# **Capítulo 6. Árboles de Decisión**

* **Introducción**

Los Árboles de Decisión son algoritmos de Machine Learning versátiles que pueden efectuar tanto modelos de clasificación como modelos de regresión.Estos modelos son bastante poderosos y son capaces de entrenar sets de datos bastante complejos.

Si nos remotamos al proyecto end-to-end de las casas en London del capítulo 2, vimos que cuando utilizamos los Árboles de Decisión como modelo, los datos se ajustaron perfectamente tanto que se cayó en un sobreajuste.

Parte importante de saber sobre este algoritmo radica en que al juntar muchos Árboles de Decisión se obtiene uno de los modelos más poderosos de Machine Learning, el bosque aleatorio o Random Forest. Sin embargo, eso lo veremos en el siguiente capítulo, por ahora cubriremos como entrenar, visualizar y hacer predicciones con Árboles de Decisión.

También hablaremos de los diferentes algoritmos que Scikit-Learn tiene para implementar Árboles de Decisión, como regularizarlos y usarlos tanto para clasificación como regresión.Por último, hablaremos sobre algunas de las ventajas y desventajas de usar este modelo.

## **Entrenar y Visualizar un Árbol de Decisión**

Para hacer este vamos a utilizar un dataframe que contiene información sobre personas que desean ser comediantes. Estás personas ya fueron a dar un pequeño show y con ese performance le pusieron un ranking y tomaron datos como la edad, experiencia y la nacionalidad. A partir de estos resultados se seleccionaron a varios afortunados para que pasaran al siguiente nivel del concurso de comedia.

Entonces, lo que se busca es armar un programa que permita saber quién va a ser comediante y quién no.

Lo primero que hay que hacer es importar todas las bibliotecas:

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import os

Después necesitamos bajar los datos desde github y declarar una función que extraiga esos datos.

DOWNLOAD\_ROOT = "https://raw.githubusercontent.com/a2Proyectos/MachineLearning\_Data/main/"

COMEDIANTES = "Capitulo\_6/Comediantes.csv"

def extraer\_datos(root,database):

csv\_path=root+ database

return pd.read\_csv(csv\_path)

Extraemos los datos y visualizamos el dataframe:

df=extraer\_datos(DOWNLOAD\_ROOT,COMEDIANTES)

df



Nota: GO es la variable que dictamina si será o no será comediante

Dentro del dataframe se encuentran dos variables, GO y Nacionalida, que están dadas de manera alfabética, como Scikit-Learn no puede tomar datos de este tipo de variables vamos a utilizar la función *LabelEncoder* para asignar un valor numérico a las variables con string.

from sklearn.preprocessing import LabelEncoder

def label\_encoder(datos\_categoria):

le = LabelEncoder()

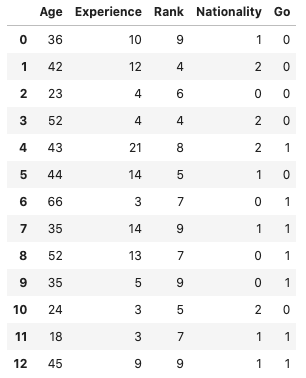
df[datos\_categoria] = le.fit\_transform(df[datos\_categoria])

variables = ["Nationality","Go"]

for l in variables:

label\_encoder(l)

df



Ahora que ya tenemos nuestros datos, lo que sigue es separar el dataframe en dos: los datos que queremos predecir (GO) y los que que nos ayudarán a predecir (todas las demás variables).

y = df["Go"]

x = df.drop("Go",axis=1)

Ya que tenemos esto listo, procederemos a hacer nuestro clasificador de DecisionTree:

from sklearn.tree import DecisionTreeClassifier

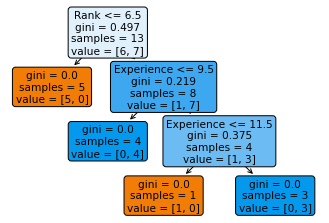
arbol = DecisionTreeClassifier()

arbol.fit(x,y)

Para visualizar los resultados podemos gráficar los datos de la siguiente manera:

from sklearn import tree

tree.plot\_tree(arbol,feature\_names=x.columns,rounded=True,filled=True)

****

* **Hacer Predicciones**

Del gráfico obtenido anteriormente necesitamos entender lo siguiente:

Todos los rectángulos que se muestran se llaman nodos. Estos pueden ser de raíz (hay más nodos debajo de él) y de hoja (que no hay otro nodo debajo de él).

