# Árboles de Decisión para Grandes Conjuntos de Datos

Anilu Franco-Arcega, J. Ariel Carrasco-Ochoa,
Guillermo Sánchez-Díaz, J. Francisco Martínez-Trinidad

Reporte Técnico No. CCC-08-001
13 de Febrero de 2008


Coordinación de Ciencias Computacionales
INAOE

Luis Enrique Erro 1
Sta. Ma. Tonantzintla,
72840, Puebla, México.

# Árboles de Decisión para Grandes Conjuntos de Datos

Anilu Franco Arcega[1], Jesús Ariel Carrasco Ochoa[2], Guillermo Sánchez Díaz[3],
José Francisco Martínez Trinidad[4]

Coordinación de Ciencias Computacionales
Instituto Nacional de Astrofísica, Óptica y Electrónica
Luis Enrique Erro # 1, Santa María Tonantzintla, Puebla, 72840, México
[1,2,4]{anifranco6,ariel,fmartine}@inaoep.mx

Tecnologías de Información
Universidad Politécnica de Victoria
Calzada General Luis Caballero No. 1200, Cd. Victoria, Tamaulipas, 87070, México
[3]gsanchezdiaz@yahoo.es

**Resumen.** Esta propuesta de tesis doctoral aborda el problema de generación de árboles de decisión para grandes conjuntos de datos. Como resultado preliminar se propone un algoritmo incremental para construir árboles de decisión multivaluados para grandes conjuntos de datos numéricos que procese uno a uno los objetos del conjunto de entrenamiento. Los resultados obtenidos muestran que el algoritmo propuesto es competitivo en calidad y más rápido que el algoritmo C4.5. Además, se comparó con ICE que es un algoritmo de generación de árboles de decisión para grandes conjuntos de datos.

**Palabras Clave.** Árboles de decisión, clasificación supervisada, grandes conjuntos de datos.

## 1. Introducción

Dentro del Reconocimiento de Patrones uno de los problemas más estudiados es el de Clasificación Supervisada, en donde se conoce que un universo de objetos se agrupa en un número dado de clases de las cuales se tiene de cada una, una muestra de objetos que se sabe pertenecen a ella y el problema consiste en dado un nuevo objeto poder establecer sus relaciones con cada una de dichas clases [1].

Los algoritmos de clasificación supervisada tienen como objetivo determinar la pertenencia de un objeto (descrito por un conjunto de atributos) a una o varias clases, basándose en la información contenida en un conjunto de objetos previamente clasificados (conjunto de entrenamiento - $CE$).

Dentro de los algoritmos utilizados para resolver problemas de clasificación supervisada se encuentran los árboles de decisión. Un árbol de decisión es una estructura que se compone de nodos (internos y hojas) y de arcos. Sus nodos internos están caracterizados por uno o varios atributos de prueba y de estos nodos se desprenden uno o más arcos. Cada uno de estos arcos tiene asociado un valor del atributo de prueba y estos valores determinan qué camino seguir en el recorrido del árbol. Los nodos hoja contienen información

que permite determinar la pertenencia del objeto a una clase. Las caraterísticas principales de un árbol de decisión son: construcción sencilla, no necesita determinar de antemano parámetros para su construcción, puede tratar problemas multi-clase de la misma forma en que trabaja con problemas de dos clases, facilidad para ser representado mediante un conjunto de reglas y la fácil interpretación de su estructura.

Existen diversas clasificaciones de los árboles de decisión, por ejemplo de acuerdo al número de atributos de prueba en sus nodos internos existen 2 tipos de árboles:

- Univaluados, sólo contienen un atributo de prueba en cada nodo. Ejemplos de estos algoritmos son: ID3 [2], C4.5 [3], CART [4], FACT [5], QUEST [6], Model Trees [7], CTC [8], ID5R [9], ITI [10] [11], UFFT [12] [13], StreamTree [14], FDT [15], G-DT [16] y SPIDA [17].
- Multivaluados, que poseen a un subconjunto de atributos en cada uno de sus nodos. Por ejemplo, PT2 [18], LMDT [19] [20], GALE [21] y C-DT [22] [23].

De acuerdo al tipo de decisión tomada por el árbol, hay dos tipos de árboles:

- Difusos, dan un grado de pertenencia a cada clase del conjunto de datos, como por ejemplo, C-DT, FDT, G-DT y SPIDA.
- Duros, asignan la pertenencia del objeto a solamente una clase, así el objeto pertenece o no pertenece a una clase, ejemplos de estos algoritmos son: ID3, C4.5, CART, FACT, QUEST, Model Trees, CTC, LMDT, GALE, ID5R, ITI, UFFT, PT2 y StreamTree.

Los algoritmos de generación de árboles de decisión pueden ser clasificados de acuerdo a su capacidad de procesar conjuntos de datos dinámicos, es decir, conjuntos en los cuales se permite agregar nuevos objetos. De acuerdo a esto existen 2 tipos de algoritmos de generación de árboles de decisión:

- Incrementales, pueden procesar conjuntos de datos dinámicos de los cuales van obteniendo una solución parcial conforme se van analizando los objetos. Algunos ejemplos de este tipo de algoritmos son: ID5R, ITI, UFFT, PT2 y StreamTree.
- No incrementales, sólo pueden trabajar sobre conjuntos de datos estáticos, ya que para obtener la solución necesitan al conjunto de datos en su totalidad. Ejemplos de ellos son: ID3, C4.5, CART, FACT, QUEST, Model Trees, CTC, FDT, G-DT, SPIDA, LMDT, GALE y C-DT.

Recientemente, se ha abordado el problema de generar árboles de decisión para grandes conjuntos de datos. Estos conjuntos cuentan con una gran cantidad de objetos y en la mayoria de los casos no caben en la memoria disponible, con lo cual la aplicación de los algoritmos mencionados anteriormente resulta en un alto costo computacional, tanto en tiempo como en espacio.

Por esa razón se han desarrollado algoritmos de generación de árboles de decisión capaces de trabajar con grandes conjuntos de datos, sin tener que mantener en memoria todo el conjunto de datos necesario para generar el árbol de decisión. Ejemplos de algoritmos de este tipo son: SLIQ [24], SPRINT [25], CLOUDS [26], RainForest [27], BOAT [28], ICE [29] y PUDT [30]. La problemática principal atacada por estos algoritmos es precisamente poder manipular todo el conjunto de datos, a fin de construir un árbol de decisión que permita determinar la pertenencia de nuevos objetos a alguna clase. Sin embargo, algunos algoritmos mencionados continúan presentando restricciones en cuanto al manejo de la memoria, por la representación que usan para el conjunto de datos (como SLIQ, SPRINT y CLOUDS), o bien algunos otros no procesan todo el conjunto de datos al generar el árbol de decisión (como BOAT e ICE).

Otro problema en algoritmos para generar árboles de decisión que procesan grandes conjuntos de datos, es el número de veces que recorren el conjunto de datos completo, algunos de ellos (como RainForest) accesan al conjunto hasta dos veces en cada nivel del árbol, con lo que resulta en un mayor tiempo de procesamiento.

De lo anterior, surge la motivación de proponer un algoritmo para generar árboles de decisión, capaz de procesar grandes conjuntos de datos, posiblemente dinámicos, que procese todo el conjunto de datos disponible en un tiempo menor al de los algoritmos existentes.

## 2. Trabajo Relacionado

Esta sección está divida en dos partes, primero se presentan algunos conceptos básicos utilizados a lo largo de este documento. Posteriormente se describen los trabajos relacionados con la línea de investigación de algoritmos que generan Árboles de Decisión para grandes conjuntos de datos.

### 2.1. Conceptos Básicos

Un Árbol de Decisión (AD) es un grafo acíclico dirigido, compuesto de un nodo llamado *raíz*, el cual no tiene arcos de entrada, y de un conjunto de nodos que poseen un arco de entrada. Aquellos nodos con arcos de salida son llamados *nodos internos* o *nodos de prueba* y los que no tienen arcos de salida son conocidos como *hojas, nodos terminales o nodos de decisión* [31].

Los principales objetivos que se persiguen al crear un AD [32] son:

- Clasificar correctamente la mayor cantidad de objetos del CE.
- Generalizar, durante la construcción del árbol, el CE a fin de que nuevos objetos sean clasificados con el mayor porcentaje de aciertos posible.
- Si el conjunto de datos es dinámico, la estructura del AD debe poder actualizarse fácilmente.

Un algoritmo de generación de árboles de decisión consta de 2 etapas: la primera es la etapa de inducción del árbol y la segunda es la etapa de clasificación. En la primer etapa se construye el árbol de decisión a partir del conjunto de entrenamiento, comúnmente cada nodo interno del árbol se compone de un atributo de prueba y la porción del conjunto de entrenamiento presente en el nodo es dividida de acuerdo a los valores que pueda tomar ese atributo. La construcción del árbol inicia generando su nodo raíz, eligiendo un atributo de prueba y particionando el conjunto de entrenamiento en dos o más subconjuntos, para cada partición se genera un nuevo nodo y así suvesivamente. Cuando en un nodo se tienen objetos de más de una clase se genera un nodo interno, cuando contiene objetos de una clase solamente, se forma una hoja a la cual se le asigna la etiqueta de la clase. En la segunda etapa del algoritmo, cada objeto nuevo es clasificado por el árbol construido, se recorre el árbol desde el nodo raíz hasta una hoja, a partir de la cual se determina la pertenencia del objeto a alguna clase. El camino a seguir en el árbol lo determinan las decisiones tomadas en cada nodo interno, de acuerdo al atributo de prueba presente en él.

Existen diversos criterios de selección de atributos que se pueden utilizar para determinar el o los atributos para caracterizar a los nodos internos del AD. Entre los criterios más usados se pueden encontrar:

- Ganancia de Información [2]. La Ganancia de Información se define a partir de la Entropía. Dada una colección $S$ de objetos, con $c$ clases, la entropía de $S$ se mide como:

$$Entropia(S) = \sum_{i=1}^{c} -p_i \log_2 p_i \tag{1}$$

Donde $p_i$ es la proporción de ejemplos en $S$ que pertenencen a la clase $i$. Usando la entropía, se define la Ganancia de Información, de un atributo $X$ en un conjunto de datos $S$, como:

$$Ganancia(S, X) = Entropia(S) - \sum_{v \in Valores(X)} \frac{|S_v|}{|S|} Entropia(S_v) \tag{2}$$

Donde $Valores(X)$ es el conjunto de todos los posibles valores que puede tomar el atributo $X$ y $S_v$ es el subconjunto de $S$, donde los objetos toman el valor $v$ en el atributo $X$. El atributo a elegir será aquél que proporcione el mayor valor de Ganancia.

- Proporción de la ganancia de Información [3]. Utiliza la medida anterior (Ganancia(S,X)) pero se le aplica una normalización, la cual ajusta la ganancia obtenida por cada atributo. La Proporción de la Ganancia de Información, utilizando 2, queda expresada como:

$$Gain(S, X) = Ganancia(S/X) \Bigg/ \left( - \sum_{v \in Valores(X)} \frac{|S_v|}{|S|} \log_2 \left( \frac{|S_v|}{|S|} \right) \right) \tag{3}$$

  El atributo a elegir será aquél que maximice 3.

- Índice de diversidad de Gini [33]. Esta medida se calcula para cada dupla atributo-valor. Sea $CE_x$ el CE descrito solamente por el atributo que será evaluado, $CE_x$ descrito por $c$ clases, el índice de Gini se define como:

$$I_{gini}(CE_x) = 1 - \sum_{j=1}^{c} p_j(X, v)^2 \tag{4}$$

  Donde $p_j$ es el número de objetos de la clase $j$ que cumplen con la condición del valor $v$ en el atributo $X$ dividido entre el número de objetos de la clase $j$. La dupla atributo-valor elegida para caracterizar a los nodos del árbol, será aquella que presente menor valor con el Índice de Gini.

En la mayoría de los algoritmos de generación de AD, el procedimiento general de inducción del árbol está constituido por dos fases: la fase de entrenamiento (construcción del AD) y la fase de poda del árbol generado.

La poda busca un balance entre el aumento del error de clasificación y la reducción del tamaño del árbol. Además, el proceso de poda ayuda a reducir el sobreajuste que tenga el árbol, ya que el árbol completo puede clasificar perfectamente al conjunto de entrenamiento, pero puede no ser tan efectivo con un conjunto de prueba (conjunto de objetos no utilizados en la fase de construcción).

