

## Python Árvore de Decisão

– 0 –

## Definição

- É um método de aprendizado supervisionado não paramétrico que pode ser usado para tarefas de classificação e regressão.
- O objetivo é criar um modelo que prediz o valor de uma variável de destino, aprendendo regras simples de decisão inferidas a partir dos recursos de dados.
- Métodos como árvores de decisão, floresta aleatória (“random forest”) e “gradient boosting” estão sendo muito usados em todos os tipos de problemas de [data science](#). Assim, para os interessados no assunto, é útil aprender esses algoritmos e usá-los em suas modelagens.

Python Árvore de Decisão

– 1 –

Prof. Luiz Alberto -

## Representação de Árvores de Decisão

- Árvores de decisão classificam instâncias ordenando-as árvore abaixo, a partir da raiz até alguma folha.
- Cada nó da árvore especifica o teste de algum atributo da instância
- Cada ramo partindo de um nó corresponde a um dos valores possíveis dos atributos.

Python Árvore de Decisão

– 2 –

Prof. Luiz Alberto -

## Tipos de Árvores de Decisão

- Os tipos de árvore de decisão baseiam-se no tipo de variável de destino que temos. São dois tipos:
  - **Árvore de decisão de variável categórica:** árvore de decisão que tem a variável de destino categórica, chamada assim de árvore de decisão de variável categórica.
  - **Árvore de decisão de variável contínua:** cuja variável alvo é contínua.

Python Árvore de Decisão

– 3 –

Prof. Luiz Alberto -

## Terminologias relacionadas às árvores de decisão

- **Nó Raiz:** Representa a população inteira ou amostra, sendo ainda dividido em dois ou mais conjuntos homogêneos.
- **Divisão:** É o processo de dividir um nó em dois ou mais sub-nós.
- **Nó de Decisão:** Quando um sub-nó é dividido em sub-nós adicionais.
- **Folha ou Nó de Término:** Os nós não divididos são chamados Folha ou Nó de Término.
- **Poda:** O processo de remover sub-nós de um nó de decisão é chamado poda. Podemos dizer que é o processo oposto ao de divisão.

## BD - Exemplo

- Imagine que você joga tênis, porém para isso depende de vários fatores, por exemplo, clima, temperatura, umidade e vento. Gostaria de usar o conjunto de dados abaixo para prever se você deve ou não jogar tênis.
- Uma maneira intuitiva de fazer isso é através de uma Árvore de Decisão.

tempo	temperatura	humidade	ventando	jogar
chuvoso	fria	normal	VERDADEIRO	nao
chuvoso	fria	normal	FALSO	sim
chuvoso	moderada	alta	VERDADEIRO	nao
chuvoso	moderada	alta	FALSO	sim
chuvoso	moderada	normal	FALSO	sim
ensolarado	fria	normal	FALSO	sim
ensolarado	moderada	alta	FALSO	nao
ensolarado	moderada	normal	VERDADEIRO	sim
ensolarado	quente	alta	FALSO	nao
ensolarado	quente	alta	VERDADEIRO	nao
nublado	fria	normal	VERDADEIRO	sim
nublado	moderada	alta	VERDADEIRO	sim
nublado	quente	alta	FALSO	sim
nublado	quente	normal	FALSO	sim

## BD – Exemplo – Árvore de Decisão



**Nó Raiz:** A primeira divisão que decide toda a população ou dados da amostra deve ser dividida em dois ou mais conjuntos homogêneos. No nosso caso, o nó da Perspectiva (Tempo).

**Divisão:** É um processo de dividir um nó em dois ou mais subnós.

**Nó de Decisão:** Este nó decide se / quando um subnó se divide em outros subnós ou não. Aqui temos o nó da Perspectiva, o nó de umidade e o vento.

**Folha:** Nó terminal que prevê o resultado (valor categórico ou contínuo). Os nós coloridos, ou seja, Sim e Não, são as folhas.