Ahora, imaginemos que queremos saber si una persona es o no comediante. Este nodo pregunta si el ranking es menor o igual a 6.5. Si es así, entonces se mueve a la derecha en donde está otro nodo raíz. Si no es así, se mueve al nodo de hoja en donde se termina la evaluación y el modelo establece que no será comediante.

Ahora sigamos con el mismo ejemplo pero veamos los parámetros de *Value.* Nosotros podemos ver que se muestra [6, 7]. Estos valores establecen que hay 6 personas con el valor 0 (no será comediante) y 7 personas con el valor 1 (si será comediante). La suma de estos dos da el valor de *samples* que en este caso es 13.

Por último, nos queda el parámetro *gini.* Este es una medida de impureza. Un nodo es puro cuando tiene un gini de 0 que se atribuye a que todas las instancias dentro de ese nodo son de la misma clase. Gini se calcula de la siguiente manera:

* **Estimar las Probabilidad de Clase**

Los Árboles de Decisión también estiman la probabilidad de que una instancia (dato) pertenezca a una clase en particular. Para ver como funciona vamos a inventarnos una persona que tenga 40 años, 10 años de experiencia laboral, 7 de ranking y que su nacionalidad sea británica.

arbol.predict\_proba([[40,10,7,1]])



Nosotros, con base a lo que ya sabemos, sospechamos que el modelo no lo iba a aceptar y dicho y hecho, nos da una probabilidad del 100% (1) de que no sea comediante.

Si cambiamos a una persona que tenga las mismas características pero solo tenga 6 años de experiencia entonces nos daría 1 de probabilidad, es decir, un 100% de probabilidad de que si sea comediante.

arbol.predict\_proba([[40,6,7,1]])



* **CART- Árbol de Clasificación y Regresión**

El algoritmo CART por sus siglas inglés Classification an Regression Tree, se utiliza por Scikit-Learn para entrenar a los Árboles de Decisión. El algoritmo funciona de la siguiente manera: primero, divide el set de entrenamiento en dos utilizando un solo criterio, por ejemplo, en el caso del árbol de arriba, el primer cirterío era el rango menor o igual a 6.5.

Pero ¿cómo va a decidir cuál es el mejor criterío para hacer la separación? Va agarrar el criterio que de el mejor GINI combinado en sus nodos siguientes. En pocas palabras, seleccionará el criterio que ayude al sistema a mantenerse lo más puro posible.

La función de costo del algoritmo CART es:

Dónde:

A CART se le conoce como un algoritmo avaricioso porque se enfoca en un salto al frente. Si esta calculado el primer nodo, no toma en cuenta si el hacer la mínima impureza posible le va a generar en los futuros niveles de nodos.

* **Complejidad Computacional**

La complejidad computacional del algoritmo GINI es baja y eficiente, es por eso que, esto se puede utilizar para el siguiente capítulo de Bosques Aleatorios. La notación *Big O* de la complejidad computacional de este algoritmo es siendo *m* la cantidad de instancia que se tiene en los datos.

El algoritmo de entrenamiento compara todas las variable de toda la muestra en cada nodo. Entonces la complejidad computacional al momento de estar entrenando el modelo resulta en .

Al final de cuentas, aunque esto crecerá a medida que aumenten los datos, la complejidad sigue siendo barata en comparación con otros modelos.

