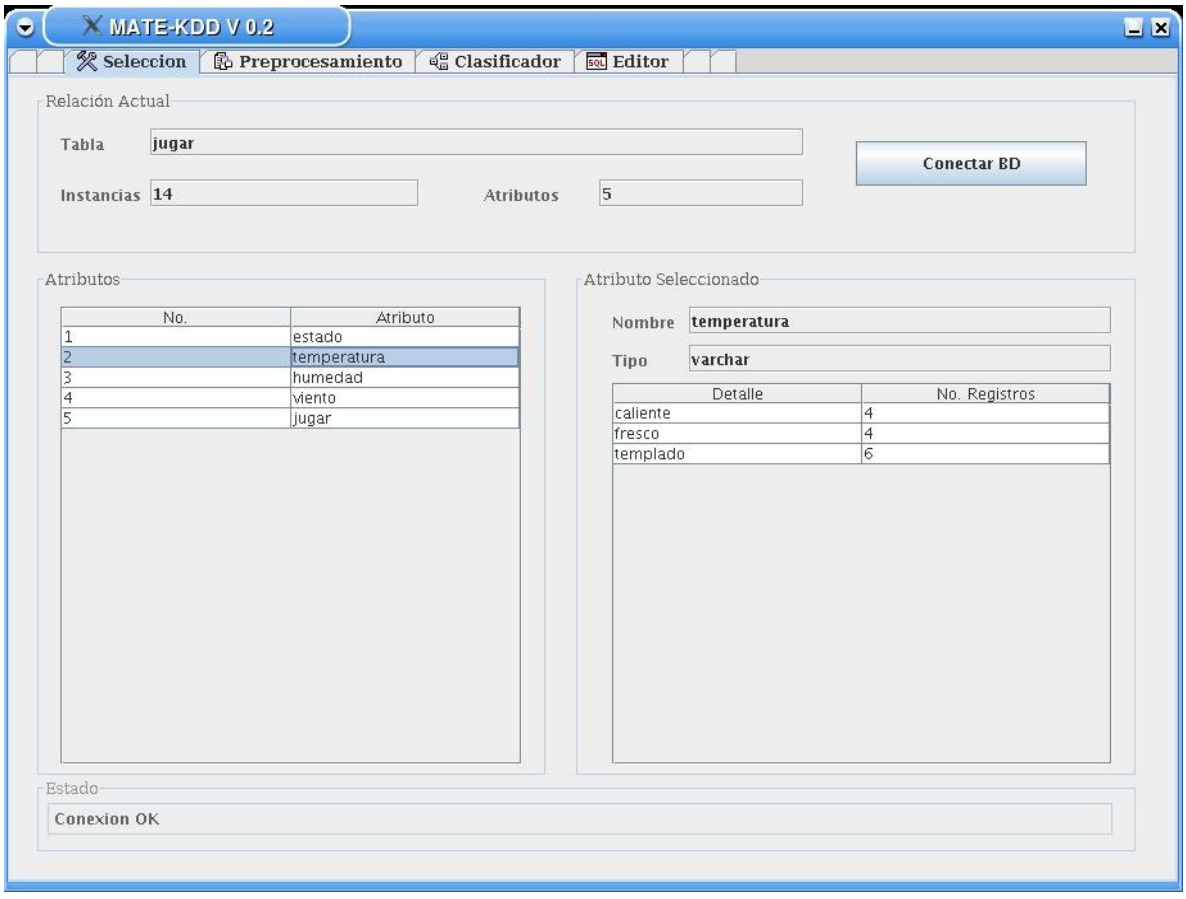
En el área de trabajo podemos diferenciar la siguiente información:

Relación Actual Muestra el nombre de la tabla, así como el número de atributos e instancias que la componen. También esta los botones para abrir o guardar una tabla.

Atributos Aquí se muestran los nombres de los atributos de la tabla.

Atributo Seleccionado Al seleccionar con un clic un atributo de *"Atributos"* en esta parte se muestra las características de ese atributo.

