PREDICCIÓN DEL ÉXITO EN LAS PRUEBAS SABER 11

UNIVERSIDAD EAFIT

Presentación del Equipo

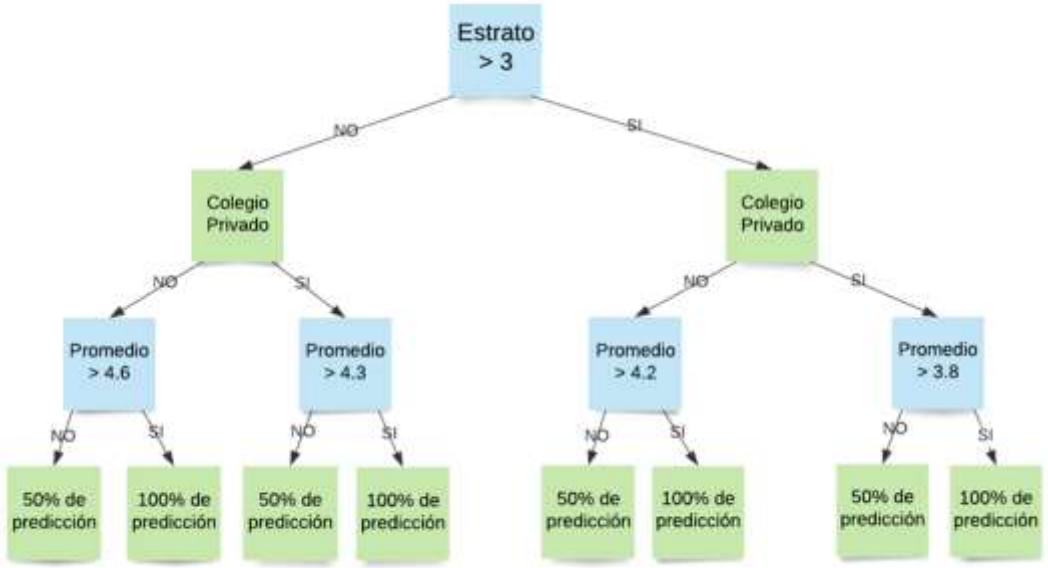
Katherin Nathalia
Allin Murillo

Alberto Andrés
Díaz Mejía

https://github.com/AlbertoD10-edu/ST0245-002/tree/master/proyecto

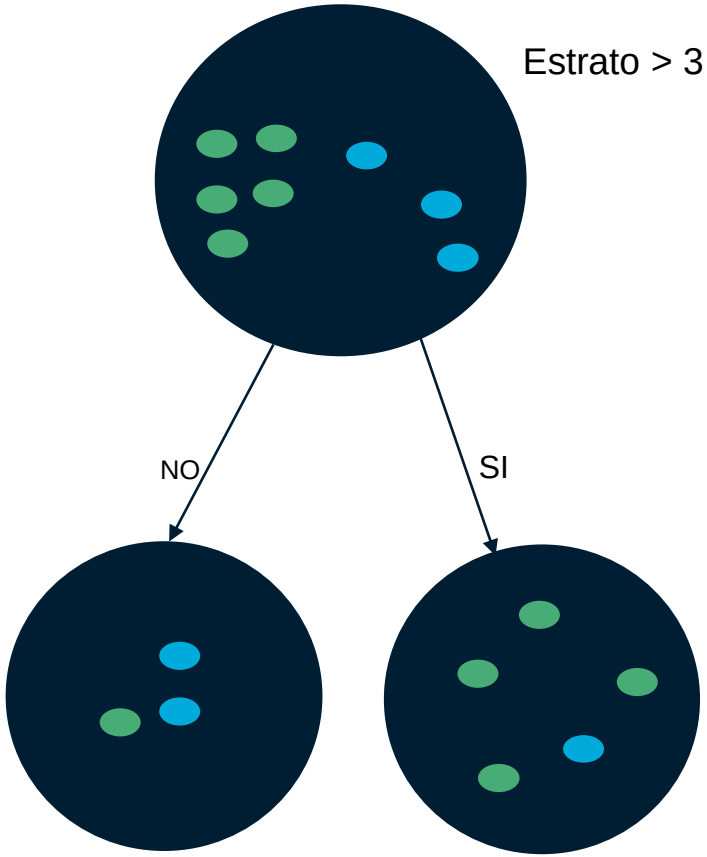
UNIVERSIDAD EAFIT

Diseño del Algoritmo

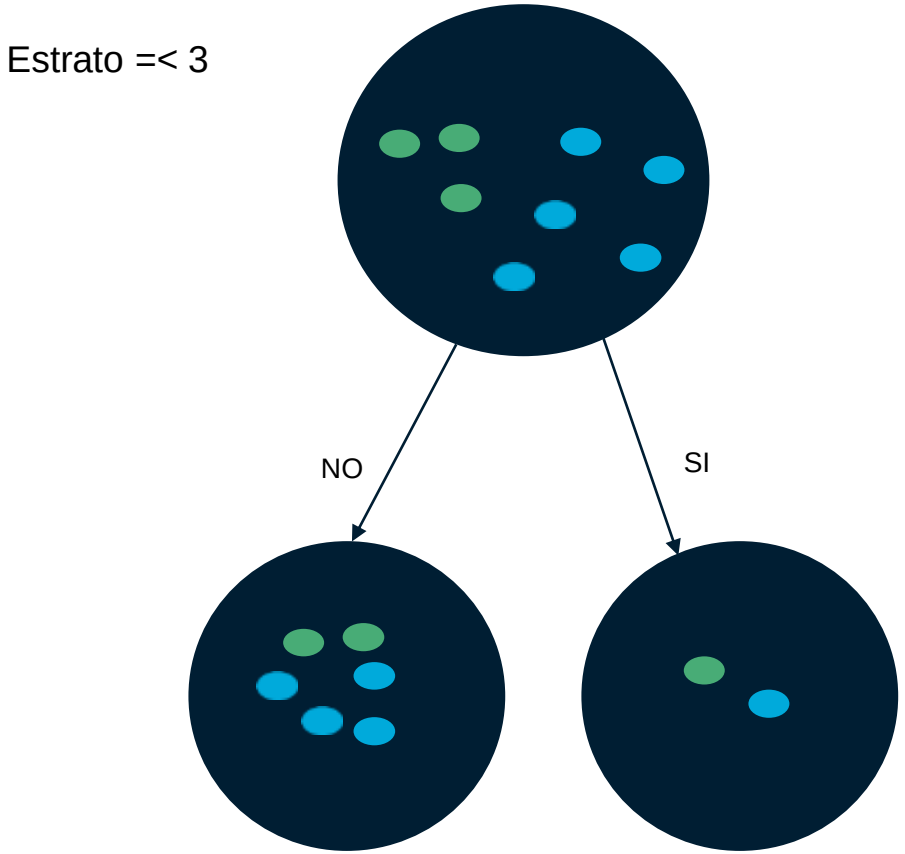


El tipo de algoritmo que decidimos implementar es el CART, nos basamos en la influencia que tiene el tipo de estrato y la educación a la que tienen acceso los alumnos, ya que estos son factores muy importantes para predecir el éxito de las pruebas; claro está que hay muchos factores que se pueden usar para una mejor predicción como promedio por materia, etc.

División de un nodo



Esta gráfica está basada en “Estrato > 3”. Para este caso la impureza Gini de la izquierda es 0,16, la impureza Gini de la derecha es 0,08 y la impureza ponderada es 0,12.