## Algoritmo ID3

- O algoritmo da árvore de decisão ID3 usa o Ganho de informação para decidir os pontos de divisão.
- O **algoritmo C4.5**, uma melhoria no ID3 usa a **relação de ganho** como uma extensão do ganho de informações.
- A vantagem de usar a taxa de ganho é lidar com a questão do viés normalizando o ganho de informações usando Informações divididas.

## Algoritmo ID3

- O algoritmo ID3 “aprende” árvores de decisão construindo-as de cima para baixo, começando com a questão:  
“Qual atributo deve ser testado na raiz da árvore?”
- Para responder esta questão, cada atributo da instância é avaliado usando um teste estatístico para determinar quão bem ele sozinho classifica os exemplos de treinamento.

## Entropia

- Para medir a quantidade de informações que obtemos, podemos usar a **entropia** para calcular a homogeneidade de uma amostra.
- É uma medida da quantidade de incerteza em um conjunto de dados. A entropia controla como uma Árvore de Decisão decide dividir os dados. Na verdade, afeta como uma **Árvore de Decisão** desenha seus limites.
- Entropia na Árvore de Decisão significa homogeneidade.
  - Se a amostra for completamente homogênea, a entropia será 0 ( $\text{prob} = 0$  ou  $1$ ) e se a amostra for distribuída igualmente entre as classes, ela terá entropia de 1 ( $\text{prob} = 0,5$ ).

## CART (Árvore de classificação e regressão)

- Outro algoritmo de árvore de decisão CART usa o *método Gini* para criar pontos de divisão, incluindo o *Índice Gini (Gini Impurity)* e *Gini Gain*.
- A definição do Índice de Gini: A probabilidade de atribuir um rótulo errado a uma amostra, escolhendo o rótulo aleatoriamente e também é usada para medir a importância do recurso em uma árvore.

## Construindo uma árvore de decisão usando o Scikit Learn

- O scikit-learn é uma biblioteca de software para machine learning destinada a linguagem Python [11]. Ela possui diversos algoritmos para classificação, regressão e clustering, como support vector machines, random forests, gradient boosting e k-means
- O próprio Scikit-Learn fornece classes muito boas para lidar com dados categóricos.
- Você precisa codificar as variáveis categóricas antes de ajustar uma árvore ao sklearn

## Etapa 1 : Importando Dados

```
import numpy as np
import pandas as pd

df = pd.read_csv('weather.csv')
```

## Etapa 2 : Convertendo variáveis categóricas em dummies

- A maioria dos algoritmos de Machine Learning não conseguem lidar com variáveis categóricas, então precisamos transformar tais variáveis em números.
- Como está, as árvores de decisão do sklearn não tratam dados categóricos
- Você precisa codificar as variáveis categóricas de uma só vez antes de ajustar uma árvore ao sklearn.

## Etapa 2 : Convertendo variáveis categóricas em dummies

- Substituição das seqüências de caracteres por um código de hash
  - deve ser evitada, pois, sendo considerada um recurso numérico contínuo, qualquer codificação que você usará induzirá uma ordem que simplesmente não existe em seus dados.
  - Um exemplo é codificar ['vermelho', 'verde', 'azul'] com [1,2,3], produziria coisas estranhas, como 'vermelho' é menor que 'azul' e, se você fizer uma média de 'vermelho' e um 'azul' você receberá um 'verde'.

## Etapa 2 : Convertendo variáveis categóricas em dummies

- Codificação do recurso categórico em **vários recursos binários**.
  - Por exemplo, você pode codificar ['vermelho', 'verde', 'azul'] com 3 colunas, uma para cada categoria, tendo 1 quando a categoria corresponder e 0 caso contrário.
  - Isso é chamado **de codificação one-hot, codificação binária, codificação one-of-k** ou qualquer outra coisa.
  - Obviamente, a codificação one-hot expandirá seus requisitos de espaço e, às vezes, prejudica o desempenho.