* **Impureza GINI y Entropía**

En estos algoritmos se utiliza por default la medida de impureza Gini, sin embargo, también se puede seleccionar la opción de entropía al seleccionar *entropy* en el hiperpárametro *criterion.*

La entropía es cero cuando contiene instancias de una sola clase. Pero ¿por qué escoger entropía en vez de la impureza Gini? La realidad es que en la mayoría de los casos no hay diferencias significativas. La impureza Gini es un poco más rápida de calcular y es un buen estándar. Sin embargo, cuando difieren, la impureza Gini tiende a aislar la clase más frecuente en su propio nodo del árbol, muentras que la entropía tiende a producir árboles más balanceados.

* **Regularización de Hiperparámetros**

Los Árboles de Decisión asumen muy poco sobre el modelo que estamos haciendo al contrario de otro modelos. Por ejemplo, los modelos lineales asumen que los datos son lineales. El árbol de decisión no va a suponer nada, si se deja sin restricciones el algoritmo se adaptará a los datos de entrenamiento tanto que tenderá a ocasionar un sobreajuste.

Para evitar el sobreajuste, se puede utilizar restricciones por medio de diferentes hiperparamentros. A este proceso se le conoce como regularización. Para hacer esto se pueden utilizar cuatro diferentes hiperparámetros de regularización

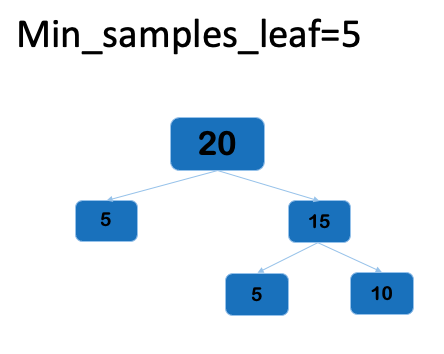
* + El primero es *min\_sample\_split* en donde le pone una restricción al modelo de cuántas instancias debe de tener en ese nodo para permitir que se ramifique en dos nodos nuevos.

Ejemplo:



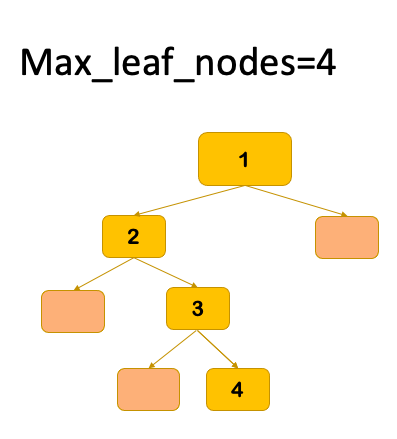
* + El segundo es *Min\_samples\_leaf* establece cuántas muestras puede tener cada una de las hojas.

Ejemplo:



* + El tecero es *Max\_leaf\_nodes* establece el máximo número de ramificaciones que un nodo puede tener, entonces no podrías tener más de los nodos raíces que establece en este hiperparámetro.

Ejemplo:

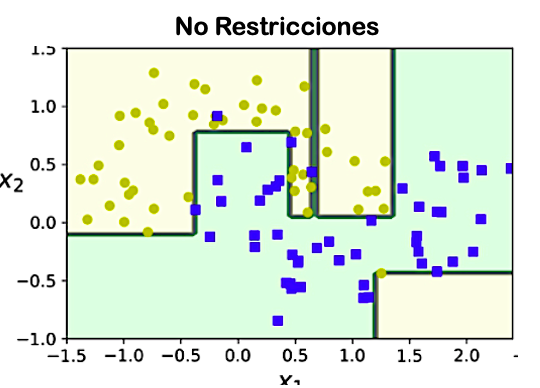


* + El cuarto es *Max\_features* limita el número de variables predictoras con las que puede trabajar el modelo.