Los procesos de poda se basan en el error de clasificación del AD construido sobre un conjunto de prueba. El procedimiento general simplifica el árbol descartando uno o más subárboles y remplazándolos por hojas, la clase asignada a la nueva hoja se encuentra examinando los objetos de entrenamiento asociados al nodo, así la clase asignada será la de mayor frecuencia.

Los subárboles a reemplazar son elegidos examinando cada nodo interno del árbol completo, reemplazando el subárbol por una hoja y calculando el error de clasificación que genera el árbol podado sobre un conjunto de prueba. Ejemplos de estas técnicas de poda [34] son:

- Poda de error reducido. Empieza a recorrer el árbol construido de las hojas hacia la raíz y en cada nodo interno se calcula el error de clasificación, posteriormente se reemplaza el nodo por una hoja y

se calcula el error de clasificación con el árbol podado, si este error es menor que el calculado con el árbol completo, se reemplaza el nodo por la hoja, de lo contrario, se queda el árbol como estaba. Este proceso usa un conjunto de prueba para calcular los errores de clasificación.

- Poda costo-complejidad. Este proceso se realiza en dos pasos. En el primer paso se genera, a partir del árbol completo, una familia de árboles denotada por $T = \{T_0, \ldots, T_k\}$, donde $T_{i+1}$ es obtenido podando el árbol $T_i$. $T_0$ es el árbol completo y $T_k$ es un árbol representado sólo por el nodo raíz. Para construir el árbol $T_1$, se calcula el parámetro $\alpha$ para cada nodo interno $t$ de $T_0$, utilizando la ecuación 5. Aquellos nodos internos que tengan el mínimo valor de $\alpha$ serán reemplazados por hojas, la clase asignada a esa nueva hoja será la de mayor frecuencia en los objetos asociados a ese nodo. Este procedimiento se efectúa hasta que se contruye el arbol $T_k$. $\alpha$ se define como:

$$\alpha = \frac{e(t) - \sum_{l \in L_t} e(l)}{n(|L_t| - 1)} \tag{5}$$

  Donde, $t$ es el nodo interno a reemplazar, $n$ es el número de objetos, del conjunto utilizado para obtener el error de clasificación, que pasan por el nodo $t$ en el recorrido que hacen del árbol, $e(t)$ es el número de errores si $t$ es reemplazado por una hoja y $L_t$ es el conjunto de hojas que tiene el subárbol con raíz $t$.

  El segundo paso es encontrar el árbol de la familia T con el menor error de clasificación, utilizando un conjunto de prueba o realizando validación cruzada con el CE, este árbol elegido caracterizará al CE.

### 2.2. Algoritmos que generan árboles de decisión para grandes conjuntos de datos

A continuación, se dará una breve descripción de los algoritmos de generación de AD para grandes conjuntos de datos. Además, se mostrarán las principales características de cada uno de ellos.

#### 2.2.1. SLIQ [24]

La principal problemática que SLIQ ataca es tener que almacenar en memoria todo el conjunto de entrenamiento para construir el AD. Este conjunto es representando por medio de estructuras, de las cuales sólo una tiene que permanecer almacenada en memoria.

SLIQ utiliza listas que almacenan todos los posibles valores de cada atributo del conjunto de entrenamiento. Cada uno de los atributos tendrá una lista que lo describa, y cada lista será de tamaño igual al número de objetos presentes en el conjunto. Además, se tiene otra lista que representa la pertenencia de cada uno de los objetos a alguna de las clases, así como el número de nodo en el cual se encuentra el objeto. Se tendrá un total de $A + 1$ listas, donde $A$ es el número de atributos que describen a los objetos.

SLIQ es capaz de manipular conjuntos de datos mezclados (objetos descritos por variables cualitativas y cuantitativas), cada lista correspondiente a un atributo de este tipo es ordenada antes de iniciar la construcción del árbol. El criterio de selección de atributo que SLIQ utiliza es el Índice de Gini. La figura 1 muestra un ejemplo de las listas que genera SLIQ. Las listas que contienen las descripciones de los atributos pueden ser almacenadas en disco, y la única que se mantiene en memoria es la lista que contiene la pertenencia de los objetos a las clases. Sin embargo, si la cantidad de objetos del conjunto de entrenamiento es muy grande, la lista de la pertenencia a las clases puede ser de gran tamaño, tanto como para no poder ser almacenada

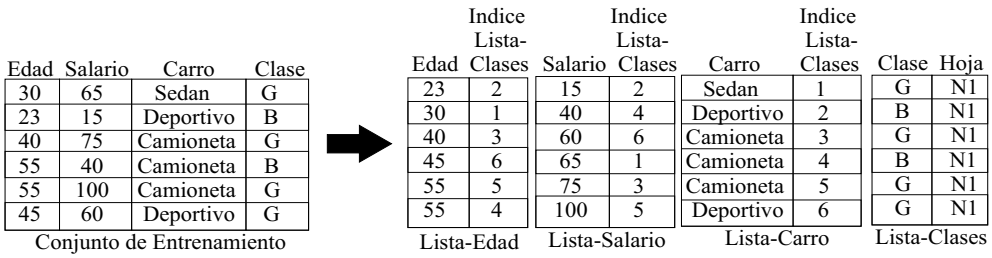


**Figura 1. Ejemplo de las listas usadas por SLIQ.**

en la memoria. Además, SLIQ construye sólo árboles de decisión binarios, con lo cual se puede perder interpretabilidad del AD por parte del usuario, ya que un mismo atributo se puede presentar en varios niveles con diferente valor de prueba [33].

### 2.2.2. SPRINT [25]

SPRINT es una mejora del algoritmo SLIQ, su diferencia radica en la forma en cómo se representan las listas para cada atributo. Ahora no será necesario almacenar en memoria ninguna lista, sin embargo, debido a que para expandir cada nodo se deben recorrer las listas, el tiempo de procesamiento puede ser demasiado grande, si el conjunto de entrenamiento cuenta con un gran número de objetos.

SPRINT continúa con la idea de tener una lista por cada atributo, pero ahora en lugar de tener una lista exclusiva para representar las pertenencias de los objetos a alguna de las clases, se le agrega un valor por cada registro (objeto) a las listas de cada atributo, este valor representa la clase a la que pertenece cada objeto. Sin embargo con esta modificación, el espacio para almacenar las listas aumenta, ya que además de tener en su descripción al valor del atributo, contiene la clase del objeto en cada lista. Un ejemplo de estas listas es mostrado en la figura 2. Debido a que SPRINT no almacena en memoria ninguna lista, porque

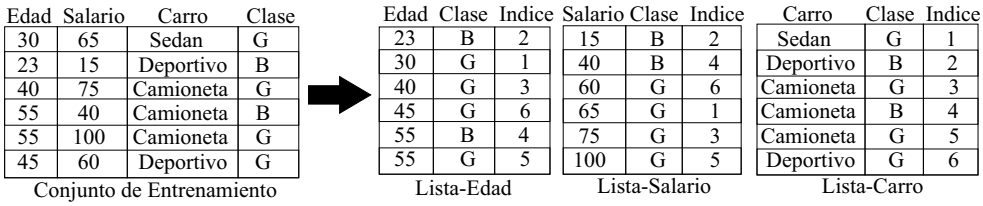


**Figura 2. Ejemplo de las listas usadas por SPRINT.**

ninguna es actualizada cuando se expande un nodo, ahora el algoritmo particiona las listas de acuerdo a las ramas generadas, con lo cual necesita accesar a todo el conjunto de datos cada vez que un nivel del árbol es expandido, así las listas son particionadas de acuerdo a los atributos elegidos en cada nodo.

SPRINT ha sido diseñado para procesar en paralelo el conjunto de entrenamiento, cada procesador contendrá una porción de este conjunto y de acuerdo al nodo en construcción, procesará las listas correspondientes de cada atributo, una vez que cada procesador tenga la dupla atributo-valor elegida, se hará un consenso para determinar cuál de ellas será la elegida para crear el nodo, de esta manera, la construcción del árbol es

más rápida. Sin embargo, no se asegura que el atributo elegido en consenso sea el mismo atributo que se hubiera elegido si las listas de los atributos se procesaran en su totalidad.

### 2.2.3. CLOUDS [26]

Al igual que SLIQ y SPRINT, CLOUDS sigue la filosofía de representar el conjunto de entrenamiento en listas para evitar almacenar todo el conjunto en la memoria principal. Las listas son manejadas como lo hace SPRINT. Sin embargo, la ventaja sobre ese algoritmo, es que los atributos numéricos son divididos en intervalos, y por lo tanto no se hace una revisión sobre todos los posibles valores de esos atributos.

CLOUDS divide el conjunto de valores que puede tomar cada atributo numérico en intervalos, de tal forma que cada intervalo contenga aproximadamente el mismo número de valores. Cuando un nodo va a ser expandido, CLOUDS aplica el índice de Gini a los límites de cada intervalo, aquél con menor valor en el índice de Gini será el atributo elegido para ese nodo. El autor presenta también una variación de este procedimiento, esta mejora sirve para encontrar un valor en esos intervalos que mejor divida al espacio de representación de cada atributo. Además de aplicar el índice de Gini a cada límite de los intervalos, estima un valor mínimo global con el índice de Gini. Con este valor mínimo se determina si cada intervalo es un candidato a tener el mejor valor con el que el atributo divida a los datos. Si un intervalo es candidato, entonces se evalúa el índice de Gini para cada uno de los valores que lo forman, lo que significa que, si todos los intervalos llegan a ser candidatos, se requerirá procesar una vez más todo el conjunto de datos.

El hecho de que CLOUDS trabaje con intervalos para los atributos numéricos, y no con todos sus valores como lo hace SPRINT, hace al algoritmo más rápido. Sin embargo, sigue presentando el problema de SPRINT, el tamaño de las listas a utilizar en la construcción del árbol puede ser casi el doble de lo que se requiere para almacenar el conjunto de datos completo.

### 2.2.4. RAINFOREST [27]

Debido a que en los algoritmos anteriores las estructuras utilizadas para representar al conjunto de entrenamiento pueden llegar a ocupar hasta el doble de espacio de lo requerido por el conjunto, RainForest trata de reducir estas estructuras de manera que puedan ser almacenadas en memoria.

La idea principal de RainForest es formar conjuntos de listas (denominados *AVC* por Atributo-Valor, Clase) para describir a los objetos del conjunto de entrenamiento. Estas listas contendrán a todos los posibles valores que un atributo pueda tomar, además de su frecuencia en el conjunto de entrenamiento. De esta manera, las listas no serán de gran longitud y con ello podrán ser almacenadas en memoria. Sin embargo, como en cada nodo se tienen que calcular estas listas *AVC*, la memoria puede ser insuficiente para almacenarlas en algún punto del procedimiento. Además, si el conjunto de datos está representado en su mayoría por atributos numéricos descritos por una gran cantidad de valores diferentes, el problema de memoria se incrementa, ya que las listas serán tan grandes como las utilizadas por el algoritmo SPRINT. La figura 3 muestra un ejemplo de cómo se representan los atributos mediante listas. RainForest, para expandir un nodo, leé una vez el conjunto de datos para construir la lista *AVC* a utilizar en ese nodo, luego se aplica un criterio de selección de atributos (este algoritmo es capaz de utilizar cualquier criterio de selección) y se crea un número determinado de hijos para el nodo (de acuerdo al atributo elegido). Por último, se lee otra vez el conjunto de datos, a fin de distribuir los objetos en alguno de los hijos creados. Como se puede observar, para la construcción de cada nivel del AD, se tiene que leer dos veces el conjunto de datos completo, además de generar las listas *AVC*, lo cual es muy costoso, especialmente si se quiere aplicar a grandes conjuntos de datos.

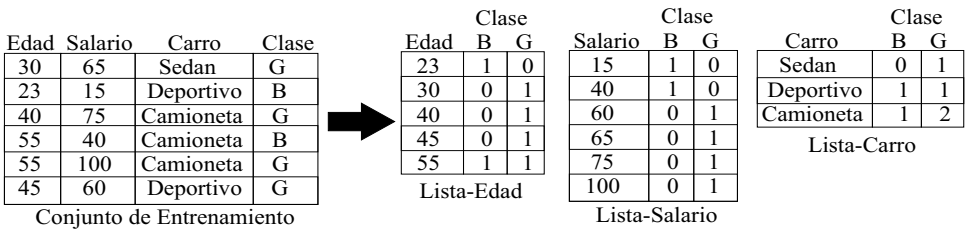


**Figura 3. Ejemplo de las listas usadas por RainForest.**

### 2.2.5. BOAT [28]

Para evitar tener que almacenar todo el conjunto de entrenamiento en memoria, BOAT sólo utiliza una porción de este conjunto para construir el AD, con lo cual no presenta restricciones espaciales. BOAT es un algoritmo incremental capaz de trabajar con grandes conjuntos de datos mezclados y con cualquier criterio de selección de atributo para construir los nodos del árbol. Lo innovador de este algoritmo es la forma de construir el AD.

