Árvores de decisão

Introdução

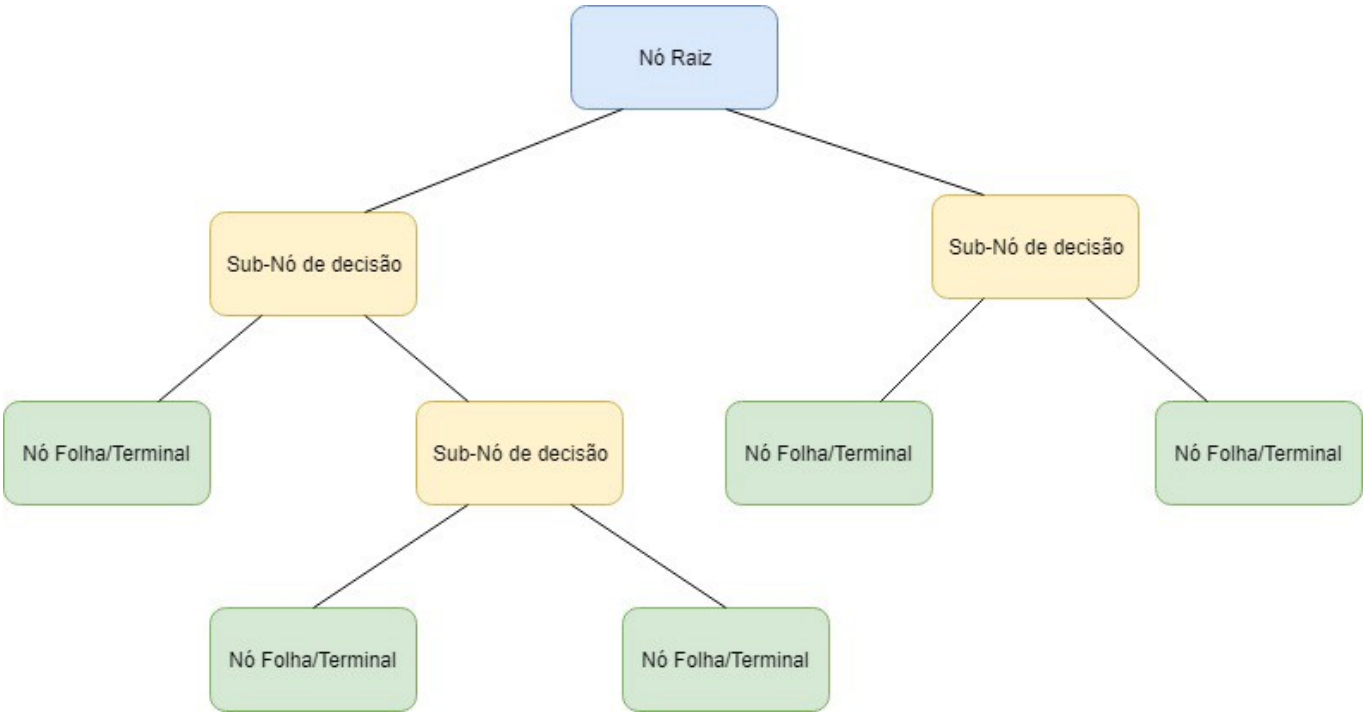
Você já ouviu falar do Akinator? O Akinator é um site que tenta "adivinhar" uma pessoa que você está pensando. Para isso, ele faz diversas perguntas para você. Quando ele estiver "satisfeito", ele faz um chute e você informa se ele acertou ou não.

Esse site está no ar desde 2007 e continua tendo vários acessos todos os dias.

Não se sabe exatamente como esse site foi desenvolvido, mas muito provavelmente o que o desenvolvedor (ou desenvolvedores) utilizou foi um modelo de árvore de decisão.

Uma árvore de decisão é aquela na qual você possui uma lista de atributos para cada alvo. Esses atributos podem ser categóricos (o personagem é real?) ou numéricos (a pessoa tem mais de 24 anos?).

Resumindo, um algoritmo de árvore de decisão trabalha no aprendizado de um conjunto de perguntas do tipo "sim/não" que podem levar a uma decisão.