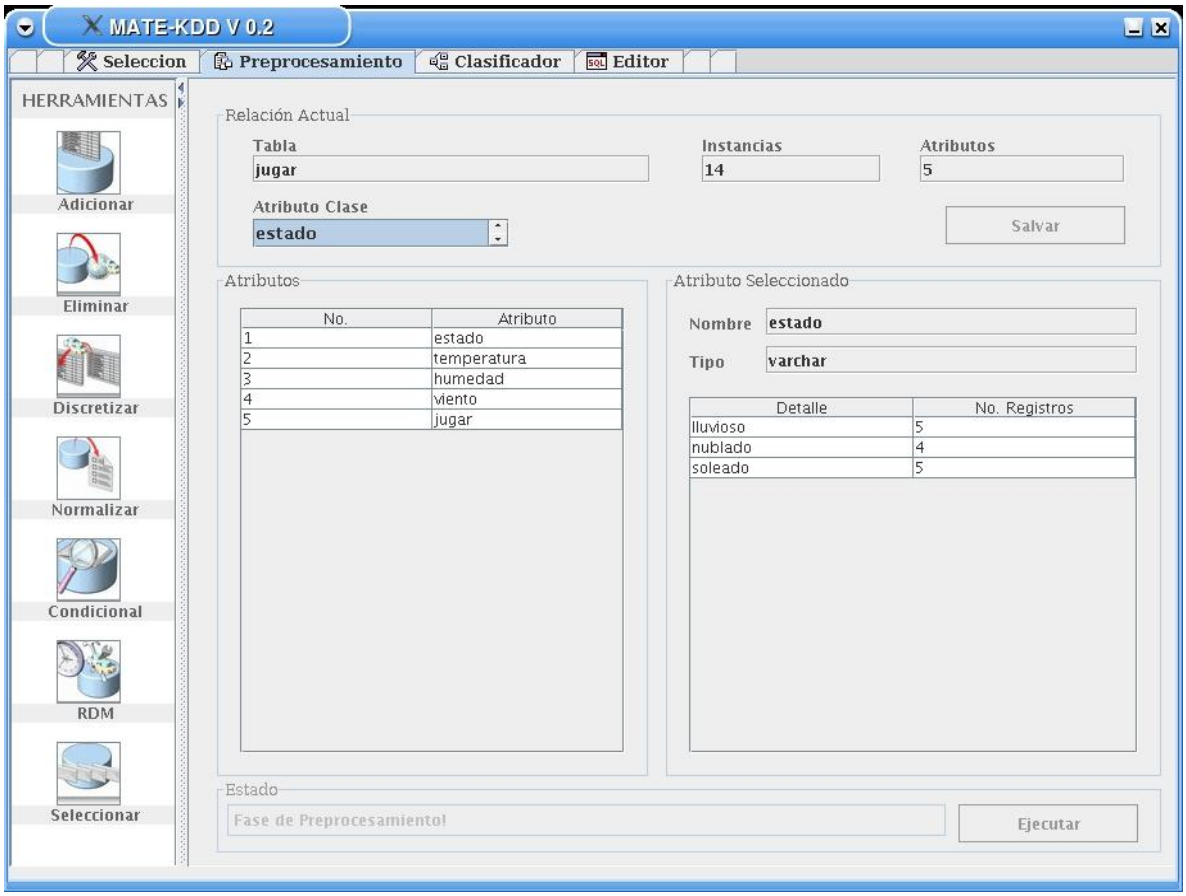
Podemos observar que en la ventana se detalla la tabla seleccionada, sus atributos, tipos de datos, cantidades.



3.1.2 Opción Preprocesamiento

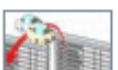
Mate-kdd permite aplicar en la etapa de preprocesamiento algunas técnicas de depuración y discretización que son indispensables para poder procesar los datos y de esta forma obtener unos patrones confiables.

Se cuenta con algunos filtros que permiten manipular la tabla a criterio del analista para depurar datos ruidosos, Discretizar atributos numéricos que es muy importante según el algoritmo que deseemos aplicar, esto tonel fin de realizar transformaciones a partir de la tabla original de la forma en que el analista vea pertinente, una vez se aplica cualquiera de los filtros se crea una tabla con la nueva configuración a la que podemos dar un nombre. Es en esta fase en donde se debe seleccionar el atributo clase.



Adicionar: Esta herramienta permite adicionar un atributo a la tabla, éste puede ser el resultado de una operación matemática entre dos atributos de la misma o un atributo con un número.

Normalizar: Esta herramienta permite normalizar los datos numéricos que se encuentran en base 10 a rangos entre 0 y 1.



Discretizar: Con esta Herramienta podemos Discretizar un conjunto de

Condicional: mediante esta herramienta podemos filtrar la tabla con una condición booleana (***or*** o ***and***).

RDM: Nos permite seleccionar randomicamente un número de datos de la tabla original, el analista especifica cuantos datos quiere

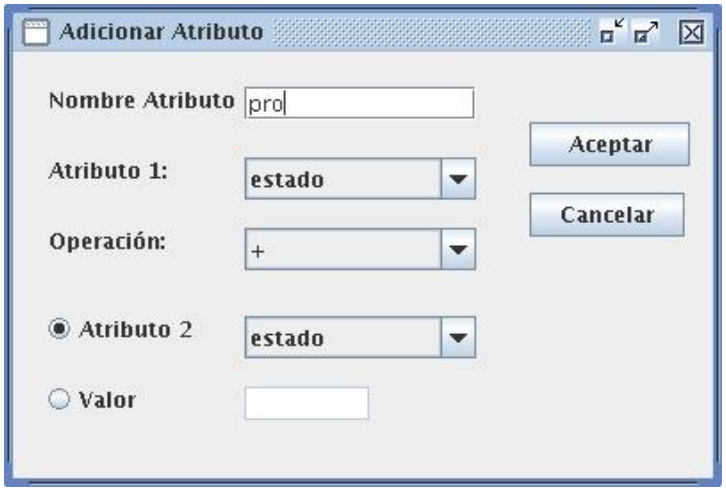
Eliminar: Podemos también eliminar un atributo de la tabla que vamos a minar cuando seleccionamos esta Herramienta.

Seleccionar: Permite seleccionar determinados atributos de una tabla para trabajar con ella.

Cuando tenemos seleccionada la tabla para minar podemos entonces aplicar alguno de los filtros antes mencionados dando clic en el botón “Ejecutar”.