## Etapa 2 : Convertendo variáveis categóricas em dummies

- Por exemplo, a variável Pclass é dividida em três valores: 1, 2 e 3. Embora a classe de cada passageiro certamente tenha algum tipo de relacionamento ordenado, a relação entre cada classe não é a mesma que a relação entre os números 1, 2 e 3. A classe 2 vale o dobro da classe 1 e a classe 3 vale o triplo da classe 1.
- Podemos resolver isso criando uma estrutura chamada Dummy Columns, para cada valor único na coluna Pclass.

## Etapa 2 : Convertendo variáveis categóricas em dummies

- Podemos fazer isso com a função pandas.get\_dummies(), que irá criar colunas exatamente como a imagem acima indica.

Pclass	Pclass_1	Pclass_2	Pclass_3
3	0	0	1
1	1	0	0
3	0	0	1
1	1	0	0
3	0	0	1
3	0	0	1
1	1	0	0
3	0	0	1
3	0	0	1
2	0	1	0

## Etapa 2 : Convertendo variáveis categóricas em dummies

```
df_getdummy=pd.get_dummies(data=df,
columns=['tempo','temperatura','humidade','ventando'])
```

df\_getdummy - DataFrame

Índice	jogar	tempo_chuvoso	tempo_ensolarado	tempo_nublado	temperatura_fria	temperatura_moderada	temperatura_quente	humidade_alta	humidade_normal	ventando_FALSO	ventando_VERDADEIRO
0	nao	0	1	0	0	0	1	1	0	1	0
1	nao	0	1	0	0	0	1	1	0	0	1
2	sim	0	0	1	0	0	1	1	0	1	0
3	sim	1	0	0	0	1	0	1	0	1	0
4	sim	1	0	0	1	0	0	0	1	1	0
5	nao	1	0	0	1	0	0	0	1	0	1
6	sim	0	0	1	1	0	0	0	1	0	1
7	nao	0	1	0	0	1	0	1	0	1	0
8	sim	0	1	0	1	0	0	0	1	1	0
9	sim	1	0	0	0	1	0	0	1	1	0
10	sim	0	1	0	0	1	0	0	1	0	1
11	sim	0	0	1	0	1	0	1	0	0	1
12	sim	0	0	1	0	0	1	0	1	1	0
13	nao	1	0	0	0	1	0	1	0	0	1

## Etapa 3 : Separando o conjunto de treinamento e o conjunto de testes

```
from sklearn.model_selection import train_test_split
X = df_getdummy.drop('jogar',axis=1)
y = df_getdummy['jogar']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30,
random_state=101)
```

## Etapa 4 : importando a Árvore de Decisão via sklearn

```
from sklearn.tree import DecisionTreeClassifier
dtree = DecisionTreeClassifier(criterion='entropy',max_depth=3)
dtree.fit(X_train,y_train)
predictions = dtree.predict(X_test)
```

## sklearn.tree.DecisionTreeClassifier - Parâmetros

- **criterion**{“gini”, “entropy”}, default=“gini”
  - A função para medir a qualidade de uma divisão. Os critérios suportados são "gini" para a impureza de Gini e "entropia" para o ganho de informações.
- **splitter**{“best”, “random”}, default=“best”
  - A estratégia usada para escolher a divisão em cada nó. As estratégias suportadas são “best” para escolher a melhor divisão e “random” para escolher a melhor divisão aleatória.
- **max\_depth** int, default=None
  - A profundidade máxima da árvore. Se Nenhum, os nós serão expandidos até que todas as folhas estejam puras ou até que todas as folhas contenham menos de amostras min\_samples\_split.