Critérios de impureza

O primeiro passo para iniciar uma árvore de decisão é escolher o nó raiz, aquele no qual toda a árvore vai se desenvolver, e para isso vamos aplicar dois conceitos normalmente utilizados nesse contexto, chamados critérios de impureza.

O primeiro critério é o índice de impureza de Gini, em homenagem ao estatístico italiano Corrado Gini, e é calculado como:

Ig = 1 - SUM(i=1, k)(p(i))^2

O segundo critério é a entropia, calculado como:

Aplicação

Para aplicar esses conceitos, vamos utilizar uma base de dados simples sobre tênis. A base inclui diversos dados sobre o clima, temperatura, umidade e incidência de vento e indica se aquelas condições são adequadas ou não para se jogar tênis. Esses dados estão disponíveis no Kaggle.

Import e análise dos dados

```
import pandas as pd
```

```
df = pd.read_csv("play_tennis.csv")
```

```
df
```

	day	outlook	temp	humidity	wind	play
0	D1	Sunny	Hot	High	Weak	No
1	D2	Sunny	Hot	High	Strong	No
2	D3	Overcast	Hot	High	Weak	Yes
3	D4	Rain	Mild	High	Weak	Yes
4	D5	Rain	Cool	Normal	Weak	Yes
5	D6	Rain	Cool	Normal	Strong	No
6	D7	Overcast	Cool	Normal	Strong	Yes
7	D8	Sunny	Mild	High	Weak	No
8	D9	Sunny	Cool	Normal	Weak	Yes
9	D10	Rain	Mild	Normal	Weak	Yes
10	D11	Sunny	Mild	Normal	Strong	Yes
11	D12	Overcast	Mild	High	Strong	Yes
12	D13	Overcast	Hot	Normal	Weak	Yes
13	D14	Rain	Mild	High	Strong	No

Pela definição, se tivermos uma categoria com apenas uma classe do nosso alvo, ela terá um índice de impureza de Gini de 0. A ideia da construção de uma árvore de decisão é escolher uma categoria (ou atributo) com o menor grau de impureza de Gini.

Vamos ver então um caso em que colocamos como raiz da nossa árvore de decisão o atributo "Outlook". Podemos ver que do total de dados, quando o tempo foi ensolarado (classe 1), em 2 dias foi jogado tênis e em 3 não foi jogado. Quando o tempo foi nublado (classe 2), tivemos jogo em 4 dias, de um total de 4. E quando o tempo estava chuvoso (classe 3), tivemos jogo em 3 dos 5 dias. Portanto, vamos calcular o índice de impureza, e em seguida calculamos a média ponderada do atributo "Outlook":

```
g1 = 1 - (2/5)^2 - (3/5)^2 = 0.48

g2 = 1 - (4/4)^2 - (0/4)^2 = 0

g3 = 1 - (3/5)^2 - (2/5)^2 = 0.48

G.I.(Outlook|Play Tennis) = (5/14) * g1 + (4/14) * g2 + (5/14) * g3 = 0.342
```

Vamos fazer o mesmo com o atributo "Temp":

```
Hot - 2 Sim / 2 Não -> g1 = 1 - (2/4)^2 - (2/4)^2 = 0.5

Mild - 4 Sim / 2 Não -> g2 = 1 - (4/6)^2 - (2/6)^2 = 0.444

Cold - 3 Sim / 1 Não -> g3 = 1 - (3/4)^2 - (1/4)^2 = 0.375

G.I.(Temp|Play Tennis) = (4/14) * g1 + (6/14) * g2 + (4/14) * g3 = 0.440
```

Fazendo o mesmo processo para Umidade e para Vento, temos respectivamente 0.367 e 0.429. Portanto, o atributo mais adequado para adotarmos como raiz é a condição do tempo.