Adicionar

En esta ventana podemos dar un nombre al atributo que vamos a adicionar, seleccionamos el primer atributo la operación matemática que aplicaremos y el segundo atributo o un valor para efectuar el proceso. Al dar clic en el botón “Aceptar”, se construye la instrucción SQL para su posterior ejecución.



Eliminar

En la ventana de Eliminar atributo podemos seleccionar el atributo que deseamos eliminar de la tabla Al dar clic en el botón “Aceptar”, se construye la instrucción SQL para su posterior ejecución.

Discretizar

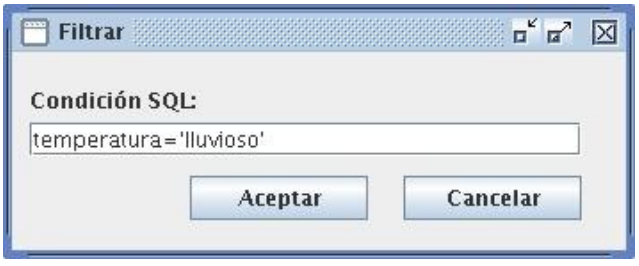
Para Discretizar una tabla o un atributo debemos tener en cuanta que los atributos deben ser numéricos para discretizarse; introducimos el número de rangos que deseamos para formar los grupos de datos que no puede ser mayor que 10, seleccionamos “Discretizar Tabla” o “Discretizar Atributo” según necesitemos y aplicamos el proceso.

Normalizar

Para Normalizar una tabla o un atributo procedemos igual que para Discretizar, seleccionamos si el proceso se va a realizar sobre la tabla o sobre un atributo para seleccionar.

Condicional

Esta opción nos permite hacer un filtro muy especial, podemos condicionar un atributo mediante una condición lógica.



Seleccionar

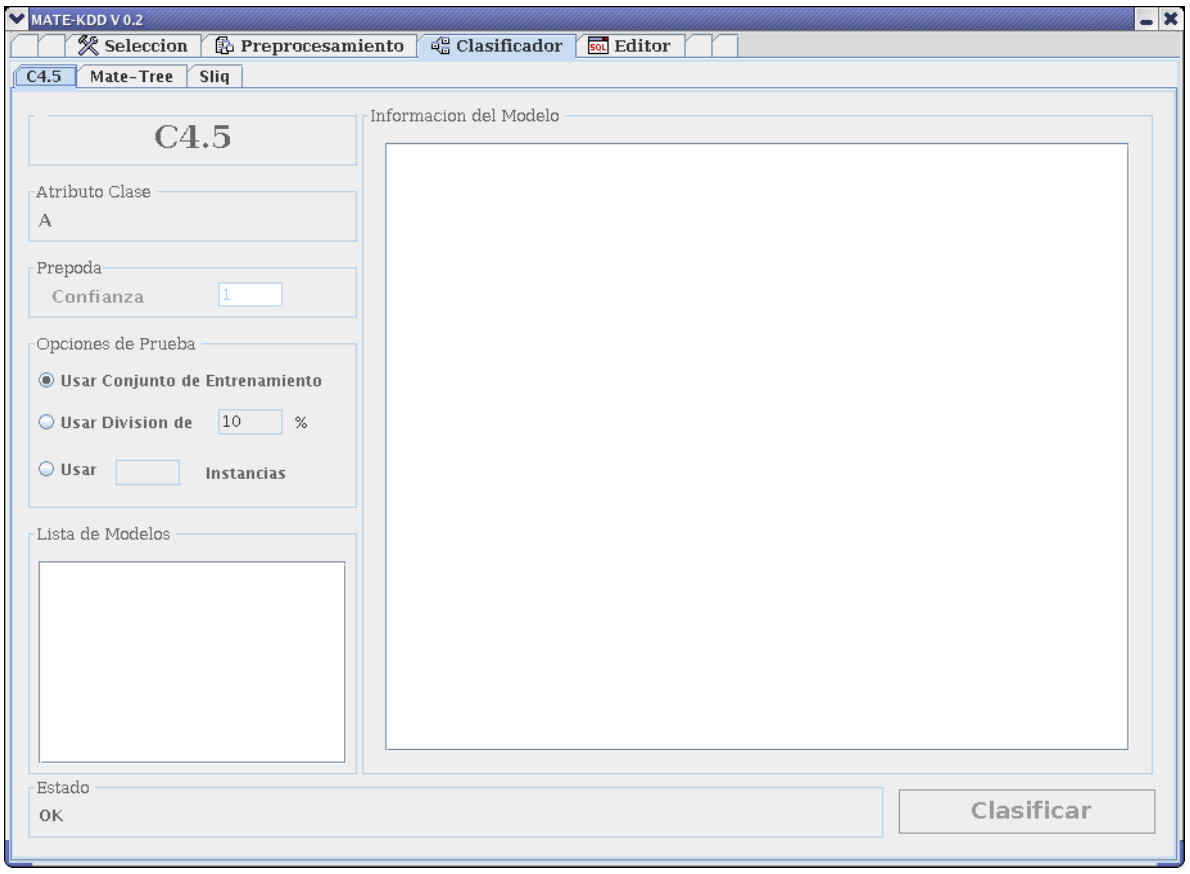
Esta opción permite escoger los atributos que el analista desee de una tabla para clasificar solo con ellos.