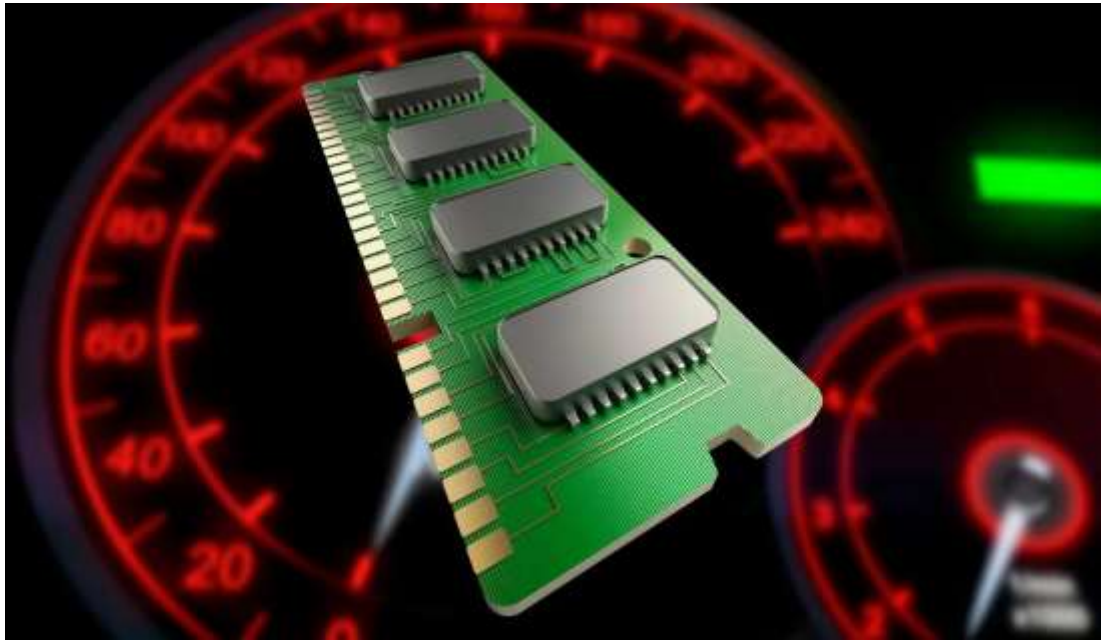


Esta gráfica está basada en “Estrato =< 3”. Para este caso la impureza Gini de la izquierda es 0,32, la impureza Gini de la derecho es 0,08 y la impureza ponderada es 0,20.

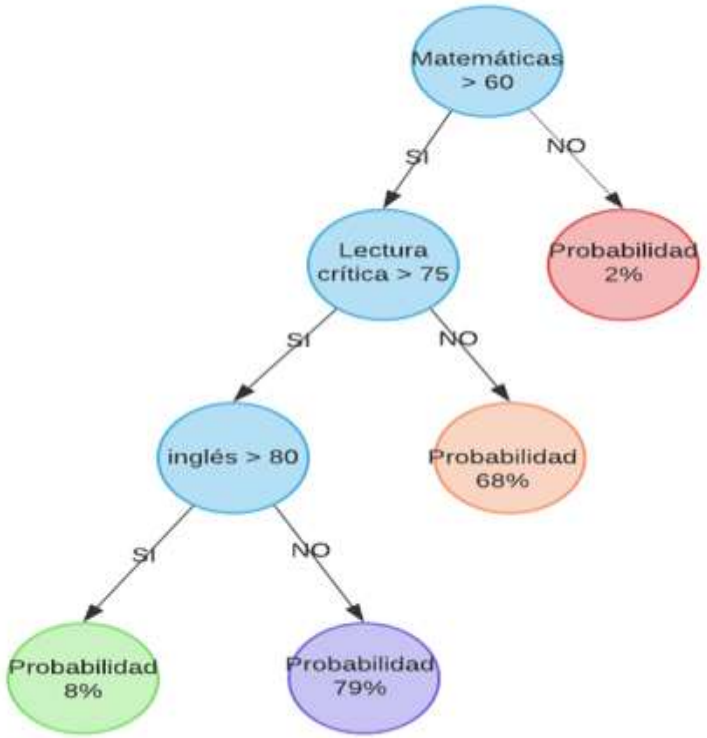
Complejidad del Algoritmo

----------	Complejidad en tiempo	Complejidad en memoria
Entrenamiento del modelo	O(log n*m*p)	O(m*n)
Algoritmo en práctica	O (n*p)	O(m*n)

Complejidad en tiempo y memoria del algorítmo basado en CART. En esta complejidad encontramos variables n, m y p, donde **n** es la representación de alumnos en el programa, **m** el número de variables a tener en cuenta y **p** el número de árboles elaborados durante el proceso