Vamos ordenar nossa tabela para ver qual seria o próximo passo.

```
df.sort_values("outlook", inplace=True)
df
```

	day	outlook	temp	humidity	wind	play
2	D3	Overcast	Hot	High	Weak	Yes
6	D7	Overcast	Cool	Normal	Strong	Yes
11	D12	Overcast	Mild	High	Strong	Yes
12	D13	Overcast	Hot	Normal	Weak	Yes
3	D4	Rain	Mild	High	Weak	Yes
4	D5	Rain	Cool	Normal	Weak	Yes
5	D6	Rain	Cool	Normal	Strong	No
9	D10	Rain	Mild	Normal	Weak	Yes

Vamos agora entender qual seria a próxima pergunta caso a resposta da primeira fosse "Sunny". Para a nossa amostra de 6 dados, recalculamos o índice de impureza de Gini, e adotamos o atributo com menor valor.

- Atributo "temp":

```
Hot - 0 Sim / 2 Não -> g1 = 1 - (0/2)^2 - (2/2)^2 = 0

Mild - 1 Sim / 1 Não -> g2 = 1 - (1/2)^2 - (1/2)^2 = 0.5

Cold - 1 Sim / 0 Não -> g3 = 1 - (1/1)^2 - (0/1)^2 = 0

G.I.(Temp|Play Tennis) = (2/5) * g1 + (2/5) * g2 + (1/5) * g3 = 0.2
```

- Atributo "humidity":

```
High - 0 Sim / 3 Não -> g1 = 1 - (0/3)^2 - (3/3)^2 = 0

Normal - 2 Sim / 0 Não -> g2 = 1 - (2/2)^2 - (0/2)^2 = 0

G.I.(Humidity|Play Tennis) = (3/5) * g1 + (2/5) * g2 = 0
```

Analogamente para "wind", temos o índice igual a 0.467. Portanto escolhemos o atributo "Humidity" para ser a nossa próxima pergunta. Como o índice de impureza nessa categoria é zero, não precisamos aprofundar a árvore nesse ponto.

No caso do tempo chuvoso, temos os seguintes índices:

```
G.I.(Temp|Play Tennis) = 0.467

G.I.(Humidity|Play Tennis) = 0.467

G.I.(Wind|Play Tennis) = 0.25
```

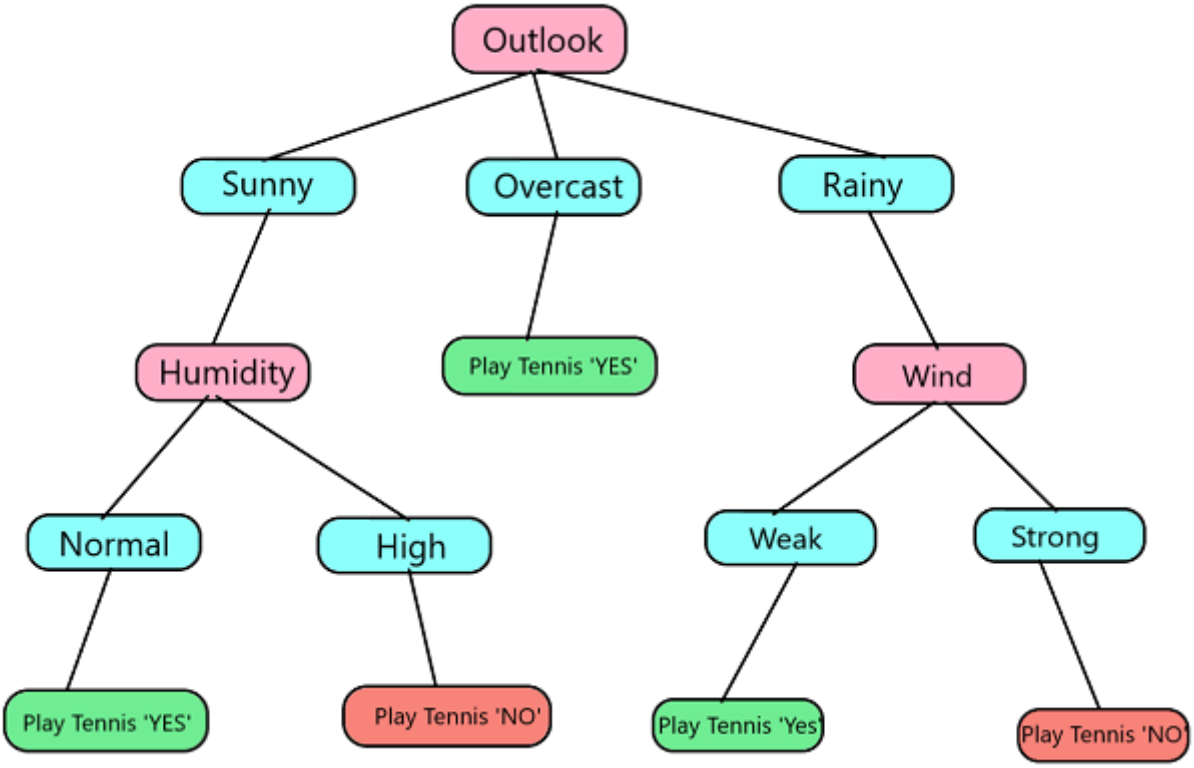
Escolhemos então o atributo "Wind".

```
df.sort_values(by=["outlook", "wind"], inplace=True)
df
```