3.1.3 Opción Clasificador

Aquí tenemos la posibilidad de seleccionar el algoritmo C4.5, Mate-Tree o SLIQ para aplicarlo.

C4.5

Al seleccionar la pestaña C4.5 aparece esta pantalla en donde se presenta el proceso de clasificación utilizando el algoritmo C4.5.



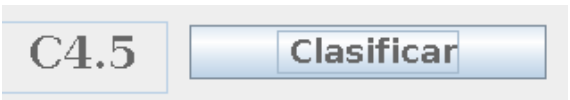
Cuando nos disponemos a modelar con cualquiera de los tres algoritmos implementados podemos configurar el los conjuntos para entrenamiento y prueba del modelo:

- *Usar conjunto de entrenamiento:* De esta forma se utiliza todo el conjunto de datos para la construcción y para la prueba del modelo.

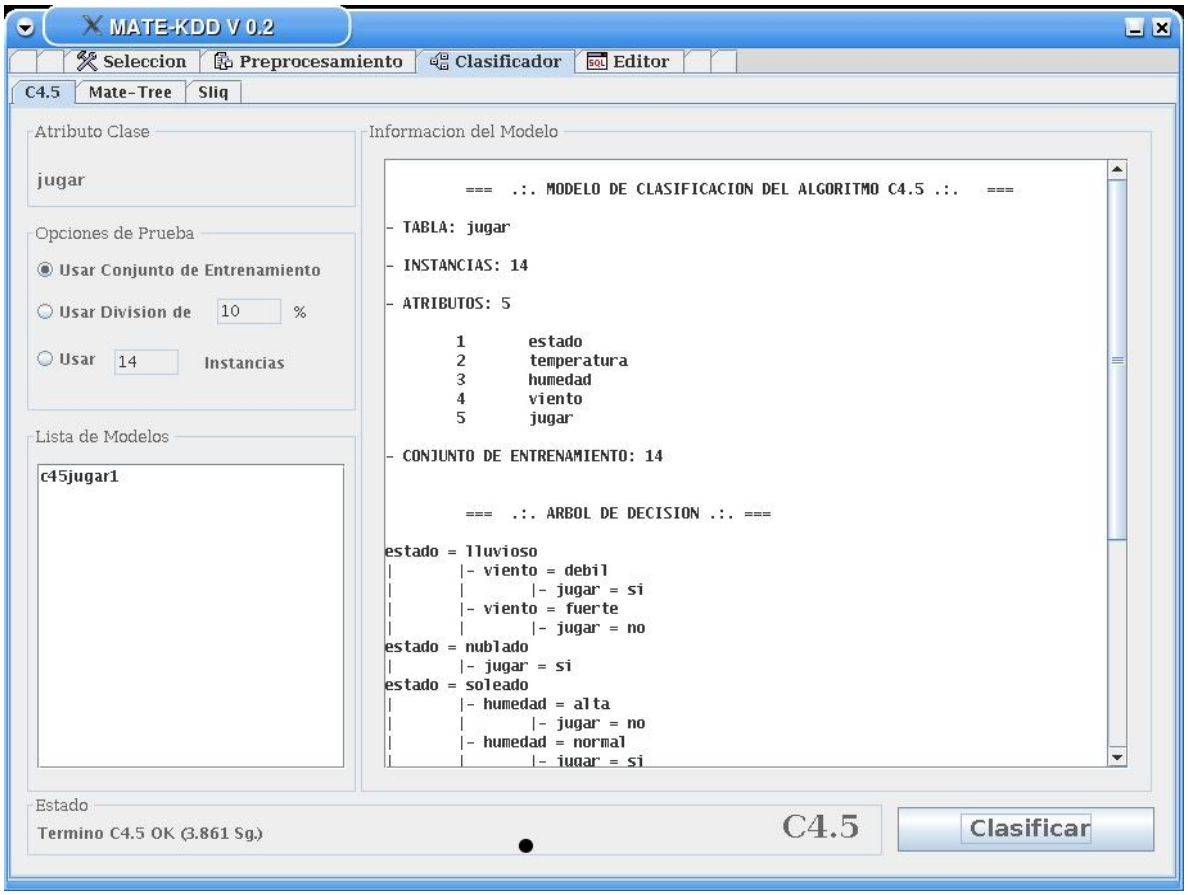
- *Usar Division de X %:* De esta forma se divide el conjunto de datos para entrenamiento y prueba.

- *Usar X Instancias:* De esta forma se define cuantos registros van a componer el conjunto de datos para la prueba del modelo.