Modelo de Árbol de Decisión



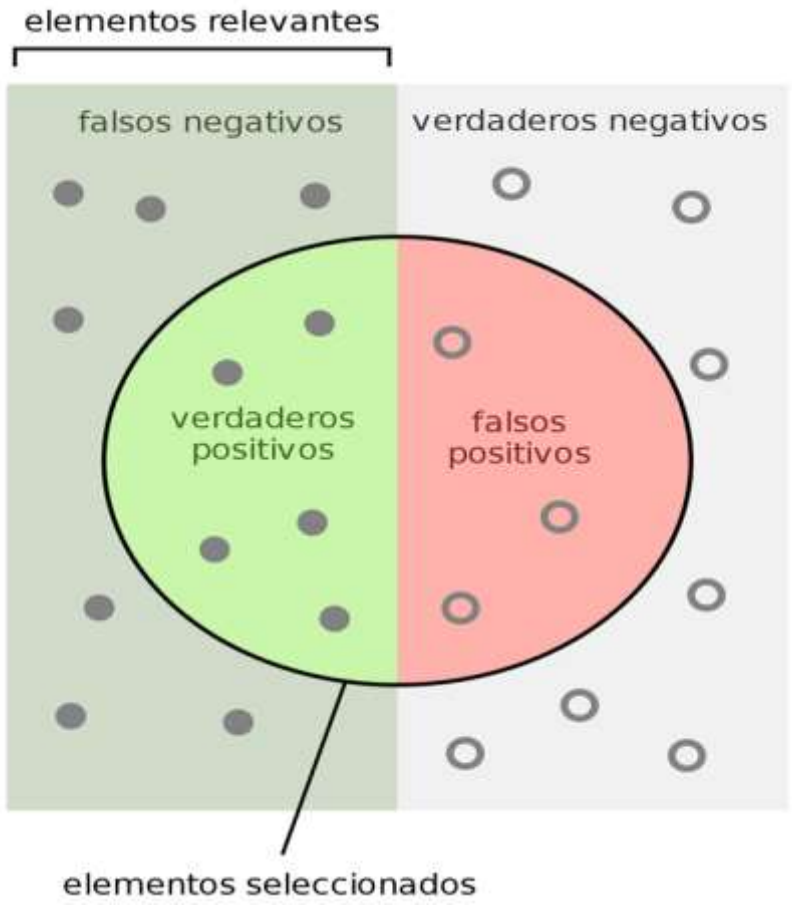
Características Más Relevantes

- Lectura Crítica
- Inglés
- Matemáticas

Nos guiamos con un estudio encontrado en el siguiente enlacehttp://rdigitales.uptc.edu.co/memorias/index.php/apli_estad/apli_esta2016/paper/viewFile/2169/2232, este estudio fue hecho hace un par de años y decidimos escoger estas 3 materias en específico porque son las que presentan mas bajos porcentajes en los niveles altos

UNIVERSIDAD EAFIT

Métricas de Evaluación



$$\text{Sensibilidad} = \frac{\textit{Número de alumnos identificados}}{\textit{Número de alumnos registrados}}$$

$$\text{Precisión} = \frac{\textit{Número de alumnos con éxito}}{\textit{Número de alumnos con predicción de éxito}}$$