	day	outlook	temp	humidity	wind	play
6	D7	Overcast	Cool	Normal	Strong	Yes
11	D12	Overcast	Mild	High	Strong	Yes
2	D3	Overcast	Hot	High	Weak	Yes
12	D13	Overcast	Hot	Normal	Weak	Yes
5	D6	Rain	Cool	Normal	Strong	No
13	D14	Rain	Mild	High	Strong	No
3	D4	Rain	Mild	High	Weak	Yes
4	D5	Rain	Cool	Normal	Weak	Yes
9	D10	Rain	Mild	Normal	Weak	Yes
1	D2	Sunny	Hot	High	Strong	No
10	D11	Sunny	Mild	Normal	Strong	Yes
0	D1	Sunny	Hot	High	Weak	No
7	D8	Sunny	Mild	High	Weak	No
8	D9	Sunny	Cool	Normal	Weak	Yes

Podemos ver que nas duas classes da categoria "wind" temos um índice de impureza igual a 0, portanto não precisamos aprofundar a árvore.

Das classes de tempo, restou apenas o "nublado". Porém, nesse caso, todas as entradas de dados estão classificadas como "Sim", logo não precisamos continuar com a árvore.



Análise pela scikit-learn

Vamos agora ver como fazer as previsões utilizando sklearn. Como nossos dados não são categóricos, não conseguimos alimentar diretamente o nosso classificador. Portanto, precisamos mapear as nossas classes, como fizemos na última nota de aula.

```
df.outlook = df.outlook.map({"Overcast": 0, "Sunny": 1, "Rain": 2})
df.temp = df.temp.map({"Hot": 0, "Mild": 1, "Cool": 2})
df.humidity = df.humidity.map({"High": 0, "Normal": 1})
df.wind = df.wind.map({"Strong": 0, "Weak": 1})
df.play = df.play.map({"No": 0, "Yes": 1})

df = df.loc[:, ["outlook", "temp", "humidity", "wind", "play"]]
df
```

	outlook	temp	humidity	wind	play
6	0	2	1	0	1
11	0	1	0	0	1
2	0	0	0	1	1
12	0	0	1	1	1
5	2	2	1	0	0

Vamos agora dividir o nosso conjunto nos dados de treino e nos alvos do treino. Como nosso conjunto de dados é muito pequeno, não vamos dividi-lo em dados de treino e dados de teste.