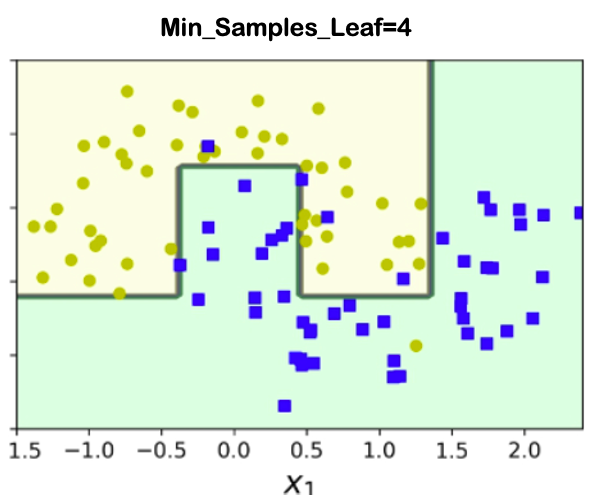
*Ejemplo:*

**

Utilizando estos hiperparámetros, podemos generar un modelo de árbol de decisión que no dará resultados perfectos pero evitará el sobreajuste. Vamos viendo un ejemplo, aquí tenemos un gráfica en dónde tenemos nuestro dataset de *make\_moons* en donde no se aplico restricción alguna:



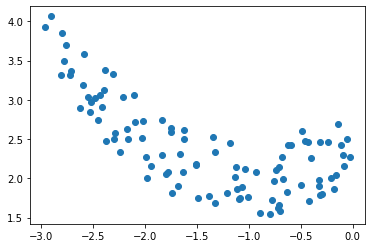
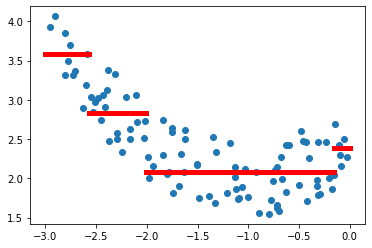
Aquí se puede ver que el modelo realizó un sobreajuste al intentar cubrir todos los datos en azul creando áreas en donde no era necesario crearlas. Veamos ahora el mismo dataset pero con una regularización con el hiperpárametro *Min\_Samples\_Leaf=4*



Aunque se omitieron algunas instancias en su correcta clasificación, podemos decir que este modelo podrá predecir de manera satisfactoria cuando otro dataset se le presente.

* **Regresión**

Los Árboles de Decisión son capaces de hacer regresiones. Por ejemplo, si nosotros tenemos este modelo y le ajustamos un Árbol de Decisión se vería así:

Básicamente, esta poniendo segmentos que establecerán un rango de clasificación. Vamos viendo este ejemplo más a fondo.

Para empezar, creemos un set de datos de juguetes:

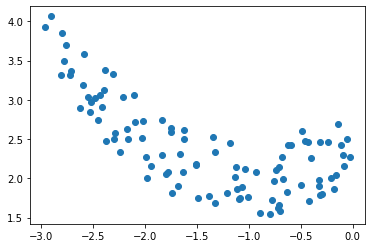
m = 100

x = 3 \* np.random.rand(m,1) - 3

y = 2 + x + 0.5\*x\*\*2 + np.random.rand(m,1)

#np.c\_[([x])]

plt.scatter(x,y)



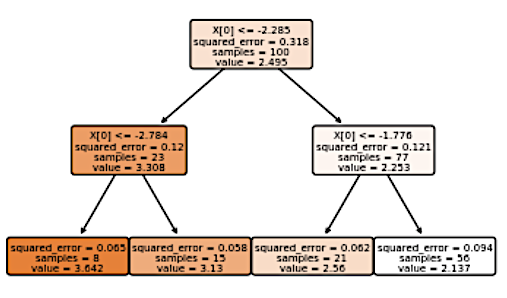
Ahora creemos el árbol de decisión con una profundidad de 2:

from sklearn.tree import DecisionTreeRegressor

arbol = DecisionTreeRegressor(max\_depth=2)

arbol.fit(x,y)

Veamos los resultados:



El árbol es similar al árbol construido para la clasificación. La diferencia principal es que en vez de predecir una clase en cada nodo, predice un valor. El squarred\_error es análogo a la impureza Gini, a medida que los nodos van avanzando esta medida disminuye.