Como primer paso obtiene una submuestra D del conjunto de entrenamiento. A esa submuestra se le aplica una técnica conocida como bootstrapping, la cual trabaja de la siguiente manera. A partir de la muestra D, se obtienen submuestras de ella y con cada una se construye un árbol. Cada árbol es recorrido simultáneamente de manera top-down (recorrido que empieza en la raíz y termina en una hoja) y para cada nodo recorrido se verifica si el atributo es el mismo en todos los árboles, si es así entonces el árbol final contendrá a ese atributo en ese nodo. En caso de que no sea el mismo atributo, se ignora ese nodo en todos los árboles construidos (borrando a los nodos y a los subárboles que se desprendan de ellos) y se sigue con el recorrido.

Con los atributos cualitativos se tendrán tantas ramas como valores en su dominio. Con los atributos numéricos se formará un intervalo considerando los valores que tengan en esos nodos los árboles que se construyeron de las submuestras. Una vez que se terminó de construir el árbol a partir de todos los árboles generados por las submuestras, se recorre una vez más todo el conjunto de entrenamiento, de tal manera que con esto se termine de construir el árbol final, ya que en los nodos con atributos de prueba numéricos, se tendrá que calcular el valor del atributo que caracterizará al nodo.

Para poder trabajar en un ambiente dinámico, es decir incremental, BOAT guarda todos los valores necesarios para poder realizar el cálculo del criterio de selección del atributo, de esta manera no es necesario calcular todo otra vez, sino sólo considerar al nuevo objeto.

El autor asegura que BOAT construye el mismo AD que cualquier método no incremental, tomando en cuenta el criterio de selección del atributo de este último, sin embargo, no demuestra que esto sea posible. Además, no muestra el procedimiento que se utiliza para formar la submuestra D del conjunto de entrenamiento, con lo cual no se sabe si esa submuestra es representativa de todo el conjunto.

### 2.2.6. ICE [29]

ICE procesa el conjunto de entrenamiento por partes, de tal manera que no lo tenga completamente almacenado en memoria. Así cada parte es procesada individualmente y de cada una de estas partes se obtiene una porción de objetos, que se van uniendo conforme se procesan las partes.

ICE es un modelo que funciona para grandes conjuntos de datos mezclados y basa su funcionamiento en un proceso incremental. El procedimiento general de este modelo inicia particionando al conjunto de

entrenamiento en subconjuntos de un mismo tamaño. Cada subconjunto será tratado como una época.

En cada época, se construye un árbol de decisión con la partición dada, aplicando cualquier algoritmo de generación de árboles de decisión, y a partir de ese árbol se genera una muestra representativa de la partición (ICE genera esta muestra aplicando alguna técnica de muestreo al AD construido, técnicas como: muestreo aleatorio, agrupamiento local, etc. aunque el autor no específica como son aplicadas estás técnicas).

El procedimiento de ICE se muestra en la figura 4, en donde para cada época, se construye un árbol $T_i$ a partir de la partición $D_i$, y de cada árbol $T_i$ se extrae un subconjunto de muestras $S_i$, estas muestras se van uniendo y se construye un árbol $C_i$. Si por ejemplo, el conjunto de entrenamiento se divide en 2 particiones, el árbol $C_2$ caracteriza a ese conjunto, en dado caso de que llegue una nueva partición ($D_3$), se descarta el arbol $C_2$ y se utiliza la unión de $S_1$ y $S_2$ ($U_2$) como la información disponible de las particiones 1 y 2, estas submuestras se unen a $S_3$ para formar el nuevo árbol que caracterizará a las particiones 1, 2 y 3. En la figura 4 el árbol $C_k$ es el AD final que caracteriza a todo el conjunto de entrenamiento. ICE construye dos árboles

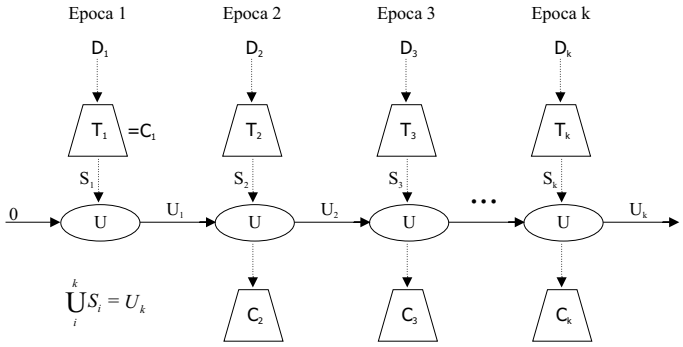


**Figura 4. Procedimiento general de ICE.**

para cada una de las épocas, uno con el que se obtienen las muestras representativas de esa partición, y otro para caracterizar a las particiones procesadas hasta el momento.

El procedimiento incremental de este algoritmo, no permite que el AD se vaya actualizando si los objetos van llegando de uno en uno. ICE espera a que un número de objetos nuevos se hayan acumulado, así esta nueva porción de objetos será tratada como una época más del conjunto de datos.

### 2.2.7. PUDT [30]

PUDT (por sus siglas en inglés - Parallel Univariate Decision Trees) construye AD univaluados en forma paralela. Este trabajo sólo sigue la idea de paralelizar el procedimiento de dos algoritmos conocidos para generar AD, uno es el C4.5 y otro es el basado en discriminantes lineales para la construcción del árbol. Debido a que se paraleliza el proceso de construcción, es posible aplicar el algoritmo a grandes conjuntos de datos.

El proceso de paralelización se realiza de alguna de las siguientes 3 maneras:

- Por atributos, cada atributo es manipulado por un procesador diferente, es decir, cada uno determina el valor de la ganancia de información de su atributo y el concentrador se encarga de elegir al atributo de mayor ganancia para formar el nodo.

- Por nodos, donde cada nodo entra a una lista de espera para que algún procesador lo tome y lo expanda.
- Por datos, donde éstos se particionan en k subgrupos (si existen k procesadores). El concentrador le indica a los procesadores la partición del conjunto de datos que le corresponde a cada uno, ellos le devuelven el atributo con mayor ganancia de su partición, el concentrador combina estos valores y finalmente determina qué atributo y valor, se van a utilizar para expandir el nodo. Sin embargo, con esto no se asegura que el atributo elegido sea el mismo que se hubiera elegido usando la muestra completa.

Para el caso del algoritmo basado en el discriminante lineal, sólo se pueden procesar problemas de 2 clases, así que aquellos que contengan más, se tendrán que dividir en subproblemas de 2 clases, lo que puede resultar más costoso al momento de construir un árbol que represente a todas las clases del conjunto de entrenamiento.

### 2.3. Discusión

Con base en el estudio anterior de los algoritmos que generan árboles de decisión para grandes conjuntos de datos, se observó que no hay alguno que procese todo el conjunto de entrenamiento accesando sólo una vez al mismo y teniendo la capacidad de procesar conjuntos dinámicos, ya que aunque algunos algoritmos son rápidos porque sólo accesan al conjunto de entrenamiento una vez, no toman en cuenta al conjunto en su totalidad, o bien, algunos otros si lo toman en cuenta en forma total, pero accesan a él en más de una vez, esto se debe a que para elegir el atributo de prueba de cada nodo, tienen que procesar todo el conjunto de entrenamiento en cada nivel del árbol para cada atributo presente en el conjunto. Esto hace que no se puedan aplicar a conjuntos de datos muy grandes. Además no todos los algoritmos pueden trabajar con conjuntos de datos dinámicos, debido a que los procedimientos que utilizan no están concebidos para trabajar de manera incremental.

Por esto, es necesario proponer nuevos algoritmos para generar AD a partir de grandes conjuntos de datos que, además de procesar los datos en un tiempo razonable, puedan considerar conjuntos de datos dinámicos, obteniendo una buena calidad de clasificación. Una opción es generar AD multivaluados, de tal manera que no sea necesario accesar al conjunto de entrenamiento cada vez que se expanda un nodo.

El cuadro 1 presenta un resumen de los algoritmos para la generación de AD para grandes conjuntos de datos y señala las características que tendrá el algoritmo de generación de AD que se propondrá en este trabajo.

| Características | Algoritmo | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | Sliq | Sprint | Clouds | RainF | Boat | Ice | Pudt | Alg. Prop. |
| Algoritmo Incremental | NO | NO | NO | NO | SI | SI | NO | SI |
| AD Univaluado (U) o Multivaluado (M) | U | U | U | U | U | U | U | M |
| Procesamiento de todo el conjunto de datos | SI | SI | SI | SI | NO | NO | SI | SI |
| Acceso al conjunto de datos sólo una vez | NO | NO | NO | NO | NO | SI | NO | SI |

**Cuadro 1. Algoritmos que generan AD para grandes conjuntos de datos**

# 3. Propuesta de Investigación

## 3.1. Motivación

Una de las razones por las cuales las técnicas de Minería de Datos son muy usadas, es que existe la necesidad de transformar una gran cantidad de datos en información y conocimiento útil.

El hecho de tener una gran cantidad de datos y no contar con herramientas que los puedan procesar ha producido un fenómeno descrito como: riqueza en datos pero pobreza en información [35]. Ese crecimiento constante de datos, el cual es guardado en grandes bases de datos, ha excedido la habilidad del ser humano de poder comprenderlos. Además, diversos problemas pueden llegar a presentar un flujo constante de datos, con lo que puede resultar más difícil el poder hacer un análisis de la información.

Cuando se trata de un problema de clasificación supervisada, contar con un conjunto de entrenamiento numeroso puede resultar en un proceso de decisión muy costoso, asignar una clase a un objeto puede no ser sencillo. Una herramienta útil en estos casos, es aquella que permita tomar una decisión basada en el conjunto de entrenamiento completo en un tiempo razonable.

No todos los algoritmos de generación de árboles de decisión para grandes conjuntos de datos toman en cuenta a todo el conjunto de entrenamiento. Hay algunos algoritmos que construyen el árbol con solo una parte del conjunto. Por otra parte, los algoritmos que si toman en cuenta al conjunto de entrenamiento completo presentan otras restricciones, como por ejemplo, la representación que hacen del conjunto de datos, la cual puede llegar a ocupar un espacio más grande que sólo almacenar el conjunto de entrenamiento, por lo que aplicar estos algoritmos a grandes conjuntos de datos resultaría en una complejidad espacial mayor.

Por esto surge la motivación de proponer un algoritmo de generación de árboles de decisión para grandes conjuntos de datos, el cual trabaje con todo el conjunto de entrenamiento disponible sin la necesidad de mantenerlo todo en memoria y que además permita el manejo de grandes conjuntos de datos dinámicos.

## 3.2. Pregunta de Investigación y Objetivos

### 3.2.1. Pregunta de Investigación

*¿Es posible crear un algoritmo de generación de árboles de decisión para grandes conjuntos de datos que procese todo el conjunto de entrenamiento, capaz de trabajar con conjuntos de datos dinámicos, el cuál sea competitivo en calidad y más rápido que los algoritmos existentes?*

### 3.2.2. Objetivo General

El objetivo general de esta investigación es proponer dos algoritmos que permitan la generación de árboles de decisión para grandes conjuntos de datos dinámicos, que sean competitivos en calidad y más rápidos que los algoritmos existentes.

### 3.2.3. Objetivos Particulares

Los objetivos particulares de este trabajo son los siguientes:

1. Desarrollar un algoritmo que genere árboles de decisión multivaluados, con todo el conjunto de atributos, para conjuntos de datos numéricos.

2. Desarrollar un algoritmo que genere árboles de decisión multivaluados, con subconjuntos de atributos, para conjuntos de datos numéricos.

3. Extender los algoritmos de los objetivos (1) y (2) para manipular conjuntos de datos mezclados.

### 3.3. Contribuciones

La principal contribución de este trabajo será el desarrollo de dos nuevos algoritmos para la generación de árboles de decisión a partir de grandes conjuntos de datos, tanto estáticos como dinámicos, que sean competitivos en calidad y más rápidos que los actuales.