Ver outros parâmetros em: <https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html>

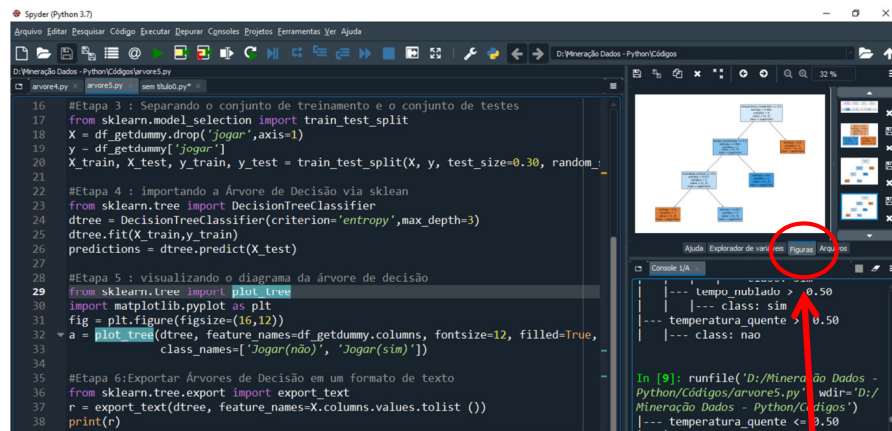
## sklearn.tree.DecisionTreeClassifier - Parâmetros

- **min\_samples\_split** int ou float, padrão = 2
  - O número mínimo de amostras necessárias para dividir um nó interno:
    - Se int, considere min\_samples\_split como o número mínimo.
    - Se flutuar, então min\_samples\_split é uma fração e é o número mínimo de amostras para cada divisão.  $\text{ceil}(\text{min\_samples\_split} * n\_samples)$

## Etapa 5 : visualizando o diagrama da árvore de decisão

```
from sklearn.tree import plot_tree
import matplotlib.pyplot as plt
fig = plt.figure(figsize=(16,12))
a = plot_tree(dtree, feature_names=df_getdummy.columns, fontsize=12,
              filled=True,
              class_names=['Jogar(não)', 'Jogar(sim)'])
```

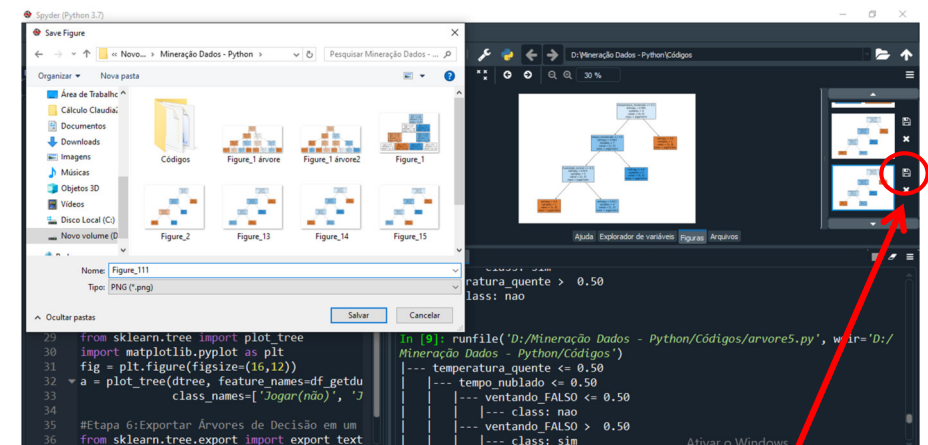
## Etapa 5 : visualizando o diagrama da árvore de decisão



```
16 #Etapa 3 : Separando o conjunto de treinamento e o conjunto de testes
17 from sklearn.model_selection import train_test_split
18 X = df.getdummy.drop("jogar",axis=1)
19 y = df.getdummy["jogar"]
20 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30, random_
21
22 #Etapa 4 : importando a Árvore de Decisão via sklearn
23 from sklearn.tree import DecisionTreeClassifier
24 dtree = DecisionTreeClassifier(criterion='entropy',max_depth=3)
25 dtree.fit(X_train,y_train)
26 predictions = dtree.predict(X_test)
27
28 #Etapa 5 : visualizando o diagrama da árvore de decisão
29 from sklearn.tree import plot_tree
30 import matplotlib.pyplot as plt
31 fig = plt.figure(figsize=(16,12))
32 a = plot_tree(dtree, feature_names=df.getdummy.columns, fontsize=12, filled=True,
33              class_names=["Jogar(não)", "Jogar(sim)"])
34
35 #Etapa 6:Exportar Árvores de Decisão em um formato de texto
36 from sklearn.tree.export import export_text
37 r = export_text(dtree, feature_names=X.columns.values.tolist ())
38 print(r)
```

