UNIVERSIDAD COMPLUTENSE MADRID

Ingeniería del conocimiento

Práctica 2: Algoritmo ID3

Álvaro del Campo Gragera
Juan Montero Gómez

16/04/2024

Índice

1. Introducción

El objetivo de la práctica es implementar el Algoritmo ID3. Empezando por el lenguaje y herramientas usadas, se explicará cómo se calcula, yendo paso por paso por cada una de las funciones que lo implementan. Además, se describirá la estructura del proyecto, incluyendo la organización de archivos. También se presentarán las dos funcionalidades opcionales: implementar todos los niveles de recursividad y comprobar el correcto funcionamiento para los ejemplos de la tabla. Finalmente, se proporcionará un manual de usuario para ejecutar el programa y la solución obtenida tras la ejecución.

2. Detalles de implementación

El Algoritmo ID3 se ha implementado en el lenguaje de programación C++, utilizando el entorno de desarrollo Visual Studio Code.

Primero se leen los dos archivos proporcionados, *AtributosJuego.txt* y *Juego.txt*, mediante las funciones **leerAtributos** y **leerEjemplos** respectivamente. La información leída se guardará en los structs *Atributo* y *Ejemplo*. El primer atributo solo contiene un string que guarda el nombre del atributo (*Temperatura*, *Humedad*…), aunque más tarde se eliminará el último atributo, *Jugar*, puesto que no es un atributo como tal sino la decisión final. El segundo atributo contiene un string que guarda la decisión final (en este caso *sí* o *no*) y un unordered_map que asocia cada atributo con su valor correspondiente en cada ejemplo (se pueden ver los structs en la imagen 1). Finalmente, creamos dos vectores que contienen la lista de los diferentes *Atributos* y *Ejemplos*. Una vez leídos los datos, se muestran por pantalla, tal y como se puede apreciar en la imagen 2.

```
// Estructura para un ejemplo
struct Ejemplo {
    unordered_map<string, string> atributos;
    string decision;
};

// Estructura para un atributo
struct Atributo {
    string nombre;
};
```

Imagen 1: structs de Ejemplo y Atributo

```
Construyendo arbol de decision...
 TiempoExterior    Temperatura       Humedad        Viento      Decision
        soleado       caluroso          alta         falso            no
        soleado       caluroso          alta        verdad            no
        nublado       caluroso          alta         falso            si
       lluvioso       templado          alta         falso            si
       lluvioso           frio        normal         falso            si
       lluvioso           frio        normal        verdad            no
        nublado           frio        normal        verdad            si
        soleado       templado          alta         falso            no
        soleado           frio        normal         falso            si
       lluvioso       templado        normal         falso            si
        soleado       templado        normal        verdad            si
        nublado       templado          alta        verdad            si
        nublado       caluroso        normal         falso            si
       lluvioso       templado          alta        verdad            no
```

Imagen 2: datos leídos mostrados en pantalla.

A continuación, se llama a la función **ID3**, que implementa el algoritmo del mismo nombre para construir un árbol de decisión de manera recursiva. Comienza evaluando el caso base, si todos los ejemplos en un nodo tienen la misma decisión. En caso de ser así, la rama del nodo termina (ver imagen 3).

```cpp
// Función principal para el algoritmo ID3
void ID3(vector<Ejemplo>& ejemplos, vector<Atributo>& atributos, int nivel) {
    // Caso base si todos tienen la misma decision
    bool mismaDecision = true;
    string decision = ejemplos[0].decision;
    for (auto& ejemplo : ejemplos) {
        if (ejemplo.decision != decision) {
            mismaDecision = false;
            break;
        }
    }
    if (mismaDecision) {
        for (int i = 0; i < nivel; ++i) {
            cout << "\t";
        }
        cout << "Entonces Decision = " + decision << endl;
        return;

    }
```