4	2	2	1	1	1

```
train_data = df.drop("play", axis=1)
train_target = df["play"]
```

1	1	0	0	0	0

```
train_data
```

	outlook	temp	humidity	wind
6	0	2	1	0
11	0	1	0	0
2	0	0	0	1
12	0	0	1	1
5	2	2	1	0
13	2	1	0	0
3	2	1	0	1
4	2	2	1	1
9	2	1	1	1
1	1	0	0	0
10	1	1	1	0
0	1	0	0	1
7	1	1	0	1
8	1	2	1	1

```
train_target
```

```
6     1
11    1
2     1
12    1
5     0
13    0
3     1
```

```
4     1
9     1
1     0
10    1
0     0
7     0
8     1
Name: play, dtype: int64
```

Agora importamos e ajustamos o nosso modelo utilizando a sklearn.

```
from sklearn.tree import DecisionTreeClassifier
model = DecisionTreeClassifier(criterion="gini")
model.fit(train_data, train_target)
```

```
DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='gini',
                       max_depth=None, max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort='deprecated',
                       random_state=None, splitter='best')
```

Agora vamos fazer as previsões utilizando a árvore que construímos:

```
pred = model.predict(train_data)
pred
```

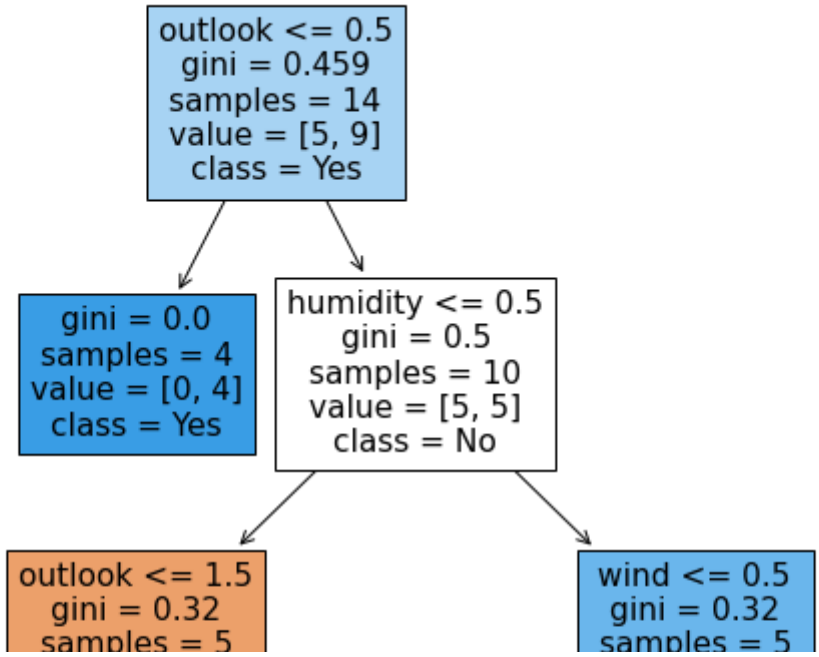
```
array([1, 1, 1, 1, 0, 0, 1, 1, 1, 0, 1, 0, 0, 1])
```

Vamos fazer a visualização dos dados utilizando um método próprio da sklearn e a matplotlib.

```
import matplotlib.pyplot as plt
from sklearn import tree

fig = plt.figure(figsize=(10,10))
tree.plot_tree(
    model,
    feature_names=train_data.columns,
    class_names=["No", "Yes"],
    filled=True
)
```

```
[Text(209.25, 475.65000000000003, 'outlook <= 0.5\ngini = 0.459\nsamples = 14\nvalue
 Text(139.5, 339.75, 'gini = 0.0\nsamples = 4\nvalue = [0, 4]\nclass = Yes'),
 Text(279.0, 339.75, 'humidity <= 0.5\ngini = 0.5\nsamples = 10\nvalue = [5, 5]\ncla
 Text(139.5, 203.85000000000002, 'outlook <= 1.5\ngini = 0.32\nsamples = 5\nvalue =
 Text(69.75, 67.94999999999999, 'gini = 0.0\nsamples = 3\nvalue = [3, 0]\nclass = No
 Text(209.25, 67.94999999999999, 'gini = 0.5\nsamples = 2\nvalue = [1, 1]\nclass = N
 Text(418.5, 203.85000000000002, 'wind <= 0.5\ngini = 0.32\nsamples = 5\nvalue = [1,
 Text(348.75, 67.94999999999999, 'gini = 0.5\nsamples = 2\nvalue = [1, 1]\nclass = N
 Text(488.25, 67.94999999999999, 'gini = 0.0\nsamples = 3\nvalue = [0, 3]\nclass = Y
```



Aqui temos no *nó raiz* que dividimos um total de 14 amostras em 5 com a primeira classe ("Não") e 9 com a classe "Sim". Podemos ver que todas as folhas possuem amostras com apenas uma classe.