Clique nessa aba para visualizar a árvore

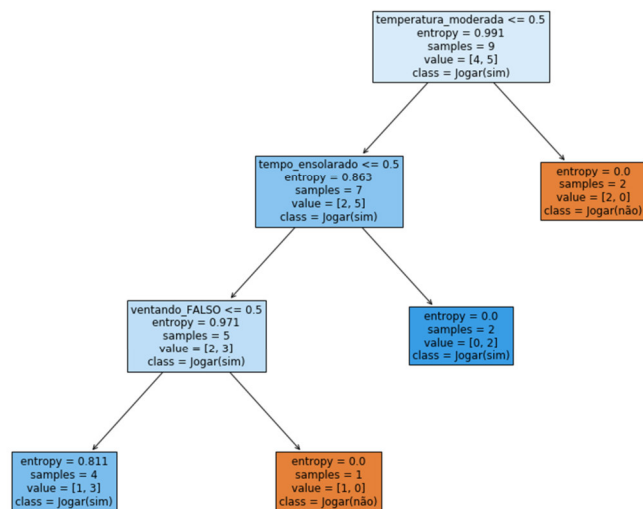
## Etapa 5 : visualizando o diagrama da árvore de decisão



```
29 from sklearn.tree import plot_tree
30 import matplotlib.pyplot as plt
31 fig = plt.figure(figsize=(16,12))
32 a = plot_tree(dtree, feature_names=df.getdu
33              class_names=["Jogar(não)", 'J
34
35 #Etapa 6:Exportar Árvores de Decisão em
36 from sklearn.tree.export import export_text
```

Clique nesse botão para salvar a árvore

## Etapa 5 : visualizando o diagrama da árvore de decisão



## Etapa 6:Exportar Árvores de Decisão em um formato de texto

```
from sklearn.tree.export import export_text
r = export_text(dtree, feature_names=X.columns.values.tolist ())
print(r)
```

```
--- temperatura_quente <= 0.50
|--- tempo_nublado <= 0.50
|   |--- humidade_normal <= 0.50
|   |   |--- class: nao
|   |   |--- humidade_normal > 0.50
|   |   |--- class: sim
|   |--- tempo_nublado > 0.50
|   |--- class: sim
|--- temperatura_quente > 0.50
|--- class: nao
```

[https://scikit-learn.org/stable/modules/generated/sklearn.tree.export\\_text.html](https://scikit-learn.org/stable/modules/generated/sklearn.tree.export_text.html)



# Arquivo completo

```
#Step 1: Importing data
import numpy as np
import pandas as pd
```

```
df = pd.read_csv('jogar2.csv')
```

```
#Etapa 2 : Convertendo variáveis categóricas em
dummies / variáveis indicadoras
df_getdummy=pd.get_dummies(data=df,
columns=['tempo','temperatura','humidade','ventando'])
```

```
#Etapa 3 : Separando o conjunto de treinamento e o
conjunto de testes
from sklearn.model_selection import train_test_split
X = df_getdummy.drop('jogar',axis=1)
y = df_getdummy['jogar']
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.30, random_state=101)
```

```
#Etapa 4 : importando a Árvore de Decisão via sklearn
from sklearn.tree import DecisionTreeClassifier
dtree = DecisionTreeClassifier(criterion='entropy',max_depth=3)
dtree.fit(X_train,y_train)
predictions = dtree.predict(X_test)
```

```
#Etapa 5 : visualizando o diagrama da árvore de decisão
from sklearn.tree import plot_tree
import matplotlib.pyplot as plt
fig = plt.figure(figsize=(16,12))
a = plot_tree(dtree, feature_names=df_getdummy.columns,
fontsize=12, filled=True,
class_names=['Jogar(não)', 'Jogar(sim)'])
```

```
#Etapa 6:Exportar Árvore de Decisão em um formato de texto
from sklearn.tree.export import export_text
r = export_text(dtree, feature_names=selected_atributes)
print(r)
```