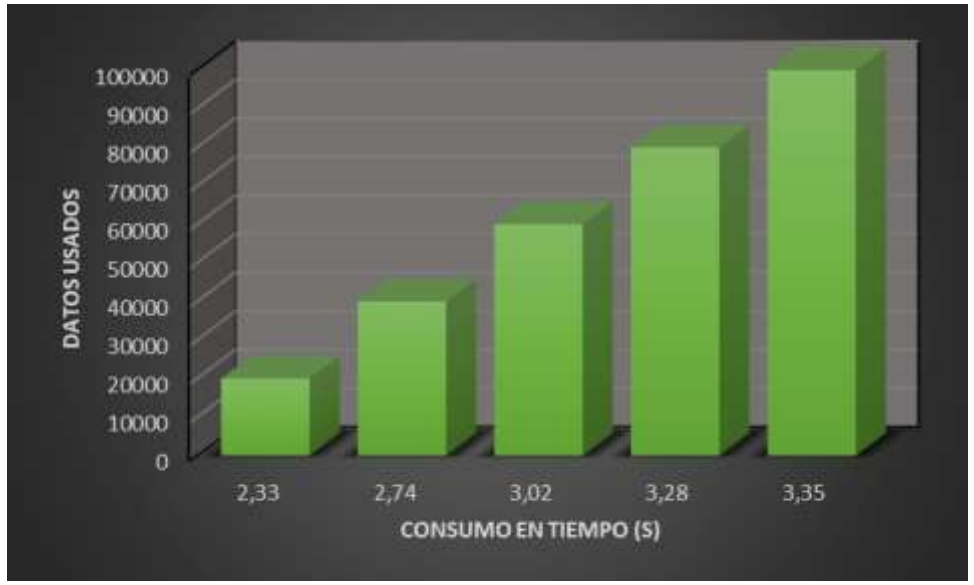
$$\text{Exactitud} = \frac{\textit{Número de entradas}}{\textit{Predicciones correctas}}$$

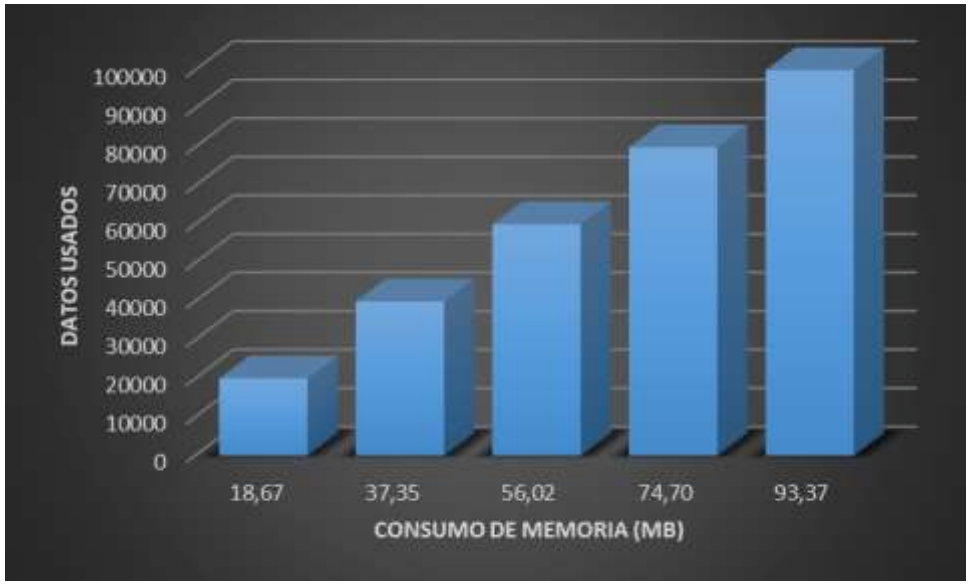
Métricas de Evaluación

----------	Conjunto de entrenamiento	Conjunto de validación
Exactitud	0,79	0,78
Precisión	0,8	0,77
Sensibilidad	0,56	0,5

Métricas de evaluación obtenidas con el conjunto de datos de entrenamiento de 100,000 estudiantes y el conjunto de datos de validación de 2,000 estudiantes.

Consumo de tiempo y memoria





Consumo de tiempo

Consumo de memoria

Código

```
public static Pair<String[][],String[][]> dividirDatos(String[][] array, int pos, int val){
    String[][] arrayOrdenado= OrdenarMatriz.mergesort(array,pos);
    int menor=0;
    double  valor= (double)val;
    int iterador=0;
    while(Double.parseDouble(arrayOrdenado[iterador][pos])<= valor && iterador<arrayOrdenado.length){
        menor++;
        iterador++;
    }
    int mayor=arrayOrdenado.length-menor;
    int filaMayor=0;
    int filaMenor=0;
    String[][] arrMayor = new String[mayor][arrayOrdenado[0].length];
    String[][] arrMenor = new String[menor][arrayOrdenado[0].length];
    for(int i=0; i< arrayOrdenado.length; i++){// O(n)
        if(Double.parseDouble(arrayOrdenado[i][pos])<=valor){
            arrMenor[filaMenor]=arrayOrdenado[i];
            filaMenor++;
        } else{
            arrMayor[filaMayor]=arrayOrdenado[i];
            filaMayor++;
        }
    }
    Pair<String[][], String[][]> p= new Pair(arrMenor,arrMayor);
    return p;
}
```