El modelo ajustado sobre el set de datos original se vería de la siguiente manera:

* + Ojo: Los cuatro nodos que tenemos arriba se ven representados por la líneas rojas

x1 = np.linspace(-3,-2.568)

y1 = np.linspace(3.5818,3.5818)

x2 = np.linspace(-2.569,-2.008)

y2 = np.linspace(2.834,2.834)

x3 = np.linspace(-2.009,-0.151)

y3 = np.linspace(2.075,2.075)

x4 = np.linspace(-0.152,0)

y4 = np.linspace(2.392,2.392)

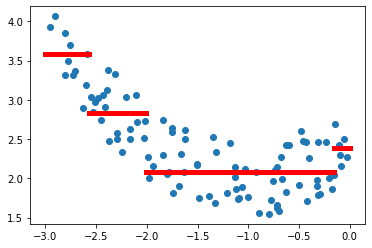
plt.scatter(x,y)

plt.plot(x1,y1,"r-",linewidth=5)

plt.plot(x2,y2,"r-",linewidth=5)

plt.plot(x3,y3,"r-",linewidth=5)

plt.plot(x4,y4,"r-",linewidth=5)



Aunque es un buen modelo, se podría hacer un mejor. Uno que no indique que esta sobreajustado. Para eso, volvamos a nuestros modelo y establezcamos un profundida máxima de 3. El cambio resultaría en el siguiente modelo:

from sklearn.tree import DecisionTreeRegressor

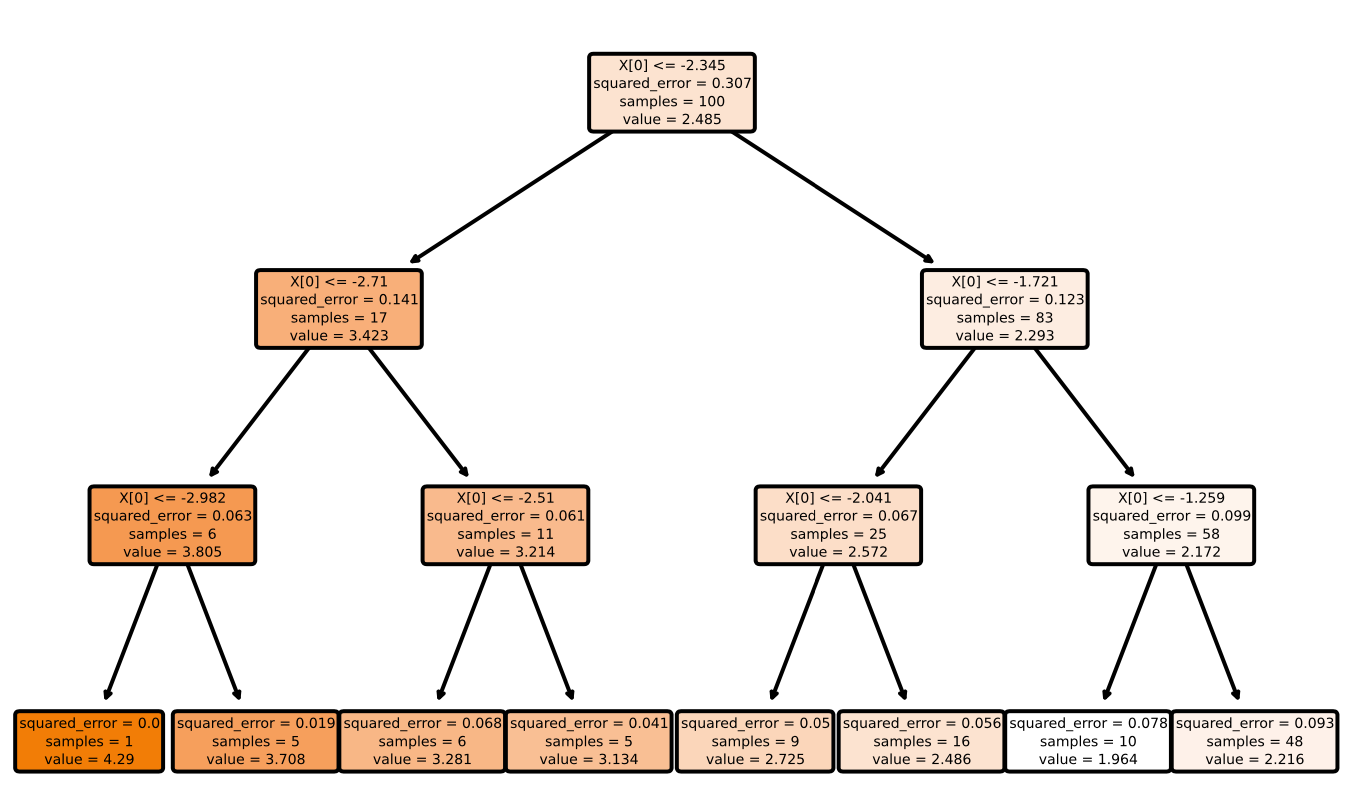
arbol = DecisionTreeRegressor(max\_depth=3)