# Árvore de decisão na prática (dados contínuos)

```
1 # Importando as bibliotecas necessárias:
2 import pandas as pd
3 from sklearn.tree import DecisionTreeClassifier,export_graphviz
4 from sklearn.model_selection import train_test_split
5 from sklearn import metrics
6 import numpy as np
```

```
1 # Carregando a base de dados:
2 df_diabetes = pd.read_csv('sample_data/diabetes.csv')
3 df_diabetes.head()
```

```
# Dividindo os dados em treino e teste:
X_train, X_test, y_train, y_test = train_test_split(df_diabetes.drop('Outcome',axis=1),df_diabetes['Outcome'],test_size=0.3)
```

```
# Instanciando o objeto classificador:
clf = DecisionTreeClassifier()
```

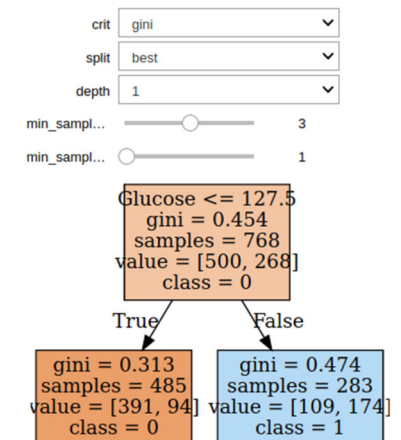
```
# Treinando o modelo de árvore de decisão:
clf = clf.fit(X_train,y_train)
```

# Árvore de decisão na prática (dados contínuos)

```
1 from sklearn import metrics
2 print(metrics.classification_report(y_test,resultado))
3
4 # Resultado do classification_report:
5
6          precision    recall  f1-score   support
7
8     0           0.75     0.82    0.78       150
9     1           0.60     0.49    0.54       81
10
11    accuracy                0.71       231
12   macro avg           0.67    0.66    0.66       231
13  weighted avg           0.70    0.71    0.70       231
```

# Árvore de decisão na prática (dados contínuos)

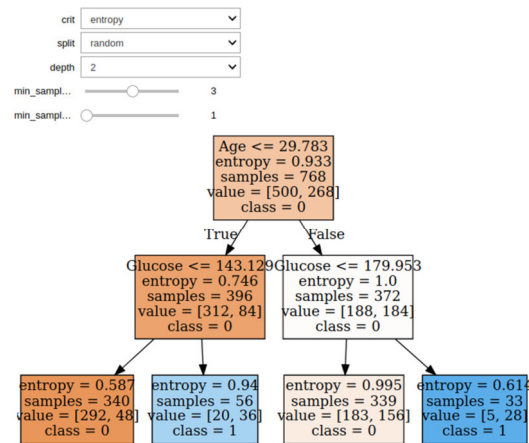
```
1 # Renderizando a árvore de forma interativa:
2 from ipynbwidgets import interactive
3 from IPython.display import SVG,display
4 from graphviz import Source
5
6 # Feature matrix
7 X,y = df_diabetes.drop('Outcome',axis=1),df_diabetes['Outcome']
8
9 # Feature labels
10 features_label = df_diabetes.drop('Outcome',axis=1).columns
11
12 # class label
13 class_label = ['0','1']
14
15
16 def plot_tree(crit, split, depth, min_samples_split, min_samples_leaf=0.2):
17     estimator = DecisionTreeClassifier(
18         ,criterion=crit
19         ,splitter=split
20         ,max_depth=depth
21         ,min_samples_split=min_samples_split
22         ,min_samples_leaf=min_samples_leaf
23     )
24     estimator.fit(X, y)
25     graph = Source(export_graphviz(estimator
26     , out_file=None
27     , feature_names=features_label
28     , class_names=class_label
29     ,separity=True
30     , filled = True))
31     display(SVG(graph.pipe(format='svg')))
32     return estimator
33
34
35 inter-interactive(plot_tree
36     ,crit = ['gini','entropy']
37     ,split = ['best','random']
38     ,depth = [1,2,3,4,5,10,20,30]
39     ,min_samples_split=(1,5)
40     ,min_samples_leaf=(1,5))
41
42 display(inter)
```