Código

```
public static String[] posiblesValores(String[][]arr, int pos){
    HashSet<String> valores= new HashSet<String>(arr.length);
    for(int i=0; i<arr.length; i++){
        if(arr[i][pos]!=null && arr[i][pos]!=""){
            valores.add(arr[i][pos]);
        }
    }
    String[] array= new String[valores.size()];
    return valores.toArray(array);
}
```

Código

```
public static float impurezaGini(float exito, float  noExito){
    float propExito=0;
    float propNoExito=0;
    float  total=exito+noExito;
    if(exito!=0){
        propExito= (exito/total);
    }
    if(noExito!=0){
        propNoExito=(noExito/total);
    }
    float impurezaObt= (1-((propExito*propExito)+(propNoExito*propNoExito)));
    return impurezaObt;
}
```

Código

```
public static Pair<Float, Integer> mejorCondicionAux(String[][] arr, int pos){
    String [] valoresP=OrdenarArreglo.mergesort(posiblesValores(arr, pos));
    float mejorValor=Float.parseFloat(valoresP[0]);
    float[][] newArr= new float[valoresP.length][3];
    float win=0;
    float fail=0;
    int Vi=0;
    int i=0;
    float iGM=0;
    for(int j=0; j<valoresP.length; j++){
        newArr[j][0]=Float.parseFloat(valoresP[j]);
        newArr[j][1]=0;
        newArr[j][2]=0;
    }
    while(Vi<valoresP.length && i<arr.length){
        if(Float.parseFloat(arr[i][pos])==Float.parseFloat(valoresP[Vi])){
            if(Integer.parseInt(arr[i][arr[0].length-1])==1){
                win++;
                newArr[Vi][1]=win;
            }
            i++;
            newArr[Vi][1]=win;
            newArr[Vi][2]=i-win;
        }else{
            Vi++;
        }
    }
    float  failT=newArr[newArr.length-1][2];
    float  winT=newArr[newArr.length-1][1];
    float impurezaT= impurezaGini(winT,failT);
    for(int o=0; o<valoresP.length;o++){
        float mayores;
        float menores;
        float menoresIG=impurezaGini(newArr[o][1],newArr[o][2]);
        float  winM=winT-newArr[o][1];
        float  failM= failT-newArr[o][2];
        float impurezaGiniMayores=impurezaGini(winM,failM);
        menores= newArr[o][1]+newArr[o][2];
        mayores= winM+failM;
        float impurezaPonderada = ((mayores*impurezaGiniMayores)+(menores*menoresIG))/(mayores+menores);
        float gananciaInfo= impurezaT-impurezaPonderada;
        if(gananciaInfo > iGM){
            iGM=gananciaInfo;
            mejorValor=Float.parseFloat(valoresP[o]);
        }
    }
    Pair<Float,Integer> p= new Pair(iGM, (int)mejorValor);
    return p;
}
```

Código

```
public static  Pair<Float, Pair<Integer, Integer>> mejorCondicion(String[][] arr){
    int columna=-1;
    float mayorG=0;
    float mayorGT=0;
    int aceptable=0;
    int aceptableT=0;
    Pair<Float, Integer> pareja1;
    for(int i=0; i<arr[0].length-1; i++){
        String [][] arreglo=OrdenarMatriz.mergesort(arr,i);
        pareja1= mejorCondicionAux(arreglo,i);
        mayorG = pareja1.getKey();
        if(mayorG>mayorGT){
            mayorGT=mayorG;
            aceptable=pareja1.getValue();
            columna=i;
        }
    }
    Pair<Float, Pair<Integer, Integer>> p2= new Pair(mayorGT, new Pair(aceptable, columna));
    return p2;
}
```


¡GRACIAS!