### 3.4. Metodología

La metodología para poder alcanzar los objetivos propuestos en este trabajo es la siguiente:

1. Obtener grandes conjuntos de datos reales, que puedan ser utilizados para probar los algoritmos.

2. Para generar un AD multivaluado para grandes conjuntos de datos numéricos, que tenga todo el conjunto de atributos en sus nodos, se seguirán los siguientes pasos:

   *a*) Definir la estructura de los nodos del AD. Buscar una representación para los nodos internos y sus arcos, así como para las hojas del árbol, de tal manera que puedan ser manipulados grandes conjuntos de datos.

   *b*) Proponer la rutina de actualización del árbol generado, el algoritmo leerá uno a uno los objetos del conjunto de entrenamiento.

   *c*) Para el proceso de expansión, crear diversas representaciones de los arcos generados por los nodos expandidos.

   *d*) Para el proceso de clasificación de objetos nuevos, definir una estrategia para recorrer el árbol construido, de tal manera que se llegue a una hoja.

   *e*) Desarrollar estrategias para clasificar objetos nuevos a partir del AD generado, cuando se alcance un nodo hoja en el recorrido que se haga del árbol.

   *f*) Validar el algoritmo propuesto con grandes conjuntos de datos numéricos, para observar su comportamiento con respecto al tamaño del conjunto de entrenamiento y de acuerdo al número de atributos que describe al conjunto. Algunas de las posibles características a evaluar serán: el porcentaje de aciertos del algoritmo, el tiempo de procesamiento y el tamaño del árbol de decisión generado.

   *g*) En caso de ser necesario, redefinir alguno de los puntos anteriores.

   *h*) Comparar los resultados del algoritmo propuesto contra C4.5 y con uno o más algoritmos presentados en el trabajo relacionado

3. Para generar un AD multivaluado para grandes conjuntos de datos numéricos, que tenga a un subconjunto de atributos en sus nodos, se seguirá el procedimiento anterior, con las siguientes modificaciones:

   *a*) Proponer estrategias para encontrar subconjuntos de atributos para utilizarlos en la representación de los nodos internos y sus arcos, y que además permitan el manejo de grandes conjuntos de datos.

*b*) Para el proceso de expansión, proponer diversas representaciones de los arcos generados por los nodos expandidos, tomando en cuenta el subconjunto de atributos elegido en el paso (a).

*c*) Desarrollar estrategias para clasificar objetos nuevos a partir del AD generado, cuando se alcance un nodo hoja en el recorrido que se haga del árbol.

*d*) Validar el algoritmo propuesto con grandes conjuntos de datos numéricos, para observar su comportamiento con respecto al tamaño del conjunto de entrenamiento y de acuerdo al número de atributos que describe al conjunto. Algunas de las posibles características a evaluar serán: el porcentaje de aciertos del algoritmo, el tiempo de procesamiento y el tamaño del árbol de decisión generado.

*e*) En caso de ser necesario, redefinir alguno de los puntos anteriores.

*f*) Comparar los resultados del algoritmo propuesto contra C4.5 y con uno o más algoritmos presentados en el trabajo relacionado.

4. En la extensión de los algoritmos para que puedan manipular grandes conjuntos de datos mezclados, aplicar los siguientes pasos:

   *a*) Desarrollar estrategias para datos mezclados que permitan encontrar representaciones de los arcos generados por los nodos expandidos y que puedan manipular grandes conjuntos de datos.

   *b*) Proponer técnicas que generen subconjuntos de atributos, que se utilicen en la representación interna del árbol.

   *c*) Proponer estrategias para clasificar objetos nuevos, descritos por atributos mezclados.

   *d*) Validar el algoritmo propuesto con grandes conjuntos de datos mezclados, para observar su comportamiento con respecto al tamaño del conjunto de entrenamiento y de acuerdo al número de atributos que describe al conjunto. Algunas de las posibles características a evaluar serán: el porcentaje de aciertos del algoritmo, el tiempo de procesamiento y el tamaño del árbol de decisión generado.

   *e*) Comparar los resultados del algoritmo propuesto contra C4.5, debido a que es un algoritmo que también permite trabajar con datos mezclados, y con uno o más algoritmos presentados en el trabajo relacionado.

### 3.5. Calendario de Actividades

El cuadro 2 muestra el calendario de actividades a seguir.

## 4. Resultados Preliminares

Esta sección presenta los resultados obtenidos hasta el momento. En la primera parte se presenta un nuevo algoritmo de generación de árboles de decisión para grandes conjuntos de datos y en la segunda parte se presentan los resultados experimentales que se han obtenido a partir de la aplicación de este algoritmo a algunos conjuntos de datos.

| Actividad | Cuatrimestre | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| Definición del tema de tesis doctoral | | | | | | | | | |
| Recopilación y análisis de los trabajos relacionados con AD | | | | | | | | | |
| Obtener grandes conjuntos de datos reales | | | | | | | | | |
| (2a), (2b), (2c) | | | | | | | | | |
| (2d), (2e), (2f), (2g), (h) | | | | | | | | | |
| (3a), (3b) | | | | | | | | | |
| (3c), (3d), (3e), (3f) | | | | | | | | | |
| (4a), (4b), (4c) | | | | | | | | | |
| (4d), (4e) | | | | | | | | | |
| Propuesta de tesis doctoral | | | | | | | | | |
| Redactar artículos para revistas o congresos | | | | | | | | | |
| Redactar la tesis doctoral | | | | | | | | | |
| Defensa de la tesis doctoral | | | | | | | | | |

**Cuadro 2. Calendario de Actividades**

### 4.1. Algoritmo Propuesto

El algoritmo propuesto constará de 2 etapas, al igual que los demás algoritmos de generación de AD, una etapa de inducción y una etapa de clasificación. En la etapa de inducción se construirá el AD a partir de todo el conjunto de entrenamiento, en la etapa de clasificación se dará la pertenencia a alguna clase, basándose en el árbol construido, a objetos que no fueron tomados en cuenta para la creación del AD.

#### 4.1.1. Etapa de inducción del AD

Dado que el objetivo es crear AD para conjuntos de entrenamiento que no puedan ser almacenados en memoria, por tratarse de grandes conjuntos de datos, se propone como primer resultado un algoritmo que genere el AD de manera incremental, procesando uno a uno los objetos del conjunto de entrenamiento. De esta manera el algoritmo propuesto no necesita mantener en memoria a todo el conjunto de entrenamiento para generar el AD, sino sólo al objeto que esté procesando en cada momento. Además para evitar recorrer el conjunto de entrenamiento cada vez que se va a seleccionar un atributo de prueba en un nodo, se propone construir un árbol multivaluado considerando todos los atributos en cada nodo.

La estructura del árbol que se propone almacenará objetos del conjunto de entrenamiento en ella, sin embargo conforme vaya creciendo el árbol se obtendrán objetos que representen a los objetos almacenados en el árbol de tal manera que el espacio requerido por esos objetos representativos sea menor que el ocupado por los objetos del conjunto de entrenamiento, logrando así que el espacio requerido para almacenar el AD no sea muy grande. Por todo lo anterior el algoritmo propuesto sólo requerirá espacio para almacenar el AD (el cual es menor que el espacio requerido para almacenar el conjunto de entrenamiento), a diferencia de los demás algoritmos de generación de AD que también requieren espacio para almacenar el conjunto de entrenamiento.

Antes de que el algoritmo procese el conjunto de entrenamiento, se construirá un árbol representado sólo por el nodo raíz. Cuando se procese el primer objeto del conjunto de entrenamiento se almacenará en el

nodo raíz, que en este momento es una hoja. La figura 5 muestra este proceso, suponiendo que se tiene un conjunto de entrenamiento de $m$ objetos, clasificados en $r$ clases, inicialmente se procesa el objeto 1 ($O1$) y se almacena en el nodo raíz.

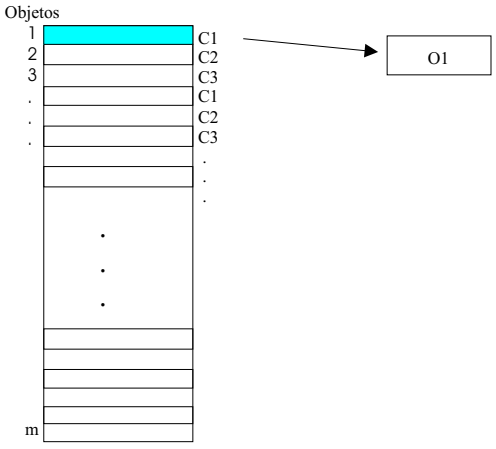


**Figura 5. Inicio del algoritmo propuesto.**

El segundo objeto procesado también es almacenado en el nodo raíz, este proceso continúa hasta que se hayan procesado $s$ objetos, siendo $s$ un parámetro predefinido que indica el número máximo de objetos que se permiten almacenar en una hoja, se define este parámetro ya que como se trabaja con grandes conjuntos de datos se propone reducir el número de objetos obteniendo, de cada porción de objetos, sólo un subconjunto que los represente. Cuando se llega a $s$ objetos entonces el nodo raíz estará lleno y hay que expandirlo convirtiéndolo en un nodo interno. En la figura 6, se observa cómo se procesaron los primeros $s$ objetos.

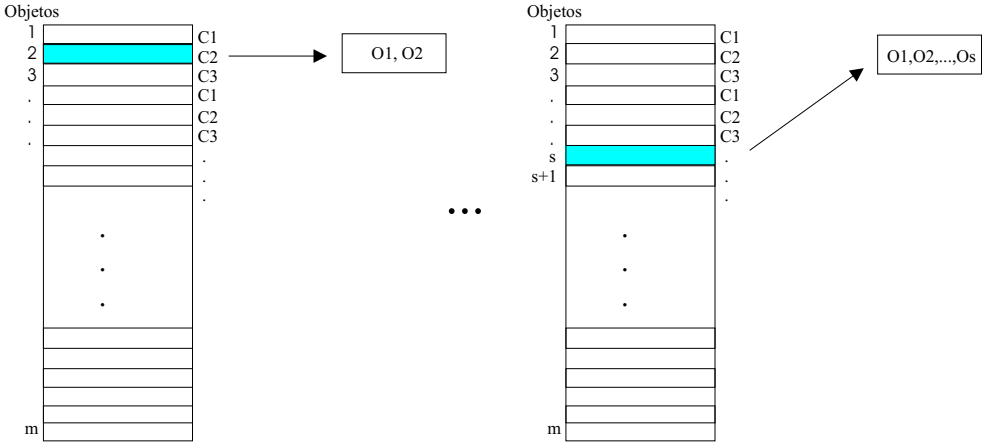


**Figura 6. Procesamiento de los primeros $s$ objetos por el algoritmo propuesto.**

Cuando una hoja tenga $s$ objetos almacenados, se expandirá el nodo, el número de arcos a crear estará determinado por el número de clases presentes en los objetos almacenados en el nodo, si tiene almacenados

objetos de $k$ clases, se generarán $k$ arcos. Cada arco corresponderá a una clase, es decir, para crear el arco 1 se utilizarán los objetos que pertenecen a la clase 1, para el arco 2 a los de la clase 2 y así sucesivamente. Como se trata de grandes conjuntos de datos, hay que tener almacenada la menor cantidad de información posible, por esto, se propone que los arcos no estén caracterizados por todos los objetos de la clase correspondiente que había en el nodo a expandir, sino sólo por un objeto o un subconjunto de objetos que sean representativos de esa porción de la clase, los cuales denotaremos por $OR$. La figura 7 muestra la expansión del nodo raíz una vez que se almacenaron $s$ objetos en él.

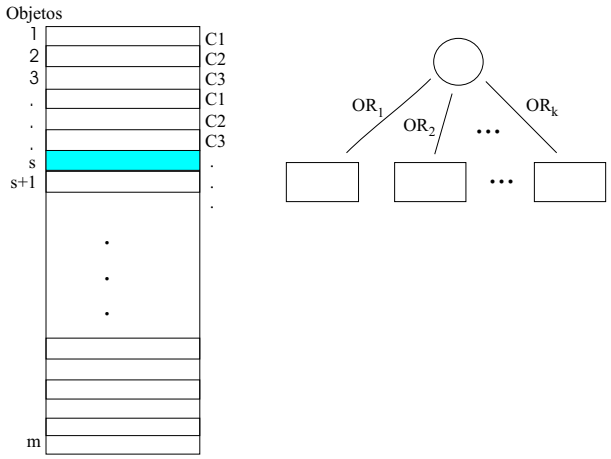


**Figura 7. Proceso de expansión de un nodo en el algoritmo propuesto.**

En el nodo raíz, que se convierte en un nodo interno, en el cual no se almacena ningún objeto, sólo se mantiene un conteo de los objetos que han pasado por él. Los nodos hojas creados están inicialmente vacíos, de este modo de los primeros $s$ objetos sólo se almacenan aquellos que sean representativos de las $k$ clases en las que se distribuyen.