Imagen 3: caso base del algoritmo ID3

En caso contrario, se selecciona el mejor atributo para dividir los ejemplos restantes utilizando la función **seleccionarMejorAtributo** (ver imagen 4). Esta función genera recursivamente subárboles para cada valor único del atributo seleccionado llamando a la función **dividirEjemplos**, y va guardando el mejor atributo mediante el cálculo de la entropía. Pero vayamos por partes con el desempeño de cada función.

```
// Función para seleccionar el mejor atributo para dividir
string seleccionarMejorAtributo(vector<Ejemplo>& ejemplos, vector<Atributo>& atributos) {
    double mejorGanancia = 1.0;
    string mejorAtributo;

    for (auto& atributo : atributos) {
        map<string, vector<Ejemplo>> subconjuntos = dividirEjemplos(ejemplos, atributo.nombre);
        double ganancia = 0.0;
        for (auto& pair : subconjuntos) {
            double peso = (double)pair.second.size() / ejemplos.size();
            ganancia += peso * calcularEntropia(pair.second);
        }
        if (ganancia < mejorGanancia) {
            mejorGanancia = ganancia;
            mejorAtributo = atributo.nombre;
        }
    }
    return mejorAtributo;
}
```

Imagen 4: función para seleccionar el mejor atributo

En **dividirEjemplos**, la función guardará en un subconjunto todos los ejemplos que tengan un atributo concreto (ver imagen 5). Por ejemplo, en caso de que se realice la llamada con el atributo *Humedad*, se guardará en el mapa que cuando *Humedad* tiene el valor *alta*, los ejemplos seleccionados serán todos aquellos del vector ejemplos cuyo atributo *Humedad* sea *alta*. Después, devuelve estos subconjuntos.

```
// Función para dividir un conjunto de ejemplos según un atributo dado
map<string, vector<Ejemplo>> dividirEjemplos(vector<Ejemplo>& ejemplos, string atributo) {
    map<string, vector<Ejemplo>> subconjuntos;
    for (auto& ejemplo : ejemplos) {
        string valor_atributo = ejemplo.atributos.at(atributo);
        subconjuntos[valor_atributo].push_back(ejemplo);
    }
    return subconjuntos;
}
```

Imagen 5: función para dividir ejemplos por valores de un atributo concreto

Ahora, para cada subconjunto se realiza el cálculo del mérito (ver imagen 6). Este consiste en el sumatorio de la multiplicación de la relación entre el número de ejemplos con un valor concreto del atributo y el número total de ejemplos (la variable “peso” de la función, se puede ver en la imagen 4) por la entropía correspondiente, que se calcula en la función **calcularEntropía**. Después de calcular el mérito de cada atributo, se compara con el menor acumulado hasta el momento, y se guarda el menor, ya que el mejor atributo es el que tenga un menor mérito.

$$a_m\text{: mérito } (a_m) = \sum_{i=1}^{n} r_i \times \textit{infor}\,(p_i, n_i)$$

Imagen 6: operación para calcular el mérito

Para el cálculo de la entropía, se guarda el número de ejemplos con decisión positiva o negativa en el map *decisionCounts*. Después, para cada valor de la decisión final, se calcula la entropía, que es la suma de la relación entre el número de ejemplos con decisión positiva o negativa con respecto al total de ejemplos multiplicado por el logaritmo en base dos de esa misma relación (ver imagen 7).

```
// Función para calcular la entropía
double calcularEntropia(vector<Ejemplo>& ejemplos) {
    map<string, int> decisionCounts;
    for (auto& ejemplo : ejemplos) {
        decisionCounts[ejemplo.decision]++;
    }
    double entropia = 0.0;
    for (auto& pair : decisionCounts) {
        double probabilidad = (double)pair.second / ejemplos.size();
        entropia -= probabilidad * log2(probabilidad);
    }
    return entropia;
}
```