arbol.fit(x,y)

#Gráficalo

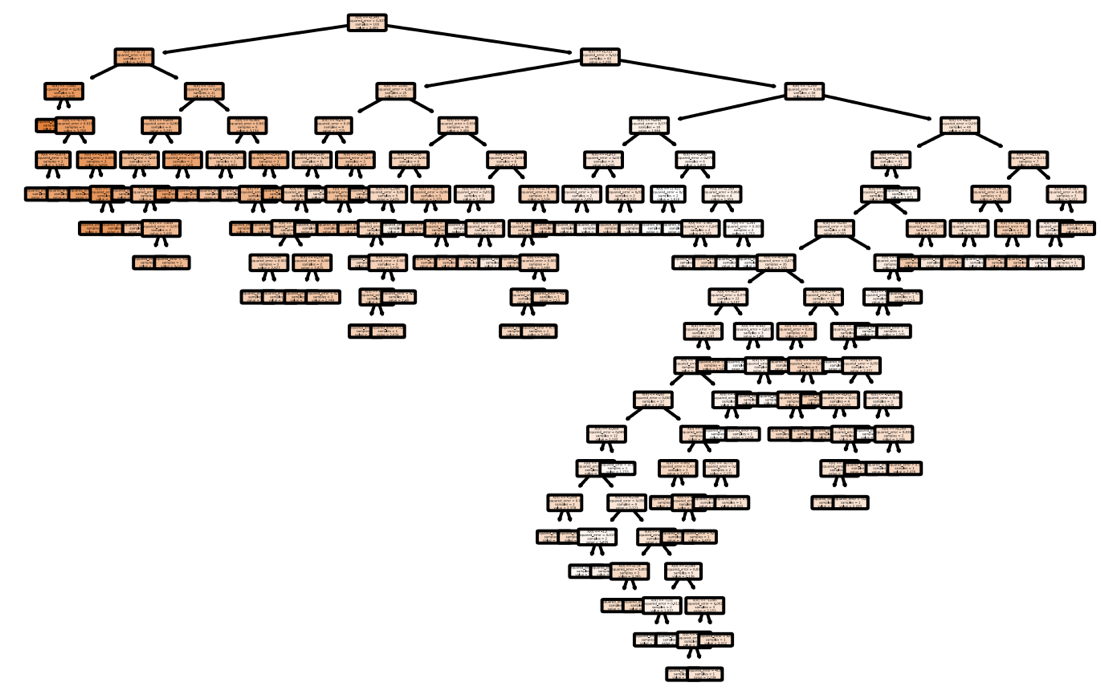
tree.plot\_tree(arbol,rounded=True,filled=True)

#tree.plot\_tree(arbol,rounded=True,filled=True,fontsize=15)Para hacer más grande la letra ajustar fontsize



Por último, si yo hiciera el modelo sin restriccione y con restricción de 10 niveles, respectivamente se vería de la siguiente manera:

Aunque se quisera conservar el árbol sin restricciones debido a su regresión perfecta, en realida no sería viable debido al sobreajuste que presentaría.