Para recorrer el árbol dado un objeto, hay que calcular con qué conjunto de objetos representativos es más parecido, aquél conjunto de $OR$ con el que tenga más parecido, será el arco por el que debe seguir. De este modo se siguen procesando los objetos de la muestra hasta que alguna de las hojas se llena (es alcanzada por $s$ objetos), siendo entonces expandida de la misma manera que el nodo raíz. La figura 8 muestra la expansión de un nodo hoja, se asume que los objetos almacenados en la hoja pertenecen a $k'$ clases, por lo que se crean $k'$ arcos.

El proceso de generación del AD continúa hasta que todos los objetos del conjunto de entrenamiento se hayan procesado, entonces el árbol estará caracterizado por los objetos representativos obtenidos de cada arco y en las hojas, por un subconjunto de $n$ objetos del conjunto de entrenamiento, $n < s$. La figura 9 muestra el árbol final una vez que se procesaron los $m$ objetos.

Debido a que el proceso de expansión de los nodos del algoritmo propuesto se basa en los objetos almacenados en los nodos, es necesario que el nodo a expandir tenga una variedad de objetos en él, es decir, que los objetos que estén almacenados ahí sean de diferentes clases, esto porque si se tienen los objetos ordenados de acuerdo a las clases, los primeros nodos a expandir contendrán sólo a objetos de la clase 1, por lo que sólo se creará un arco de esos nodos, luego cuando se procesen los objetos de la clase 2, se formarán nodos que sólo tengan a objetos de esa clase, y así sucesivamente. Es por esto, que se propone que en una fase de preprocesamiento los objetos del conjunto de entrenamiento se ordenen alternando un objeto de cada clase

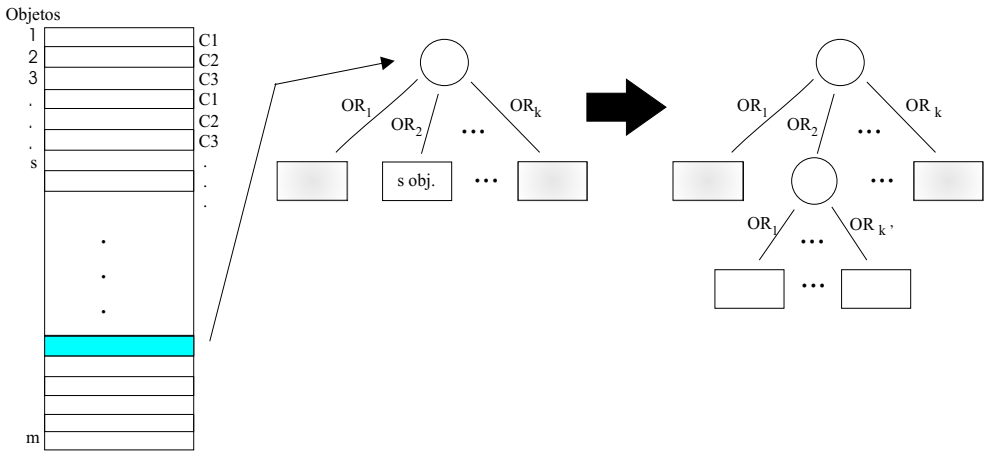


**Figura 8. Ejemplo del proceso de expansión del algoritmo propuesto.**

a la vez.

El algoritmo de generación de AD para grandes conjuntos de datos propuesto en este trabajo es el siguiente:

**Entrada:** $CE$(conjunto de entrenamiento), $s$
**Salida:** Árbol de decisión
*Paso 1:*$RAIZ \leftarrow$ CreaNodo()
*Paso 2:* Para cada $O_i \in CE$, hacer
ActualizaAD($O_i$, $RAIZ$)

Donde la función *CreaNodo()* asigna espacio en memoria para la creación de un nuevo nodo y la función *ActualizaAD($O_i$, RAIZ)* está definida de la siguiente manera:

**ActualizaAD ($O_i$, NODO)**
Si NODO.numObj $\leq s$, entonces
AgregarObjNodo(NODO, $O_i$)
Si NODO.numObj = $s$, entonces
ExpandeNodo(NODO)
NODO.numObj = NODO.numObj + 1
En otro caso /* NODO.numObj $> s$ */
Para cada arco $R_j \in$ NODO, hacer
semej[j] = Semejanza($O_i$, NODO.ORj)
sigueArco = ArcoMax (semej)
ActualizaAD ($O_i$, NODO.sigueArco)

En este procedimiento las funciones utilizadas realizan las siguientes tareas:

- *AgregarObjNodo(NODO, $O_i$)*: Almacena el objeto $O_i$ en NODO e incrementa en uno el número de objetos que almacena.

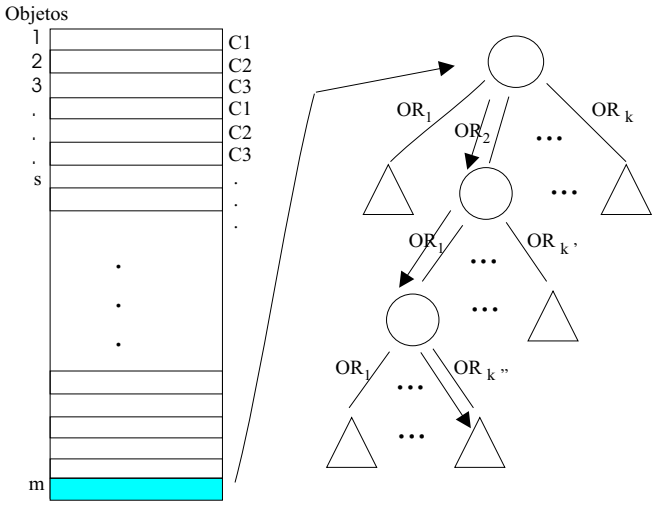


**Figura 9. Proceso finalizado por el algoritmo propuesto.**

- *ExpandeNodo(NODO)*: Genera los arcos de NODO, uno por cada clase presente en él y caracterizados por un subconjunto de objetos representativos (OR). Además, se crea un nodo hoja para cada una de ellos.
- *Semejanza($O_i$, NODO.ORj)*: Obtiene el valor de semejanza entre el objeto $O_i$ y el conjunto de objetos representativos del arco $j$. En los experimentos se usó el valor de semejanza que tiene el objeto $O_i$ con el objeto más parecido del conjunto de objetos representativos $j$.
- *ArcoMax(semej)*: Especifica el número de arco, que posee la mayor semejanza con el objeto $O_i$.

### 4.1.2. Etapa de Clasificación

Cuando se quiere clasificar un objeto nuevo, se recorre el árbol de manera top-down con este objeto. Cuando el objeto nuevo pase por un nodo interno, el algoritmo determinará el arco por el cual descenderá, verificando a qué conjunto de objetos representativos es más parecido, cuando se llegue a una hoja, el algoritmo decidirá cuál es la clase que se le asignará al objeto de acuerdo a la información almacenada en el nodo, una forma de hacer esta asignación es buscando a qué objeto almacenado en la hoja se parece más el objeto nuevo y asignando la clase a la que pertenece el objeto más parecido.

El algoritmo para clasificar a los objetos del conjunto de prueba (CP) utilizando el AD generado por el algoritmo propuesto se describe a continuación:

**Entrada:** $CP$(conjunto de prueba), $AD$
**Salida:** Pertenencia de los objetos a alguna clase.
*Paso 1:* $RAIZ = InicioAD()$
*Paso 2:* Para cada $O_i \in CP$, hacer
Clasifica($O_i$, $RAIZ$)

Donde $InicioAD$ situa a $RAIZ$ en el nodo raíz del árbol generado y $Clasifica$ se describe a continuación:

**Clasifica ($O_i$, NODO)**
Si NODO.numObj $\geq s$, entonces
Para cada arco $R_j \in$ NODO, hacer
semej[j] = Semejanza($O_i$, NODO.ORj)
sigueArco = ArcoMax (semej)
Clasifica ($O_i$, NODO.sigueArco)
En otro caso /* NODO.numObj $< s$ */
Clase = SemejanzaHoja(NODO.obj, $O_i$)
$O_i \leftarrow Clase$

$SemejanzaHoja$ calcula la semejanza que tiene el objeto a clasificar ($O_i$) con todos los objetos almacenados en esa hoja, la función regresa la etiqueta de la clase a la que pertenece el objeto más semejante a $O_i$.

### 4.1.3. Selección de los objetos representativos (OR)

La primera forma que se ha utilizado para calcular el objeto representativo de cada arco, es calculando la media de los objetos con los que se creó el arco correspondiente. En el ejemplo que se ha manejado, antes de expandir cada nodo, éstos tenían almacenados $s$ objetos, ahora sólo se tienen $k$ objetos que caracterizan a cada uno de los arcos creados, donde $k$ es el número de clases en las que se distribuyen los $s$ objetos.

### 4.1.4. Ejemplo de la generación de un AD por el algoritmo propuesto.

A continuación se presenta un ejemplo de la generación de un AD por el algoritmo propuesto a partir del conjunto de entrenamiento descrito en la figura 10. En este ejemplo $X1 \in [2{,}4 - 4{,}0]$ y $X2 \in [45{,}3 - 55{,}8]$ para la clase $C1$, mientras que en la clase $C2$ $X1 \in [9{,}6 - 12{,}1]$ y $X2 \in [15{,}3 - 39{,}9]$.

| Objeto | X1 | X2 | Clase |
|---|---|---|---|
| O1 | 3.2 | 50.2 | C1 |
| O2 | 2.4 | 45.3 | C1 |
| O3 | 2.6 | 51.7 | C1 |
| O4 | 4.0 | 49.9 | C1 |
| O5 | 3.8 | 55.8 | C1 |
| O6 | 10.3 | 15.3 | C2 |
| O7 | 11.5 | 20.6 | C2 |
| O8 | 12.1 | 19.6 | C2 |
| O9 | 9.6 | 39.9 | C2 |
| O10 | 11.1 | 38.3 | C2 |

| Objeto | X1 | X2 | Clase |
|---|---|---|---|
| O1 | 3.2 | 50.2 | C1 |
| O6 | 10.3 | 15.3 | C2 |
| O2 | 2.4 | 45.3 | C1 |
| O7 | 11.5 | 20.6 | C2 |
| O3 | 2.6 | 51.7 | C1 |
| O8 | 12.1 | 19.6 | C2 |
| O4 | 4.0 | 49.9 | C1 |
| O9 | 9.6 | 39.9 | C2 |
| O5 | 3.8 | 55.8 | C1 |
| O10 | 11.1 | 38.3 | C2 |

**Figura 10. Conjunto de entrenamiento de ejemplo para el algoritmo propuesto.**

Supongamos que $s = 3$ y los objetos representativos se generan calculando la media de los objetos almacenados en el nodo.

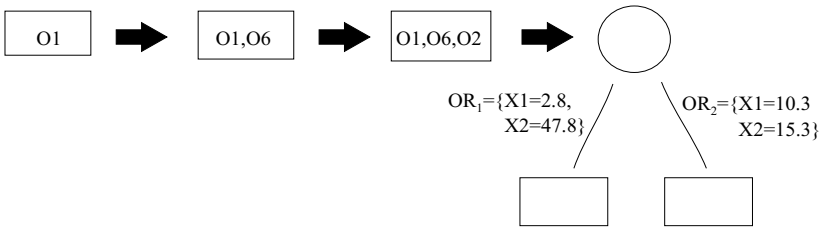


**Figura 11. Inicio del algoritmo propuesto con el conjunto de entrenamiento de la figura 10.**

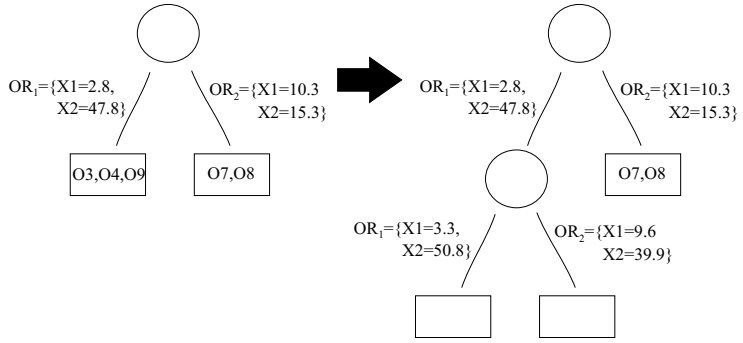


**Figura 12. Proceso de expansión del algoritmo propuesto con el CE de la figura 10.**

Se inicia generando el nodo raíz y leyendo objetos hasta que se llene el nodo, en este caso se leerán 3 objetos y se procederá a expandir el nodo raíz. La figura 11 muestra este procedimiento.