# Árvore de decisão na prática (dados contínuos)

```
1 # Renderizando a árvore de forma interativa:
2 from IPython.display import Interactive
3 from IPython.display import SVG, display
4 from graphviz import Source
5
6 # Feature matrix
7 x, y = df_diabetes.drop('Outcome', axis=1), df_diabetes['Outcome']
8
9 # Feature labels
10 features_label = df_diabetes.drop('Outcome', axis=1).columns
11
12 # Class label
13 class_label = ['0', '1']
14
15 def plot_tree(crit, split, depth, min_samples_split, min_samples_leaf=0.2):
16     estimator = DecisionTreeClassifier(
17         random_state=0,
18         criterion=crit,
19         splitter=split,
20         max_depth=depth,
21         min_samples_split=min_samples_split,
22         min_samples_leaf=min_samples_leaf
23     )
24     estimator.fit(x, y)
25     graph = Source(export_graphviz(estimator,
26                                   out_file=None,
27                                   feature_names=features_label,
28                                   class_names=class_label,
29                                   impurity=True,
30                                   filled=True))
31     display(SVG(graph.pipe(format='svg')))
32     return estimator
33
34 inter-interactive(plot_tree
35                  , crit = ['gini', 'entropy']
36                  , split = ['best', 'random']
37                  , depth=[1,2,3,4,5,10,20,30]
38                  , min_samples_split=(1,5)
39                  , min_samples_leaf=(1,5))
40
41 display(inter)
```



- 32 -

Prof. Luiz Alberto -

## Vantagens

- **Fácil de entender:** A visualização de uma árvore de decisão torna o problema fácil de compreender, mesmo para pessoas que não tenham perfil analítico. Sua representação gráfica é muito intuitiva e permite relacionar as hipóteses também facilmente.
- **Útil em exploração de dados:** A árvore de decisão é uma das formas mais rápidas de identificar as variáveis mais significativas e a relação entre duas ou mais variáveis.
- **Menor necessidade de limpar dados:** Requer menos limpeza de dados em comparação com outras técnicas de modelagem. Até um certo nível, não é influenciado por pontos fora da curva “outliers” nem por valores faltantes (“missing values”).
- **Não é restrito por tipos de dados:** Pode manipular variáveis numéricas e categóricas.

Python Árvore de Decisão

- 33 -

Prof. Luiz Alberto -

## Desvantagem

- **Sobreajuste (“Over fitting”):** Sobreajuste é uma das maiores dificuldades para os modelos de árvores de decisão. Este problema é resolvido através da definição de restrições sobre os parâmetros do modelo e da poda (discutido em mais detalhes abaixo).
- **Não adequado para variáveis contínuas:** ao trabalhar com variáveis numéricas contínuas, a árvore de decisão perde informações quando categoriza variáveis em diferentes categorias.

Python Árvore de Decisão

- 34 -

Prof. Luiz Alberto -

## Referências

- <https://www.vebuso.com/2020/01/decision-tree-intuition-from-concept-to-application/>
- <https://www.vooo.pro/insights/um-tutorial-completo-sobre-a-modelagem-baseada-em-tree-arvore-do-zero-em-r-python/>
- <https://minerandodados.com.br/arvores-de-decisao-conceitos-e-aplicacoes/>
- <https://towardsdatascience.com/understanding-decision-tree-classification-with-scikit-learn-2ddf272731bd>
- <https://glowingpython.blogspot.com/2019/06/exporting-decision-tress-in-textual.html>

Python Árvore de Decisão

- 35 -

Prof. Luiz Alberto -