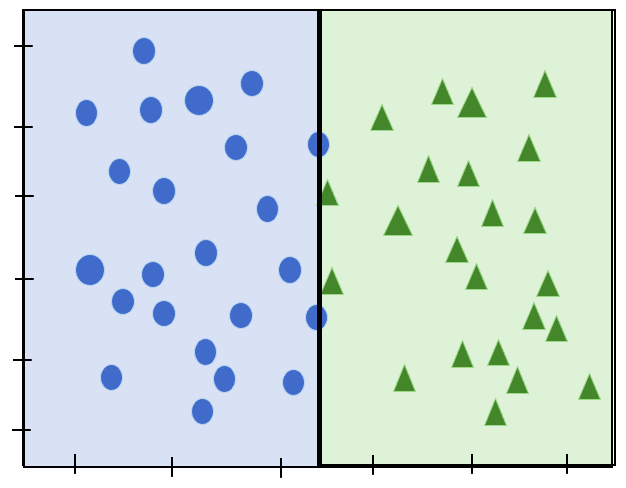
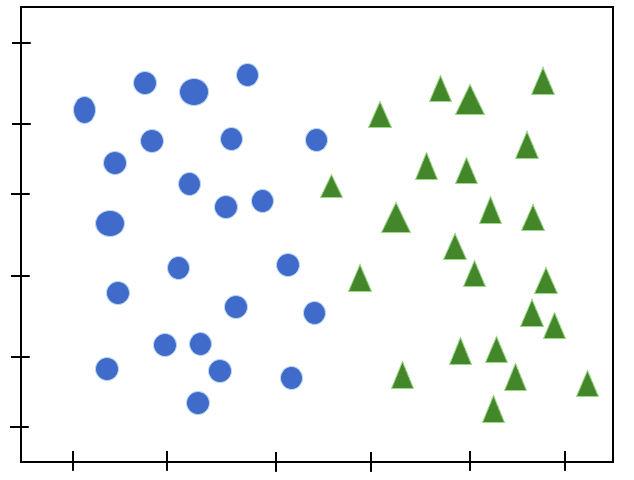
Por lo tanto, se podría tomar el modelo con nivel máximo de 10 pues consiste en un modelo más preciso pero quizás sin llegar a los extremos de sobreajuste.



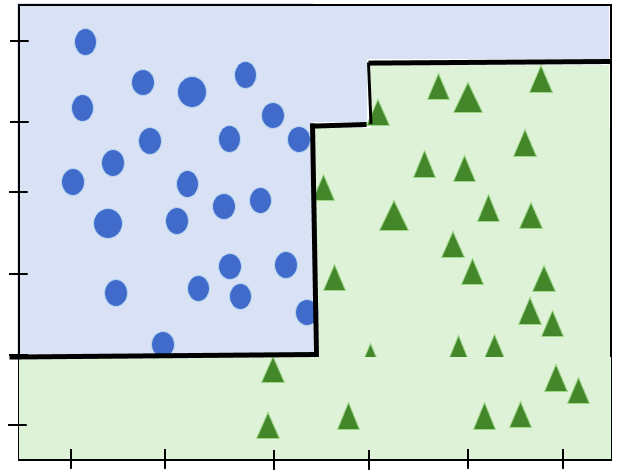
* **Inestabilidad**

Ahora que ya entiendes los Árboles de Decisión, vamos a ver una de las limitaciones de estos algoritmos que en realidad no es sorpresa a todo el contenido que hemos estado revisando. El problema son los sobreajustes.

Veamos un ejemplo. Aquí tenemos un set de datos que es sencillo de clasificar. Facilmente se puede observar que la clasificación viable es con una línea en medio y ya.



El problema se presenta El Árbol de Decisión se pone creativo y hace esto:



Volvemos a lo mismo, los Árboles de Decisión suelen sobreajustar y cualquier rotación de los datos, cualquier dato sobresaliente lo metera en problemas.