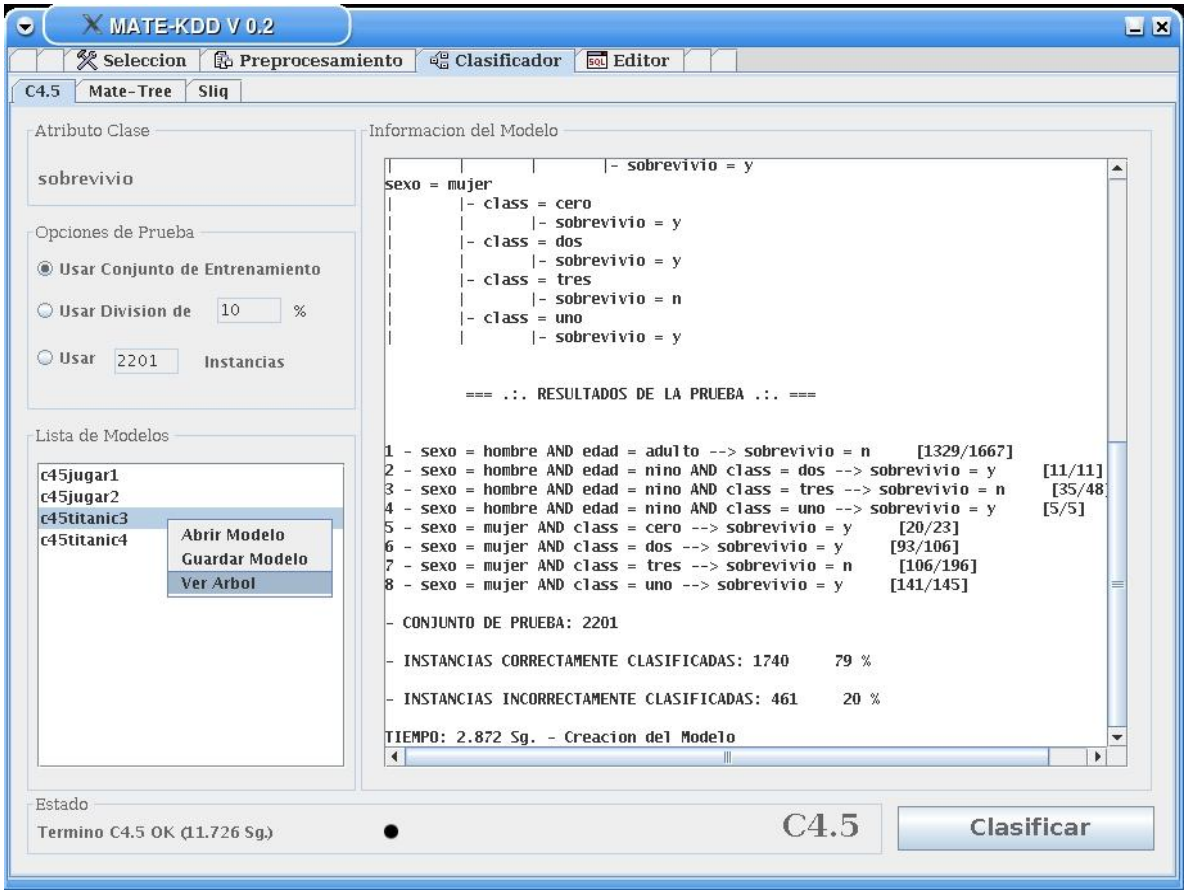
Una vez se haya configurado la opción de prueba, se inicia el modelaje con clic sobre el botón “Clasificar”.



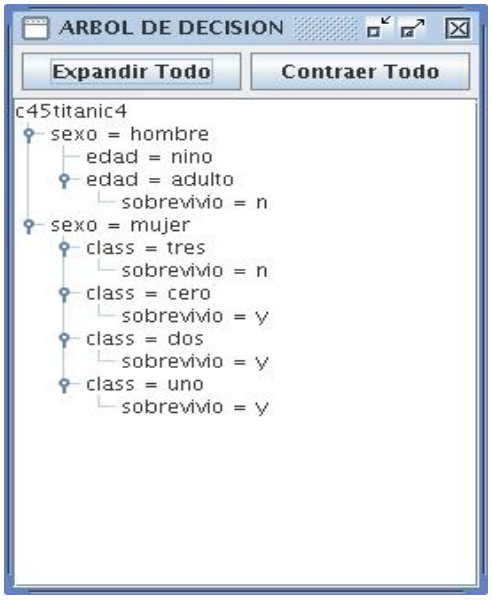
El modelo creado se visualiza en el recuadro blanco al lado derecho de la pantalla, y en el recuadro pequeño al lado izquierdo de la ventana se muestra en detalle el modelo.



Podemos guardar el modelo dando clic derecho sobre el nombre que aparece por defecto en el recuadro “Lista de modelos”.