Imagen 7: función para calcular la entropía

Volviendo a la función ID3, tras haber determinado cuál es el mejor atributo gracias a las funciones **seleccionarMejorAtributo**, **dividirEjemplos** y **calcularEntropia**, se creará un vector *nuevosAtributos* que contendrá todos los atributos a excepción del mejor encontrado previamente, puesto que el árbol partirá del mejor atributo. Después de esto, gracias a la función **dividirEjemplos** hacemos llamadas recursivas a la función **ID3** para cada valor del mejor atributo. A la hora de imprimir los resultados finales, se hace uso del atributo *nivel*, que indica el nivel del árbol, para imprimir espacios según dicho nivel, para una mejor comprensión de los niveles del árbol (ver imagen 8).

```
// Seleccionar el mejor atributo para dividir
string mejorAtributo = seleccionarMejorAtributo(ejemplos, atributos);

// Recursión para los subárboles
vector<Atributo> nuevosAtributos = atributos;
for (auto it = nuevosAtributos.begin(); it != nuevosAtributos.end(); ++it) {
    if (it->nombre == mejorAtributo) {
        nuevosAtributos.erase(it);
        break;
    }
}

map<string, vector<Ejemplo>> subconjuntos = dividirEjemplos(ejemplos, mejorAtributo);
for (auto& pair : subconjuntos) {
    for (int i = 0; i < nivel; ++i) {
        cout << "\t";
    }
    cout << "Si " << mejorAtributo << " = " << pair.first << endl;
    ID3(pair.second, nuevosAtributos, nivel + 1);
}
```

*Imagen 8: llamada recursiva a **ID3***

Respecto a las posibilidades de ampliación, tenemos dos. La primera es implementar todos los niveles de recursividad, que gracias al caso base y a la llamada de forma recursiva a la función **ID3**, lo hemos conseguido. La segunda es comprobar el correcto funcionamiento del algoritmo para los ejemplos de la tabla, para lo que hemos implementado la función **comprobarConclusion**, que toma dos vectores de *ejemplos*, uno que representa los ejemplos reales y otro que contiene las conclusiones generadas por el algoritmo ID3. Itera sobre cada ejemplo y compara sus atributos con los de cada conclusión. Si encuentra una coincidencia completa, verifica si las decisiones coinciden. Si todas las conclusiones son correctas para todos los ejemplos, devuelve *true*; de lo contrario, devuelve *false*, indicando que al menos una conclusión es incorrecta (ver imagen 9).

```cpp
bool comprobarConclusion(const vector<Ejemplo>& ejemplos, const vector<Ejemplo>& conclusiones) {
    bool correcto = true;
    int nEjemplos = ejemplos.size();
    // para cada ejemplo
    int idEjemplo = 0;
    for (auto &ejemplo : ejemplos) {
        idEjemplo++;
        cout <<  endl;
        cout << "Ejemplo " << idEjemplo << endl;
        cout << "----------";
        // se mira cada conclusion
        for(int i = 0; i < conclusiones.size(); i++) {
            cout << "\nConclusion " << i+1 << ": " << endl;

            //Si todos los atributos de una conclusión aciertan
            int nAtributosC = conclusiones[i].atributos.size();
            for(auto& atributo : conclusiones[i].atributos) {
                auto atributo1 = ejemplo.atributos.find(atributo.first);
                if ( atributo1 != ejemplo.atributos.end()) {
                    if (atributo1->second == atributo.second) {
                        nAtributosC--;
                    }
                    cout << "\tAtributo conclusion " << i+1 << ": " << atributo.second << " | ";
                    cout << "Atributo ejemplo " << idEjemplo << ": " << ejemplo.atributos.find(atributo.first)->second << endl;
                }
            }
            if (nAtributosC <= 0) {
                cout << "\nDecision conclusion " << i+1 << ": " << conclusiones[i].decision << " | ";
                cout << "Decision ejemplo " << idEjemplo << ": " << ejemplo.decision;
                if (ejemplo.decision == conclusiones[i].decision) {
                    cout << "\nCORRECTO" << endl;
                    cout << "Se cumple conclusion (" << i+1 << ") en el ejemplo (" << idEjemplo << ")" << endl << endl;
                    break;
                }
                else {
                    cout << "\nFALSO" << endl;
                    cout << "No se cumple ninguna conclusion en el ejemplo (" << idEjemplo << ")" << endl << endl;
                    correcto = false;
                }
            }
        }
    }
    return correcto;
}
```