El objeto 4 tiene que recorrer el árbol para saber qué camino seguir y llegar a una hoja en donde será almacenado. Para esto se calcula con qué objeto representativo es más semejante y ese será el arco por el que descienda, en este caso es el arco correspondiente a $OR_2$ y como por ese camino se llega a una hoja, el objeto es almacenado en ella. Este procedimiento se repite hasta que alguna hoja alcance el número máximo de objetos que puede almacenar. La figura 12 muestra el árbol hasta que una hoja se llena con s objetos y se procede a expandirla.

Se continúan procesando los objetos del conjunto de entrenamiento hasta que todos los objetos hayan sido procesados. La figura 13 muestra el árbol final para el conjunto de entrenamiento de la figura 10.

Supongamos que se quiere clasificar el objeto $O = \{X1 = 9{,}8, X2 = 37{,}4\}$, entonces se tiene que recorrer el árbol de la figura 13, calculando, en los nodos internos, con que objeto representativo es más parecido $O$ para seguir ese camino, así hasta llegar a un nodo hoja. La figura 14 muestra el camino que sigue $O$ en el árbol.

Una vez que $O$ llegó a la hoja, se procede a calcular con qué objeto, de los almacenados ahí, es más parecido. En este caso como sólo se tiene un objeto en la hoja ($O10$) se le asigna la clase a la que pertenece este objeto, por lo tanto $O \in C2$. Lo que indica que el objeto fue clasificado correctamente, tomando en cuenta cómo son las clases $C1$ y $C2$, ya que en $C2$ $X1 \in [9{,}6 - 12{,}1]$ y $X2 \in [15{,}3 - 39{,}9]$ y la descripción de $O$ corresponde a estos intervalos.

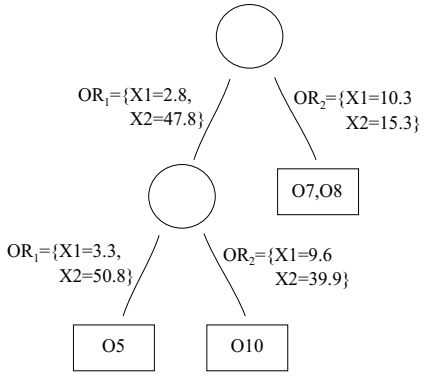


**Figura 13. Árbol generado por el algoritmo propuesto para el CE de la figura 10.**

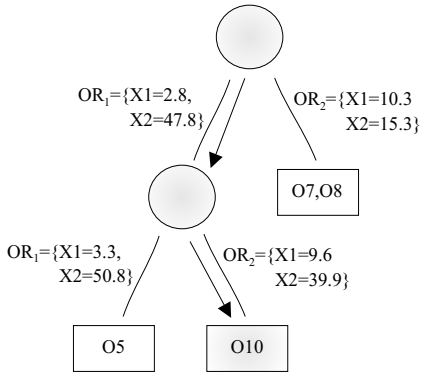


**Figura 14. Recorrido del objeto $O$ en el árbol de la figura 13.**

### 4.2. Experimentos

El algoritmo propuesto se ha aplicado a diversos conjuntos de datos. Dos conjuntos de datos obtenidos del repositorio de bases de datos de la UCI [36] y un conjunto obtenido de una base de datos proporcionada por SDSS (Sloan Digital Sky Survey) [37], el cuadro 3 muestra las características de los conjuntos de datos utilizados.

| Conjunto | No. Clases | No. Objetos | No. Atributos |
|---|---|---|---|
| Letter | 26 | 20,000 | 16 |
| Poker | 10 | 1,000,000 | 10 |
| SpecObj | 6 | 884,054 | 5 |

**Cuadro 3. Descripción de los conjuntos de datos utilizados en los experimentos**

En todos los experimentos realizados se utilizó la media para obtener el objeto representativo de cada arco a generar, además para clasificar los objetos del conjunto de prueba utilizado en cada experimento, se verificó con que objeto de los almacenados en la hoja a la que llegó el nuevo objeto era más semejante a él, la clase a la que pertenece ese objeto más semejante es la clase a asignar al nuevo objeto.

Los resultados fueron comparados contra los del algoritmo C4.5, considerando el árbol completo y el árbol podado que genera este algoritmo, se utilizó a C4.5 debido a que algunos de los algoritmos de generación de árboles de decisión para grandes conjuntos de datos lo toman como referencia en sus procesos comparativos. Además se realizaron comparaciones con el algoritmo de generación de AD para grandes conjuntos de datos ICE. La comparación se hizo sobre el porcentaje de aciertos del algoritmo, los tiempos de ejecución y el número de nodos que forman el AD. Se comparan estos datos, ya que además de que el algoritmo debe tener un buen porcentaje de aciertos, en algoritmos de generación de AD que procesan grandes conjuntos de datos, es fundamental que el proceso se realice en un tiempo razonable y que el árbol sea lo más compacto posible.

#### 4.2.1. Letter

Letter [36] consta de 20,000 objetos descritos por 16 atributos numéricos (todos enteros). Los objetos están agrupados en 26 clases. Con este conjunto de datos se realizaron 2 pruebas diferentes, la primera es para observar el comportamiento del algoritmo propuesto al variar el valor del parámetro $s$ y la segunda prueba para comparar, mediante validación cruzada, el algoritmo propuesto con un valor de $s = 100$.

En la primera prueba con Letter, se utilizó sólo una partición del conjunto de datos, el conjunto de entrenamiento se compone de 19,500 objetos y el conjunto de prueba de 500 objetos. Esta partición fue elegida separando los primeros 500 objetos del conjunto de datos completo. Los resultados obtenidos en esta prueba para los algoritmos C4.5 e ICE se muestran en el cuadro 4.

El cuadro 5 muestra los resultados obtenidos con el algoritmo propuesto, variando el valor de s.

La variación del parámetro $s$ en el algoritmo propuesto, tiene como consecuencia dos efectos cuando su valor aumenta, el tamaño del árbol generado disminuye y el tiempo de ejecución aumenta ligeramente. En cuanto al porcentaje de aciertos del algoritmo, se puede observar que se mantiene estable aunque se varíe el valor de $s$. Comparando estos resultados con los de C4.5, se puede observar que el algoritmo propuesto es superior en cuanto a tiempos de ejecución y tamaño del árbol para todos los valores de $s$ y respecto al porcentaje de aciertos del algoritmo se puede apreciar que es mayor a la de C4.5 en el caso de $s = 100$. Para

| | Tiempo | Porcentaje de aciertos | | Tamaño | |
|---|---|---|---|---|---|
| | | Completo | Podado | Completo | Podado |
| C4.5 | 6.72 | 87.8 | 87.6 | 2705 | 2489 |
| ICE | 84.77 | 70.0 | | 683 | |

**Cuadro 4. Resultados con C4.5 e ICE para Letter con 19,500 objetos de entrenamiento y 500 de prueba**

| s | Tiempo | Porcentaje de aciertos | Tamaño |
|---|---|---|---|
| 50 | 2.23 | 86.4 | 1227 |
| 100 | 2.30 | 88.6 | 557 |
| 200 | 2.38 | 87.8 | 417 |
| 300 | 2.31 | 87.6 | 412 |
| 400 | 2.22 | 85.0 | 401 |
| 500 | 2.08 | 82.4 | 416 |
| 600 | 2.23 | 84.8 | 303 |
| 700 | 2.36 | 86.2 | 226 |
| 800 | 2.71 | 87.0 | 163 |
| 900 | 2.61 | 86.8 | 155 |

**Cuadro 5. Resultados con el algoritmo propuesto para Letter con 19,500 objetos de entrenamiento y 500 prueba, variando el valor de $s$**

el caso del algoritmo ICE, los resultados del algoritmo propuesto también se muestran superiores en cuanto a tiempo de procesamiento y porcentaje de aciertos, además de que los AD generados por el algoritmo propuesto son de menor tamaño en comparación con los de ICE, excepto para el valor de $s = 50$.

Para la segunda prueba con el conjunto de datos Letter, se utilizó validación cruzada de 5 pliegues y un valor de $s = 100$. Los resultados obtenidos se muestran en el cuadro 6.

| Algoritmo | Tiempo | Porcentaje de aciertos | | Tamaño | |
|---|---|---|---|---|---|
| Propuesto | 2.97 | 87.6 | | 522.2 | |
| ICE | 69.01 | 67.6 | | 587.3 | |
| | | Completo | Podado | Completo | Podado |
| C4.5 | 6.08 | 87.4 | 87.4 | 2372.2 | 2163.4 |

**Cuadro 6. Resultados obtenidos para la base de datos Letter**

En el cuadro 6, el algoritmo propuesto obtuvo un porcentaje de aciertos superior a C4.5 e ICE, en menor tiempo y generando un AD de un tamaño 4 veces menor en comparación con el generado por C4.5, además de que también es menor su tamaño con respecto al generado por ICE.

### 4.2.2. Poker

Poker [36] se define dentro del repositorio de datos en dos muestras, una de entrenamiento de 25,010 objetos y una de control de 1'000,000 de objetos. Sin embargo, para mostrar el comportamiento que tiene el algoritmo en grandes conjuntos de datos, se utilizó la muestra más grande para obtener el conjunto de entrenamiento. Poker está definido por 10 atributos numéricos (todos enteros) y sus objetos están distribuidos en

10 clases. Sin embargo, 2 clases contienen a la mayoría de los objetos de la muestra (cerca del 92 %), por lo que se decidió sólo utilizar estas clases para realizar los experimentos.

A partir de esa muestra (sólo de la clase 0 y de la clase 1) se obtuvieron diversos conjuntos de entrenamiento, se fueron construyendo conjuntos de diferente tamaño, y de cada tamaño se generaron 5 diferentes particiones, el tamaño del conjunto de entrenamiento se fue incrementando, para observar el comportamiento del algoritmo propuesto respecto al tamaño de la muestra. Con cada partición creada, se utilizó un conjunto de prueba de 400,000 objetos, a fin de observar el porcentaje de aciertos que tiene el AD con un conjunto de prueba grande. Se aplicó el algoritmo con un valor de $s = 100$, el cuadro 7 presenta los tiempo de ejecución empleados por los algoritmos y el cuadro 8 los resultados obtenidos en porcentaje de aciertos y tamaño de los AD generados.

| Tamaño CE | Alg. Propuesto | C4.5 | ICE |
|---|---|---|---|
| 20000 | 154.11 | 14.20 | 15.02 |
| 40000 | 152.85 | 27.75 | 25.14 |
| 50000 | 129.32 | 37.98 | 40.9 |
| 100000 | 135.22 | 97.34 | 75.3 |
| 150000 | 141.94 | 180.75 | 113.85 |
| 200000 | 138.79 | 216.75 | 155.25 |
| 250000 | 123.08 | 366.01 | 202.08 |
| 300000 | 147.29 | 538.35 | 244.26 |
| 350000 | 156.35 | 892.89 | 268.10 |
| 400000 | 146.92 | 800.91 | 319.84 |
| 450000 | 135.34 | 1223.36 | 365.99 |
| 500000 | 134.38 | 1583.18 | 420.68 |