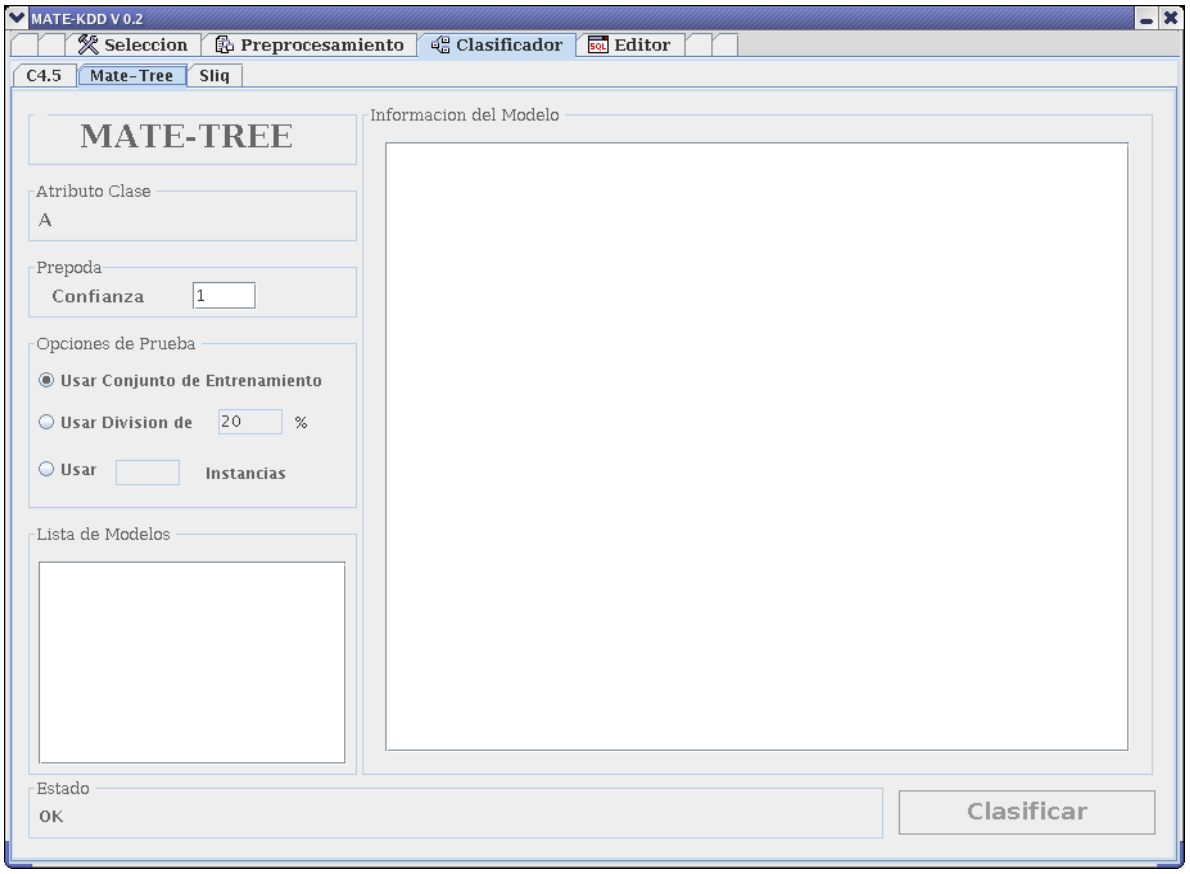
Otra opción es ver árbol, cuando la seleccionamos podemos visualizar en la pantalla el árbol del modelo de clasificación.



La opción “Abrir Modelo” permite escoger y visualizar en la pantalla un modelo que ya se haya creado y guardado anteriormente.

Mate-Tree

En la pestaña Mate-Tree tenemos que configurar la tabla para clasificar los datos para entrenamiento y prueba del modelo. Para aplicar el modelo los datos deben estar discretizados en caso de que hallan atributos numéricos, con el botón *“clasificar”* aplicamos el algoritmo.