O nó filho à esquerda da raiz representa uma resposta positiva à pergunta, enquanto que o filho à direita representa uma resposta negativa.

Se tivermos, por exemplo, uma nova amostra, com os parâmetros:

```
Outlook = 1, Temperature = 2, Humidity = 0, Wind = 1
```

Chegaríamos no fluxo que teríamos uma resposta "No".

Implementando essa análise, temos:

```
import numpy as np

new_data = pd.DataFrame(np.array([[1, 2, 0, 1]]), columns=["out", "temp", "hum", "wind"])
new_data
```

	out	temp	hum	wind
0	1	2	0	1

```
pred = model.predict(new_data)
```

```
pred
```

```
array([0])
```

De forma a complementar a análise, podemos usar uma função chamada `predict_proba`, que calcula as probabilidades das amostras para cada classe. Essas probabilidades nos ajudam a entender melhor o nosso modelo.

```
model.predict_proba(train_data)
```

```
array([[0., 1.],
       [0., 1.],
       [0., 1.],
       [0., 1.],
       [1., 0.],
       [1., 0.],
       [0., 1.],
       [0., 1.],
       [0., 1.],
       [1., 0.],
       [0., 1.],
       [1., 0.],
       [1., 0.],
       [0., 1.]])
```

Esse método retorna as probabilidades de um determinado dado ser se cada classe. No nosso caso, todas as probabilidades estão sendo 0 ou 1. Isso pode não acontecer sempre.

É comum estabelecermos um limite para a altura da árvore, ou seja, para o número máximo de perguntas que devem ser feitas no modelo. Isso é feito para reduzir o risco de sobreajuste do modelo. No nosso exemplo, vamos, por exemplo, definir a altura máxima da nossa árvore como sendo de três nós:

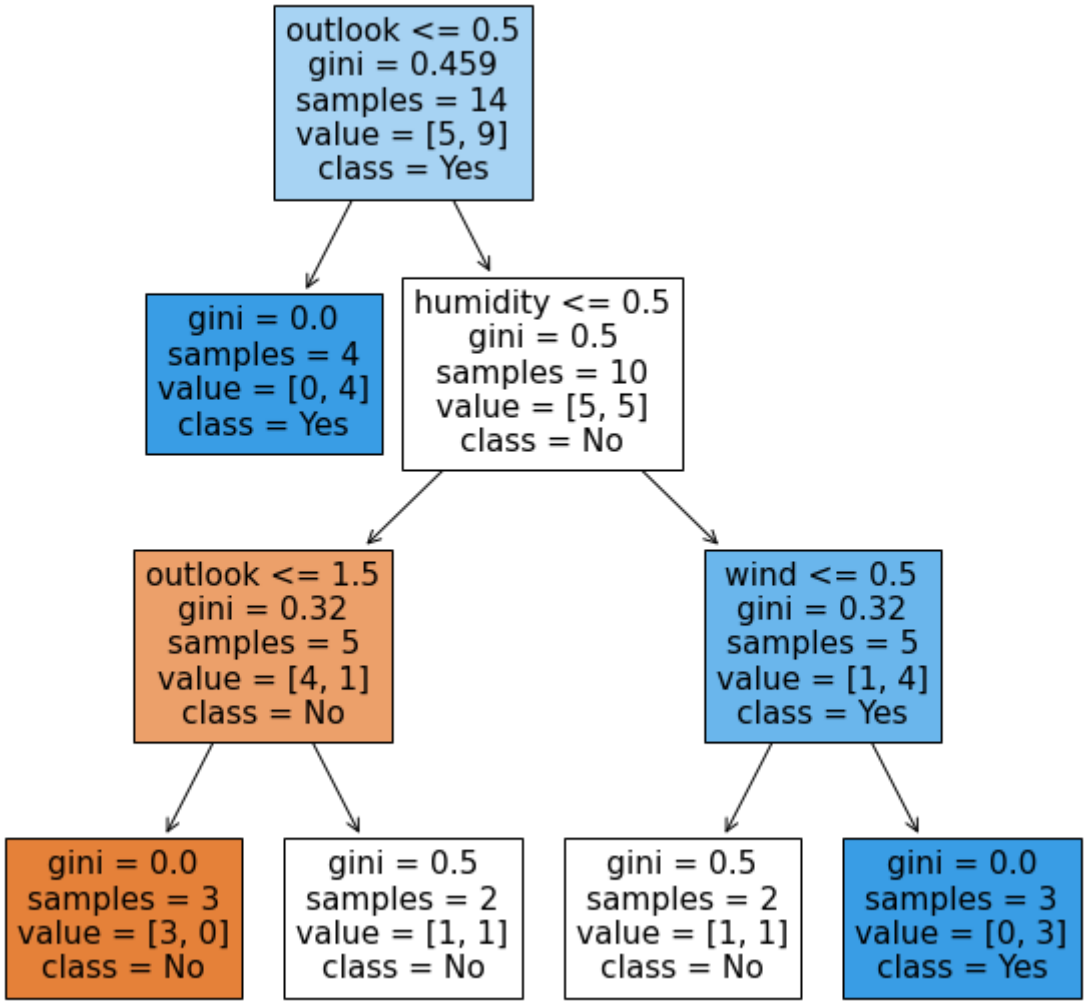
```
model = DecisionTreeClassifier(criterion="gini", max_depth=3)
model.fit(train_data, train_target)
model.predict_proba(train_data)
```

```
array([[0. , 1. ],
       [0. , 1. ],
       [0. , 1. ],
       [0. , 1. ],
       [0.5, 0.5],
       [0.5, 0.5],
       [0.5, 0.5],
       [0. , 1. ],
       [0. , 1. ],
       [1. , 0. ],
       [0.5, 0.5],
       [1. , 0. ],
       [1. , 0. ],
       [0. , 1. ]])
```

Agora estamos vendo alguns casos em que a probabilidade dos dados não é necessariamente 0 ou 1. Vamos analisar o gráfico.

```
fig = plt.figure(figsize=(10,10))
tree.plot_tree(
    model,
    feature_names=train_data.columns,
    class_names=["No", "Yes"],
    filled=True
)
```

```
[Text(209.25, 475.65000000000003, 'outlook <= 0.5\ngini = 0.459\nsamples = 14\nvalue
 Text(139.5, 339.75, 'gini = 0.0\nsamples = 4\nvalue = [0, 4]\nclass = Yes'),
 Text(279.0, 339.75, 'humidity <= 0.5\ngini = 0.5\nsamples = 10\nvalue = [5, 5]\ncla
 Text(139.5, 203.85000000000002, 'outlook <= 1.5\ngini = 0.32\nsamples = 5\nvalue =
 Text(69.75, 67.94999999999999, 'gini = 0.0\nsamples = 3\nvalue = [3, 0]\nclass = No
 Text(209.25, 67.94999999999999, 'gini = 0.5\nsamples = 2\nvalue = [1, 1]\nclass = N
 Text(418.5, 203.85000000000002, 'wind <= 0.5\ngini = 0.32\nsamples = 5\nvalue = [1,
 Text(348.75, 67.94999999999999, 'gini = 0.5\nsamples = 2\nvalue = [1, 1]\nclass = N
 Text(488.25, 67.94999999999999, 'gini = 0.0\nsamples = 3\nvalue = [0, 3]\nclass = Y
```



Duas das três folhas no nosso gráfico não são totalmente conclusivas. Portanto, caso alguma nova amostra caia em um desses nós, o modelo irá definir o valor com base na maior

probabilidade.

```
new_data = pd.DataFrame(np.array([[2, 2, 0, 1]]), columns=["out", "temp", "hum", "wind"])
pred = model.predict(new_data)
pred
```

```
array([0])
```