**Cuadro 7. Tiempos de ejecución para el conjunto de datos Poker**

| Tamaño CE | Alg. Propuesto | | C4.5 | | | | ICE | |
|---|---|---|---|---|---|---|---|---|
| | Porcentaje de aciertos | Tamaño | Porcentaje de aciertos | | Tamaño | | Porcentaje de aciertos | Tamaño |
| | | | Completo | Podado | Completo | Podado | | |
| 20000 | 56.4 | 131.4 | 55.5 | 56.0 | 4375.8 | 3068.2 | 56.7 | 513.2 |
| 40000 | 57.3 | 267.0 | 55.0 | 55.8 | 9109.8 | 6241.0 | 58.3 | 849.8 |
| 50000 | 57.2 | 365.4 | 56.0 | 56.9 | 11727.8 | 7891.8 | 58.3 | 992.1 |
| 100000 | 57.9 | 721.4 | 58.3 | 60.0 | 22796.2 | 13700.6 | 58.4 | 1263.8 |
| 150000 | 58.5 | 1044.6 | 57.6 | 58.9 | 31406.2 | 19610.6 | 58.6 | 1936.2 |
| 200000 | 58.7 | 1447.4 | 57.6 | 60.9 | 31816.6 | 14494.6 | 59.0 | 2579.0 |
| 250000 | 58.9 | 1873.0 | 59.7 | 62.6 | 44521.0 | 22292.6 | 61.1 | 3051.0 |
| 300000 | 59.4 | 2102.2 | 57.9 | 63.0 | 54766.2 | 19680.6 | 61.4 | 3791.8 |
| 350000 | 59.7 | 2431.8 | 59.5 | 65.0 | 80479.4 | 28895.0 | 60.9 | 4403.8 |
| 400000 | 59.7 | 2883.8 | 56.3 | 57.9 | 63350.2 | 37296.6 | 61.5 | 5066.5 |
| 450000 | 59.6 | 3373.4 | 60.9 | 65.6 | 86870.6 | 35741.4 | 62.9 | 5408.5 |
| 500000 | 60.0 | 3742.2 | 63.3 | 69.0 | 104955.4 | 37080.6 | 63.2 | 5986.5 |

**Cuadro 8. Porcentaje de aciertos y Tamaño del AD generado para el conjunto de datos Poker**

Para poder apreciar mejor los resultados obtenidos con este conjunto de datos, las figuras 15 - 17 presentan los resultados de manera gráfica.

Basándose en los resultados se observa que el algoritmo propuesto obtiene un porcentaje de aciertos similar a la de los algoritmos C4.5 e ICE. El tiempo de ejecución empleado por nuestro algoritmo para grandes conjuntos de entrenamiento es considerablemente menor y es estable en su comportamiento en comparación con los dos algoritmos, además el tamaño del AD generado por el algoritmo propuesto es también más pequeño, aún con el procedimiento de poda que C4.5 realiza.

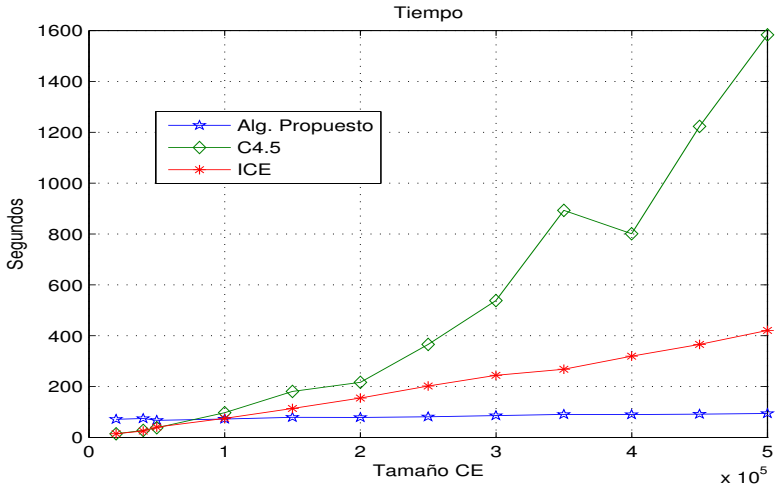


**Figura 15. Tiempo de ejecución con Poker, variando el tamaño del conjunto de entrenamiento de 20000 a 500000 objetos.**

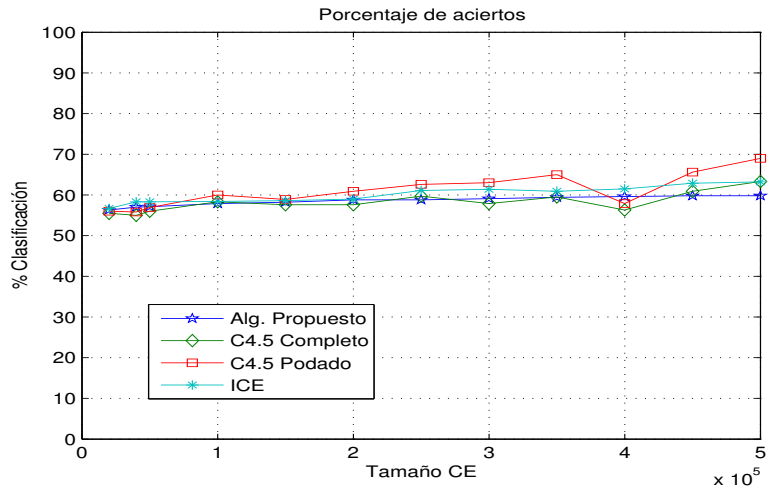


**Figura 16. Porcentaje de aciertos con Poker, variando el tamaño del conjunto de entrenamiento de 20000 a 500000 objetos.**

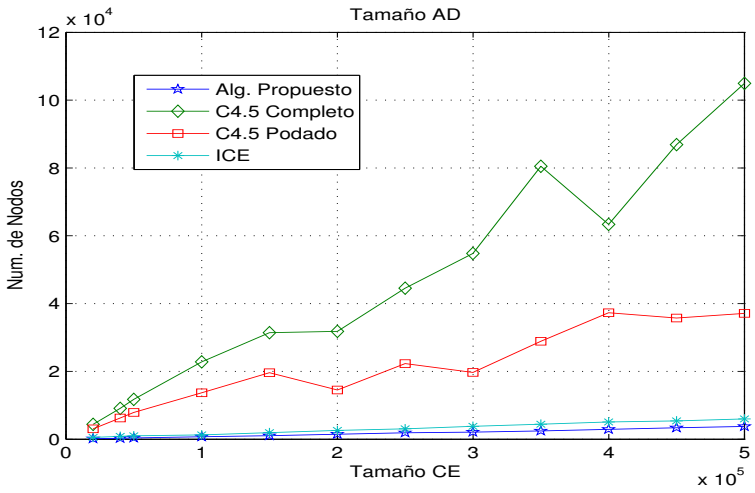


**Figura 17. Tamaño del AD con Poker, variando el tamaño del conjunto de entrenamiento de 20000 a 500000 objetos.**

### 4.2.3. SpecObj

El tercer experimento se realizó con una base de datos denominada SpecObj [37] obtenida del Sloan Digital Sky Survey. Las clases que forman a este conjunto de datos contienen descripciones de los espectros que generan algunos objetos que pueden ser observados en el cielo, como por ejemplo galaxias, estrellas, quasares, quasares con alto corrimiento al rojo, estrellas tardías y objetos desconocidos. En total se tienen 884,054 objetos, descritos por 5 atributos numéricos (todos reales). Se utiliza, al igual que en el experimento anterior, un conjunto de prueba de 400,000 objetos y se van generando conjuntos de entrenamiento de diferentes tamaños. Para el algoritmo propuesto se uso un valor de $s = 100$. Los tiempos de ejecución empleados por el algoritmo se muestran en el cuadro 9 y los resultados de porcentaje de aciertos y tamaño del AD en el cuadro 10.

| Tamaño CE | Alg. Propuesto | C4.5 | ICE |
|---|---|---|---|
| 20000 | 81.12 | 16.40 | 1880.51 |
| 40000 | 94.08 | 23.69 | 3363.97 |
| 50000 | 96.67 | 21.60 | 9956.70 |
| 100000 | 98.65 | 43.45 | - |
| 150000 | 103.31 | 69.06 | - |
| 200000 | 111.69 | 100.02 | - |
| 250000 | 115.28 | 130.81 | - |
| 300000 | 118.46 | 171.32 | - |
| 350000 | 122.94 | 214.13 | - |
| 400000 | 126.89 | 263.37 | - |
| 450000 | 131.51 | 321.60 | - |

**Cuadro 9. Tiempos de ejecución para el conjunto de datos SpecObj**

Las figuras 18 - 20, muestran en forma gráfica los resultados obtenidos para este conjunto de datos.
Con este conjunto de datos, el comportamiento del algoritmo propuesto es muy parecido al que tuvo con

| Tamaño CE | Alg. Propuesto | | C4.5 | | | | ICE | |
|---|---|---|---|---|---|---|---|---|
| | Porcentaje de aciertos | Tamaño | Porcentaje de aciertos | | Tamaño | | Porcentaje de aciertos | Tamaño |
| | | | Completo | Podado | Completo | Podado | | |
| 20000 | 90.1 | 417.3 | 88.8 | 89.3 | 1861.1 | 1571.3 | 71.3 | 115 |
| 40000 | 89.6 | 650.0 | 91.6 | 91.8 | 2471.3 | 2037.8 | 82.5 | 203 |
| 50000 | 89.5 | 756.0 | 91.7 | 91.8 | 2588.6 | 2061.0 | 85.4 | 221 |
| 100000 | 90.0 | 1481.2 | 92.1 | 92.2 | 4542.2 | 3674.6 | - | - |
| 150000 | 90.3 | 2183.2 | 92.4 | 92.6 | 6350.2 | 5051.0 | - | - |
| 200000 | 90.8 | 2888.8 | 92.8 | 93.0 | 8257.0 | 6505.8 | - | - |
| 250000 | 90.8 | 3555.2 | 92.9 | 93.1 | 9985.0 | 7815.8 | - | - |
| 300000 | 91.1 | 4268.8 | 93.1 | 93.4 | 11547.8 | 8980.6 | - | - |
| 350000 | 91.0 | 4929.4 | 93.0 | 93.2 | 13075.4 | 10215.8 | - | - |
| 400000 | 90.9 | 5618.0 | 93.1 | 93.4 | 14677.4 | 11423.0 | - | - |
| 450000 | 91.1 | 6256.4 | 93.2 | 93.4 | 16263.4 | 12632.2 | - | - |

**Cuadro 10. Porcentaje de aciertos y Tamaño del AD generado para el conjunto de datos SpecObj**

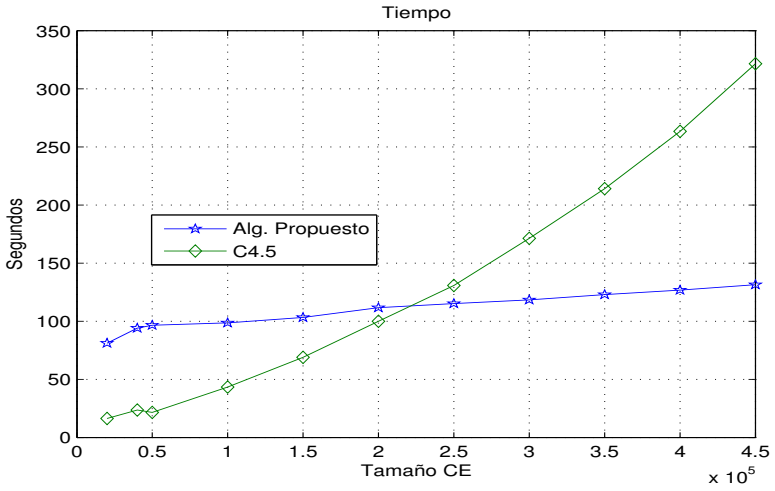


**Figura 18. Tiempo de ejecución con SpecObj, variando el tamaño del conjunto de entrenamiento de 20000 a 450000 objetos.**

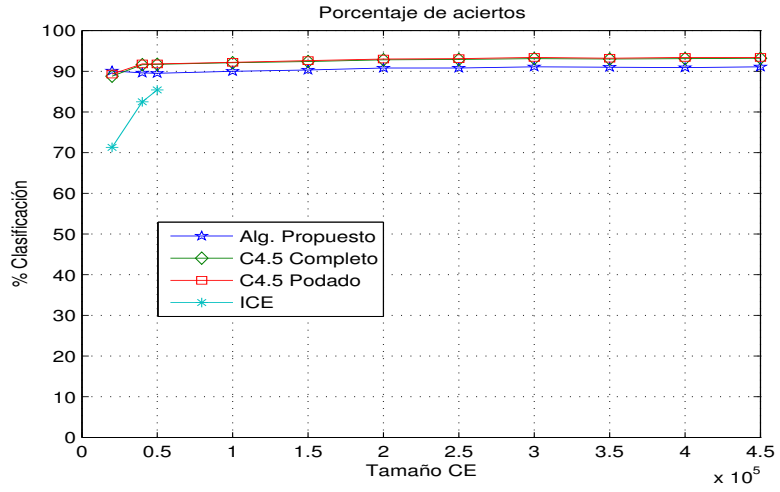


**Figura 19. Porcentaje de aciertos con SpecObj, variando el tamaño del conjunto de entrenamiento de 20000 a 450000 objetos.**