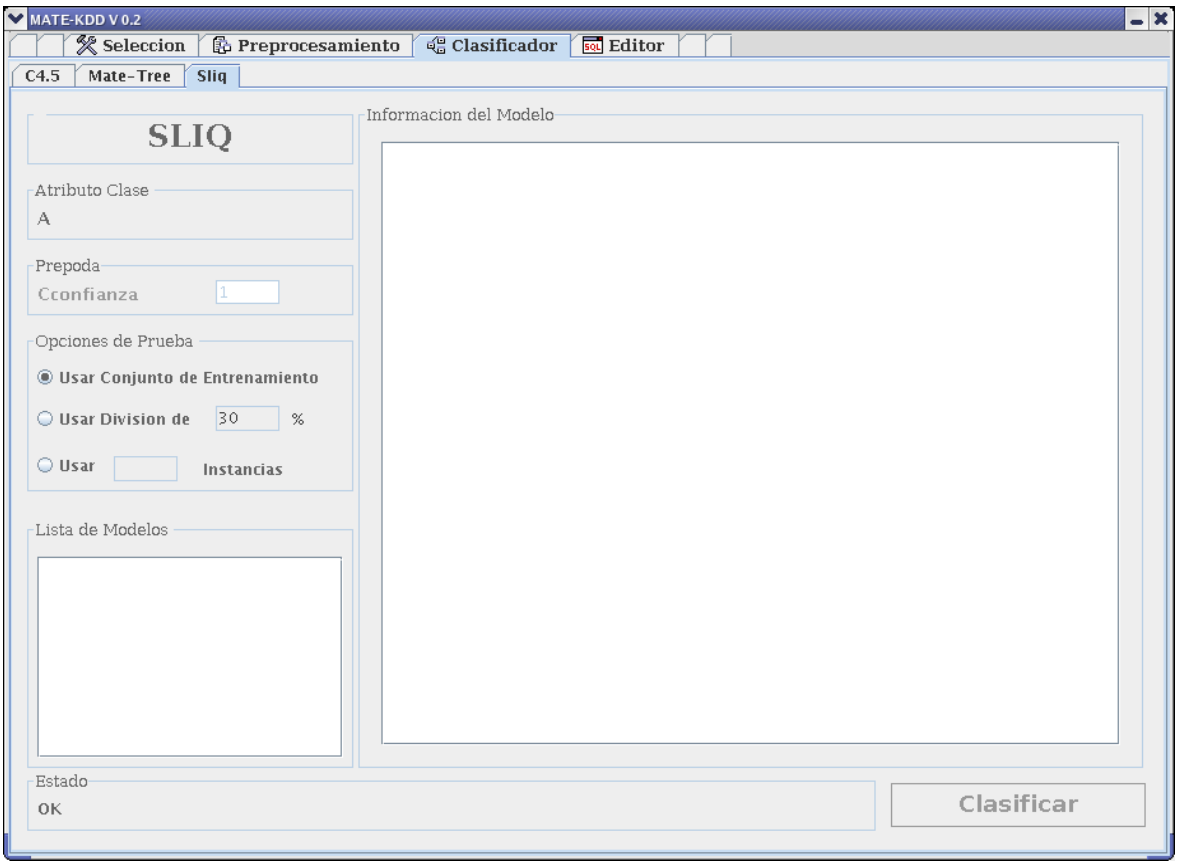


Podemos guardar el modelo, visualizar el árbol y si ya tenemos modelos guardados anteriormente podemos abrirlos y revisar.

SLIQ

Al seleccionar la pestaña Sliq nos encontramos con la pantalla en donde se presentan los diferentes procesos que permiten el proceso clasificación utilizando el algoritmo Sliq.

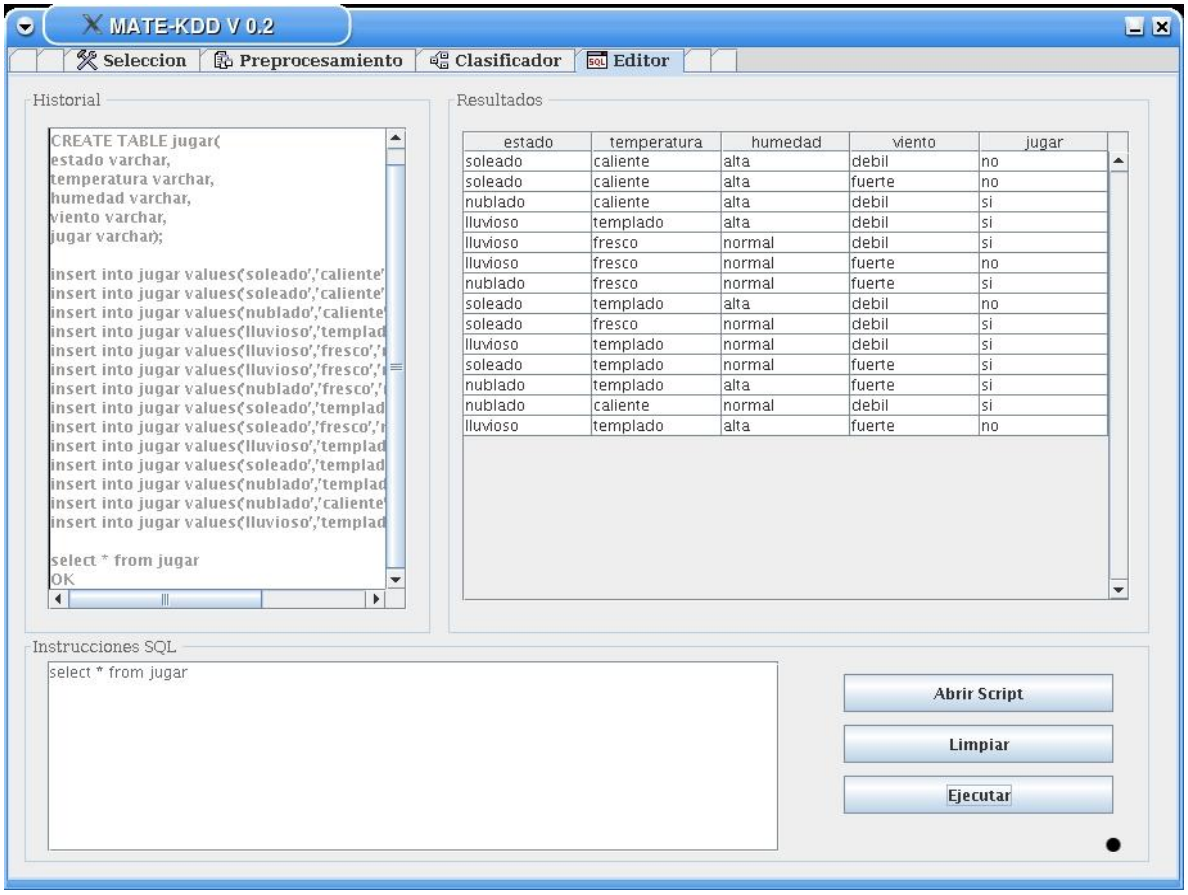
Para este algoritmo no es necesario realizar una discretización previa de los atributos numéricos a menos que el analista lo considere necesario, ya que Sliq trabaja con atributos numéricos, Y se da clic en el botón “Clasificar” para inicial el modelaje.



Los modelos creados los podemos guardar, abrir y mirar el árbol de cada modelo, igual que en los anteriores algoritmos, esto con el fin de visualizar los diferentes modelos resultado de aplicar cada algoritmo a una tabla en particular.