Imagen 9: función para comprobar el funcionamiento correcto del algoritmo

3. Manual de usuario

El primer paso es ejecutar el fichero, que se puede ejecutar tanto en Visual Studio como en Visual Studio Code. En caso de hacerlo en Visual Studio, la opción más sencilla, tan solo hay que crear un proyecto en el que se encuentren los tres archivos y ejecutar el programa con el botón de la flecha verde en la parte superior. En caso de hacerlo en Visual Studio Code, el proceso es un poco más laborioso. Tras haber creado un proyecto con los tres archivos en el mismo directorio, se tendrá que abrir la terminal (atajo con `CTRL + Ñ`) e introducir el siguiente comando (asumiendo que el sistema tiene instalado el compilador g++):

```
g++ -o main main.cpp
```

Antes de ejecutar el fichero en terminal, se deben de proporcionar los archivos que contengan los datos que se quieran usar. En *AtributosJuego.txt* deben de estar los atributos, separados por comas, mientras que en Juego.txt deben estar los valores de los atributos, separados también por comas. Ahora sí, para ejecutar el fichero en terminal, se puede hacer con el siguiente comando:

```
./main.exe
```

Al ejecutar el programa, se comenzará a construir el árbol de decisión basado en los datos proporcionados en los archivos de entrada. Se mostrará una tabla que representa los datos de entrada, donde cada fila corresponde a un ejemplo y cada columna corresponde a un atributo, seguido de la decisión. Después de mostrar la tabla, el programa construirá el árbol de decisión utilizando el algoritmo ID3. El árbol de decisión se mostrará en la salida estándar, indicando los atributos seleccionados en cada nodo del árbol y los valores posibles para esos atributos. La construcción del árbol de decisión continuará hasta que se alcance un criterio de parada.

4. Ejemplo de ejecución

Se puede ver la solución del algoritmo para este caso en la imagen 10.

```
Construyendo arbol de decision...
 TiempoExterior    Temperatura        Humedad         Viento       Decision
        soleado       caluroso           alta          falso             no
        soleado       caluroso           alta         verdad             no
        nublado       caluroso           alta          falso             si
       lluvioso       templado           alta          falso             si
       lluvioso           frio         normal          falso             si
       lluvioso           frio         normal         verdad             no
        nublado           frio         normal         verdad             si
        soleado       templado           alta          falso             no
        soleado           frio         normal          falso             si
       lluvioso       templado         normal          falso             si
        soleado       templado         normal         verdad             si
        nublado       templado           alta         verdad             si
        nublado       caluroso         normal          falso             si
       lluvioso       templado           alta         verdad             no

Si TiempoExterior = lluvioso
        Si Viento = falso
                Entonces Decision = si
        Si Viento = verdad
                Entonces Decision = no
Si TiempoExterior = nublado
        Entonces Decision = si
Si TiempoExterior = soleado
        Si Humedad = alta
                Entonces Decision = no
        Si Humedad = normal
                Entonces Decision = si
```

Imagen 10: salida por consola de la ejecución del algoritmo ID3

A continuación en la imagen 11, comprueba la solución con todos los ejemplos de la tabla:

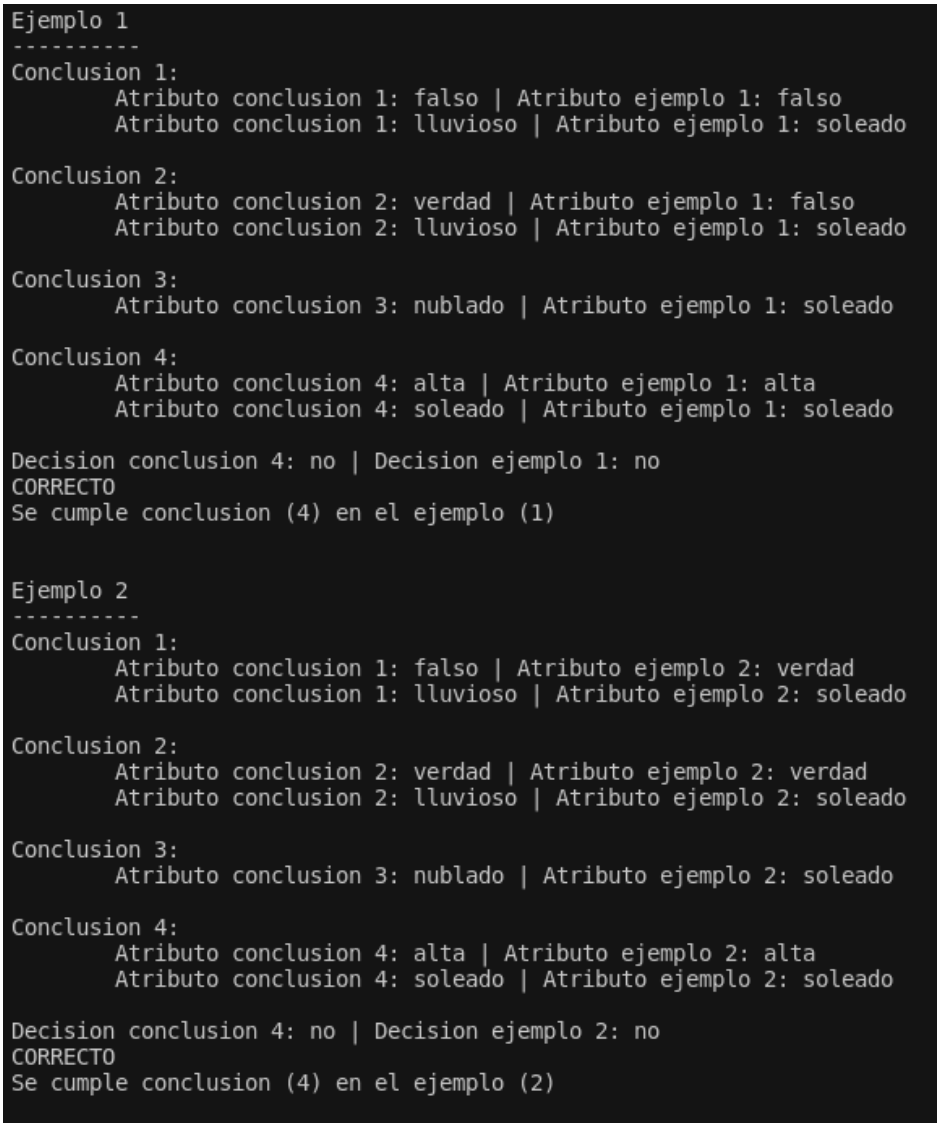

Imagen 11: salida por consola de la comprobación de los dos primeros ejemplos del algoritmo ID3.

Finalmente, indica si la solución es correcta según la tabla de ejemplos:

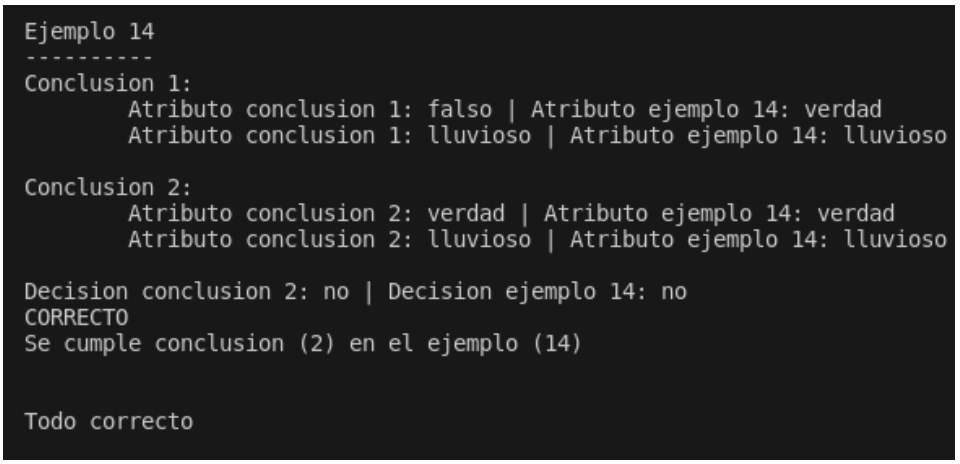

Imagen 12: salida por consola de la comprobación del último ejemplo del algoritmo ID3 y el resultado de la comprobación