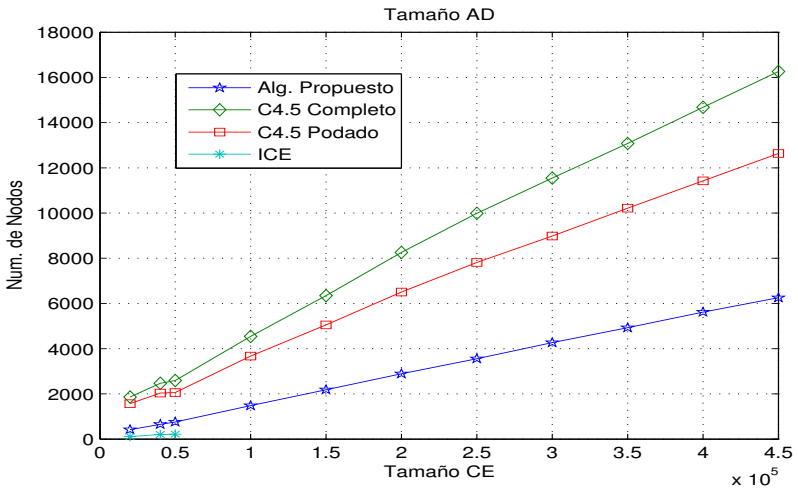


**Figura 20. Tamaño del AD con SpecObj, variando el tamaño del conjunto de entrenamiento de 20000 a 450000 objetos.**

la base de datos Poker. Con respecto a C4.5 el porcentaje de aciertos del algoritmo es ligeramente inferior, sin embargo, en cuanto al tiempo y al tamaño del AD, el algoritmo propuesto obtiene mejores resultados. Para el algoritmo ICE se realizaron las pruebas con los primeros tamaños del CE, los resultados obtenidos muestran que el algoritmo requiere de un gran tiempo de procesamiento en comparación con el algoritmo propuesto, además de que el porcentaje de aciertos del algoritmo propuesto fue superior a ICE.

### 4.2.4. Análisis de los resultados obtenidos

En los experimentos se utilizó la media para obtener los objetos representativos de cada arco de los nodos internos, sin embargo este no es el único procedimiento que se puede aplicar para obtenerlos, por lo que se buscarán diversas representaciones que puedan ser aplicadas para este fin. Además, también se buscarán otras estrategias que permitan la clasificación de nuevos objetos, una vez que éstos lleguen a alguna hoja, aunque la aplicación de la regla del vecino más cercano para esta tarea resultó eficiente y se han obtenido resultados del porcentaje de aciertos competitivos.

Una característica que se pudo observar con los experimentos realizados es el comportamiento que tuvieron el algoritmo propuesto y C4.5 cuando el número de atributos, que describen al conjunto de entrenamiento, varía. El tiempo de ejecución de C4.5 es alrededor de 5 veces más cuando el número de atributos se duplica, mientras que en el algoritmo propuesto, el tiempo de ejecución es muy similar. Sin embargo, es necesario realizar más experimentos con conjuntos de datos que tengan más atributos.

ICE mostró que sus tiempos de ejecución son mayores que los del algoritmo propuesto cuando se procesan grandes conjuntos de datos y que el porcentaje de aciertos obtenido por él es similar al de nuestro algoritmo.

Acerca del comportamiento del algoritmo propuesto cuando se varía el valor del parámetro $s$, se observó que para el caso del tamaño del árbol, es claro que conforme vaya aumentando el valor de $s$, disminuye el tamaño del árbol, pero para el tiempo de ejecución y del porcentaje de aciertos del algoritmo, no se ha determinado con exactitud que comportamiento tiene el algorimo, por lo que es necesario realizar más experimentos.

Con base en estos resultados obtenidos, se puede observar que el algoritmo propuesto obtiene resultados competitivos respecto a la calidad de clasificación, además de que en los experimentos realizados el tiempo de ejecución empleado es mucho menor que el de los algoritmos C4.5 e ICE cuando se utilizan grandes conjuntos de datos.

## 5. Conclusiones

Los árboles de decisión son una herramienta útil para resolver problemas de clasificación supervisada. En la actualidad se han desarrollado diversos algoritmos que generan árboles de decisión, incluso algunos que trabajan con grandes conjuntos de datos, sin embargo, los algoritmos de generación de árboles de este tipo presentan algunas restricciones, como por ejemplo: el espacio que ocupan, el número de veces que tienen que recorrer el conjunto de entrenamiento para construir el árbol o el no tener la capacidad de actualizarse si el conjunto de datos tiene un flujo continuo de información.

En el presente trabajo se propone desarrollar un algoritmo de generación de árboles de decisión para grandes conjuntos de datos que supere estas restricciones, es decir, un algoritmo que sea incremental, que no necesite almacenar todo el conjunto de entrenamiento en memoria para poder construir el árbol y que accese la menor cantidad de veces posible al conjunto. El algoritmo propuesto preliminarmente ha sido

diseñado para trabajar con conjuntos de datos numéricos, pero se plantea aplicar ténicas que nos permitan trabajar con conjuntos de datos mezclados.

Los resultados experimentales que se han realizado, muestran que el algoritmo propuesto resulta competitivo en porcentaje de aciertos y genera árboles compactos en un tiempo razonable. Estos resultados son alentadores y muestran que es factible alcanzar los objetivos propuestos es esta tesis.

## Referencias


[1] Ruiz Shulcloper, R., Guzmán Arenas, A., y Martínez Trinidad, J.F.: Enfoque Lógico Combinatorio al Reconocimiento de Patrones. Instituto Politécnico Nacional, 1999.

[2] Mitchell, T.: Machine Learning. McGraw Hill, 1997.

[3] Quinlan, J. R.: C4.5: Programs for Machine Learning. Morgan Kaufmann, San Mateo, C.A. (1993).

[4] Breiman, L., Friedman, J., and Olshen, R.: Classification and Regression Trees. Wadsworth International Group, 1984.

[5] Vanichsetakul, N. and Loh, Wei-Yin: Tree-structured classification via generalized discriminant analysis. Journal of the American Statistical Association, 83-403 (1988) 715–728.

[6] Shih, Yu-Shan and Loh, Wei-Yin: Split selection methods for classification trees. Statistica Sinica, 7-4 (1997) 815–840.

[7] Shou Chih, C., Hsing Kuo, P. and Yuh Jye, L.: Model trees for classification of hybrid data types. In Intelligent Data Engineering and Automated Learning - IDEAL: 6th International Conference. (2005) 32–39.

[8] Pérez, J., Muguerza, J., Arbelaitz, O., Gurrutxaga, I., and Martín, J.: Combining multiple class distribution modified subsamples in a single tree. Pattern Recognition Letters. 28-4 (2007) 414–422.

[9] Utgoff, P.E.: Incremental induction of decision trees. Machine Learning. 4 (1989) 161–186.

[10] Utgoff, P.E.: An improved algorithm for incremental induction of decision trees. In Proc. 11th International Conference on Machine Learning. (1994) 318–325.

[11] Utgoff, P.E., Berkman, N.C. and Clouse, J.A.: Decision tree induction based on efficient tree restructuring. Machine Learning. 29-5 (1997) 5–44.

[12] Gama, J. and Medas, P.: Learning decision trees from dynamic data streams. Journal of Universal Computer Science. 11-8 (2005) 1353–1366.

[13] Gama, J., Medas, P. and Rocha, R.: Forest trees for on-line data. In SAC '04: Proceedings of the 2004 ACM symposium on Applied computing. (2004) 632–636.

[14] Jin, R. and Agrawal, G.: Efficient decision tree construction on streaming data. In Proceedings of the ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. (2003) 571–576.

[15] Janikow, C.Z.: Fuzzy decision trees: Issues and methods. IEEE Transactions on Systems, Man and Cybernetics - Part B: Cybernetics. 28-1 (1998) 1–14.

[16] Pedrycz, W. and Sosnowski.: Genetically optimized fuzzy decision trees. IEEE Transactions on Systems, Man and Cybernetics - Part B: Cybernetics. 35-3 (2005) 633–641.

[17] Wang, X., Nauck, D. and Spott. M.: Intelligent data analysis with fuzzy decision trees. Soft Computing - A Fusion of Foundations, Methodologies and Applications. 11 (2007) 439–457.

[18] Utgoff, P.E. and Brodley, C.E.: An incremental method for finding multivariate splits for decision trees. In Proc.7th International Conference on Machine Learning. (1990) 58–65.

[19] Utgoff, P.E. and Brodley, C.E.: Multivariate decision trees. Machine Learning. 19-1 (1995) 45–77.

[20] Brodley, C.E. and Utgoff, P.E.: Linear machine decision trees. Technical Report TR-91-10, University of Massachusetts, Department of Computer and Information Science, Amherst, MA (1991).

[21] Llora, X. and Wilson, S.: Mixed decision trees: Minimizing knowledge representation bias in LCS. Genetic and Evolutionary Computation. GECCO. 3103 (2204) 797–809.

[22] Hsin Wei, C., Chen Sen, O. and Shie Jue, L.: Improved c-fuzzy decision trees. In IEEE International Conference on Fuzzy Systems, (2006) 1763–1768.

[23] Pedrycz, W. and Sosnowski.: C-fuzzy decision trees. IEEE Transactions on Systems, Man and Cybernetics - Part C: Applications and reviews. 35-4 (2205) 498–511.

[24] Mehta, M., Agrawal, R. and Rissanen, J.: SLIQ: A fast scalable classifier for data mining. In Proc. 5th International Conference Extending Database Technology (EDBT), Avignon, France. (1996) 18–32.

[25] Shafer, J.C., Agrawal, R. and Mehta, M.: SPRINT: A scalable parallel classifier for data mining. In Proc. 22nd International Conference Very Large Databases. (1996) 544–555.

[26] Alsabti, K., Ranka, S. and Singh, V.: CLOUDS: A decision tree classifier for large datasets,. In Proc. Conference Knowledge Discovery and Data Mining (KDD'98). (1998) 2–8.

[27] Gehrke, J., Ramakrishnan, R. and Ganti, V.: Rainforest - a framework for fast decision tree construction of large datasets. Data Mining Knowledge Discovery. 4-2/3 (1998) 127–162.

[28] Gehrke, J., Ganti, V., Ramakrishnan, R. and Loh, W.: BOAT - optimistic decision tree construction. In Proc. of the ACM SIGMOD Conference on Management of Data. (1999) 169–180.

[29] Yoon, H., Alsabti, K. and Ranka, S.: Tree-based incremental classification for large datasets. Technical Report TR-99-013, CISE Department, University of Florida, Gainesville, FL. 32611. (1999).

[30] Taner O. and Dikmen O.: Parallel univariate decision trees. Pattern Recognition Letters. 28-7 (2007) 825–832.

[31] Rokach L. and Maimon O.: Top-down induction of decision trees classifiers - a survey. IEEE Transactions on Systems, Man and Cybernetics. 35-4 (2205) 476–487.

[32] Safavian, S.R. and Landgrebe, D.: A survey of decision tree classifier methodology. IEEE Transactions on Systems, Man and Cybernetics. 21-3 (1991) 660–674.

[33] Kohavi, R. and Quinlan, J.R.: Decision-tree discovery. Will Klosgen and Jan M. Zytkow, editors. Oxford University Press, 2002.

[34] Kuncheva, L.: Combining Pattern Classifiers. John Wiley and Sons. (2004).

[35] Han, J. and Kamber. M.: Data Mining, Concepts and Techniques. Morgan Kaufmann Publishers. (2001).

[36] Asuncion, A. and Newman, D.J.: UCI machine learning repository. http://www.ics.uci.edu/»mlearn/MLRepository.html, University of California, Irvine, School of Information and Computer Sciences. (2007).

[37] Sloan Digital Sky Survey. http://www.sdss.org/.