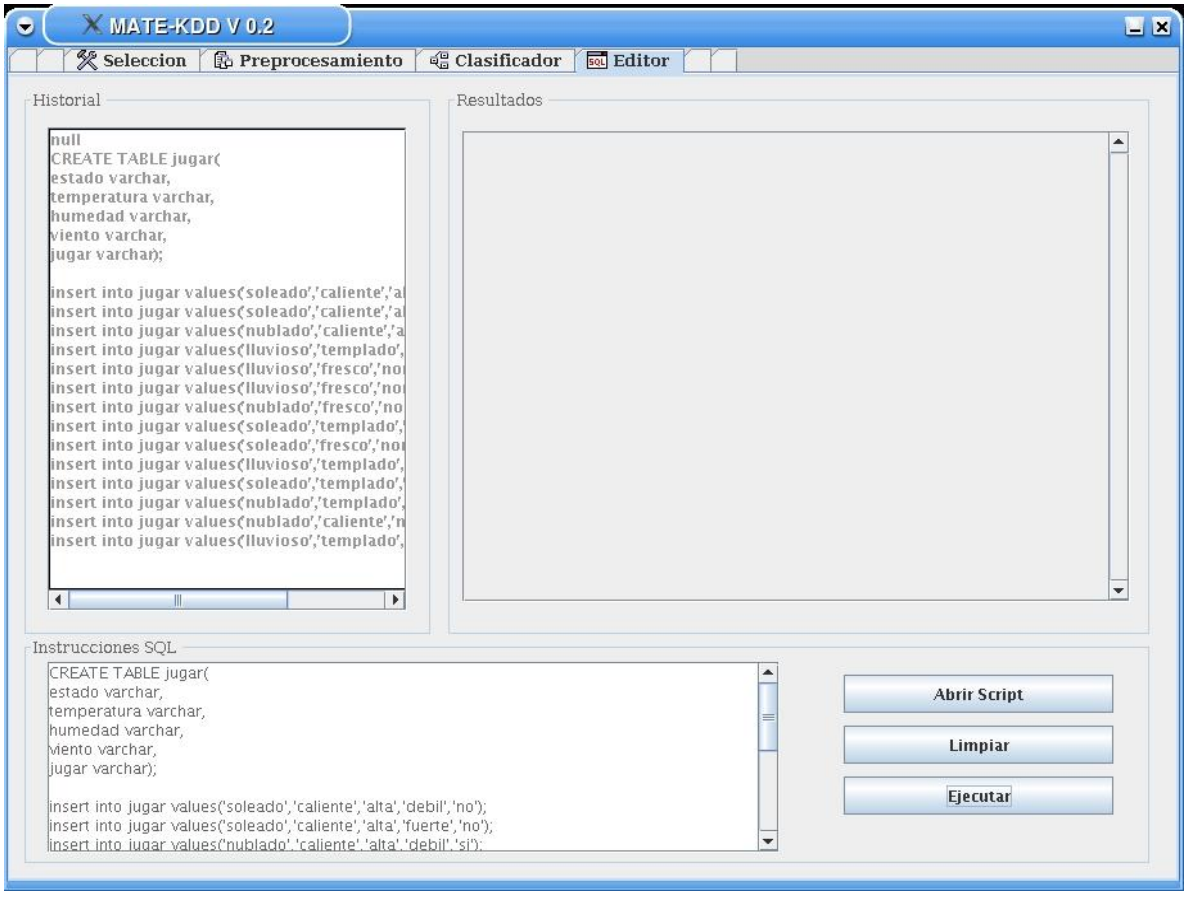
3.1.4 Editor

El editor es una interfaz que nos permite ejecutar sentencias SQL en el SGBD Postgresql, y tiene un área en la que visualizamos el resultado de la instrucción si es una consulta.



Otra opción muy útil en el editor es la de abrir y ejecutar Scripts, así que no se tiene que ir a la Base de datos Postgres pues desde la aplicación podemos crear tablas y cargar datos para comenzar a utilizar la aplicaron.


Abrir
Buscar en:
scripts
apariencia.sql
apariencia2.sql
bd2.sql
bd2.sql~
clases.sql
hepatitis.sql
hepatitis.sql~
hepatitis2.sql
jugartenis.sql
tablatitanic.sql
Nombre de archivo:
jugartenis.sql
Archivos de tipo:
Todos los archivos
Abrir
Cancelar


MATE-KDD V 0.2
Seleccion
Preprocesamiento
Clasificador
Editor
Historial
null
CREATE TABLE jugar(
estado varchar,
temperatura varchar,
humedad varchar,
viento varchar,
jugar varchar);
Resultados
Instrucciones SQL
CREATE TABLE jugar(
estado varchar,
temperatura varchar,
humedad varchar,
viento varchar,
jugar varchar);
insert into jugar values('soleado','caliente','alta','debil','no');
insert into jugar values('soleado','caliente','alta','fuerte','no');
Abrir Script
Limpiar
Ejecutar

Como miramos a lo largo de este manual, la aplicación es fácil de utilizar debido a su interfaz amigable, podemos seguir cada paso del proceso de Minería en detalle y hacer una seguimiento hasta el final, además que con el editor podemos interactuar con el SGBD desde la aplicación sin tener que ir a Postgres si tenemos los scripts de las tablas que queremos minar.