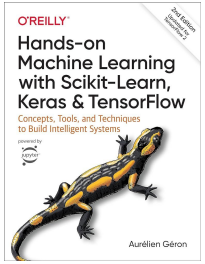
Árvores de decisão

Pedro Henrique da Silva Hinerasky

Bibliografia



- *Géron, A. (2019). Hands-on machine learning with Scikit-Learn, Keras and TensorFlow: concepts, tools, and techniques to build intelligent systems (2nd ed.). O'Reilly.*

Sumário

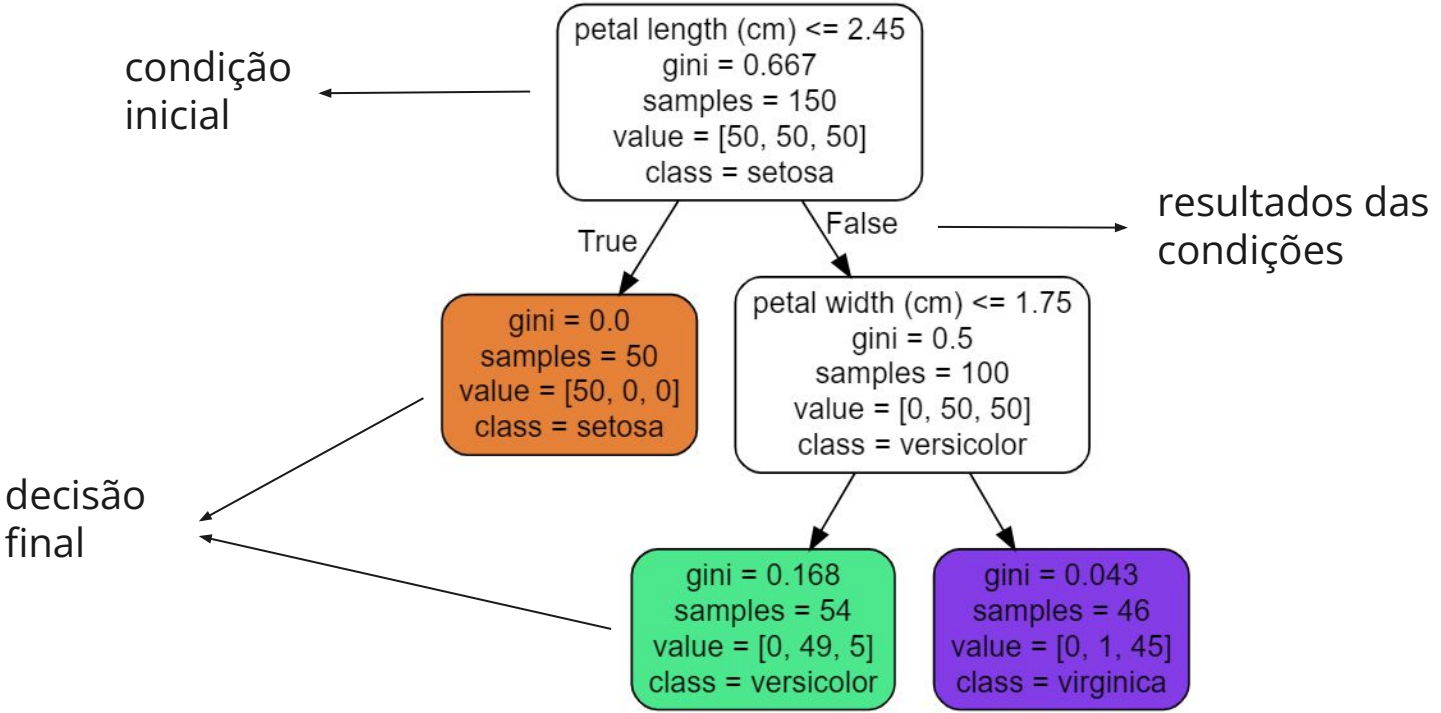
Introdução

Árvores de Decisão

- São algoritmos de aprendizagem supervisionada muito utilizada na estatística e mineração de dados, conhecidos pela facilidade no uso e entendimento, além de seu grande poder.
- Possuem uma estrutura hierárquica que inclui um nó raiz, nós internos que representam testes ou decisões baseadas em características dos dados, e nós folha que representam os resultados finais ou categorias.
- São utilizadas principalmente na Classificação, mas também podem ser usadas com Regressão.
- Árvores de decisão são a base para *Random Forests*, que combinam múltiplas árvores para melhorar a precisão e a robustez das previsões.
- Como treinar, visualizar e fazer predições com Árvores de Decisão?

Treinamento e Visualização

Treinamento e Visualização



Fazendo Previsões

Como é feita a Classificação ?

- Sempre inicia-se pelo nó raiz.

- O nó fará uma pergunta, indicando se o próximo nó deve ser o filho da esquerda ou da direita.

- Se o próximo nó for um nó folha (sem filhos), ele não fará mais nenhuma pergunta e dirá a qual classe a Árvore previu para esse item.

- Caso o nó filho não for um nó folha, ele fará outra pergunta, que levará para mais dois filhos, até chegar em algum nó folha.

Atributos dos Nós

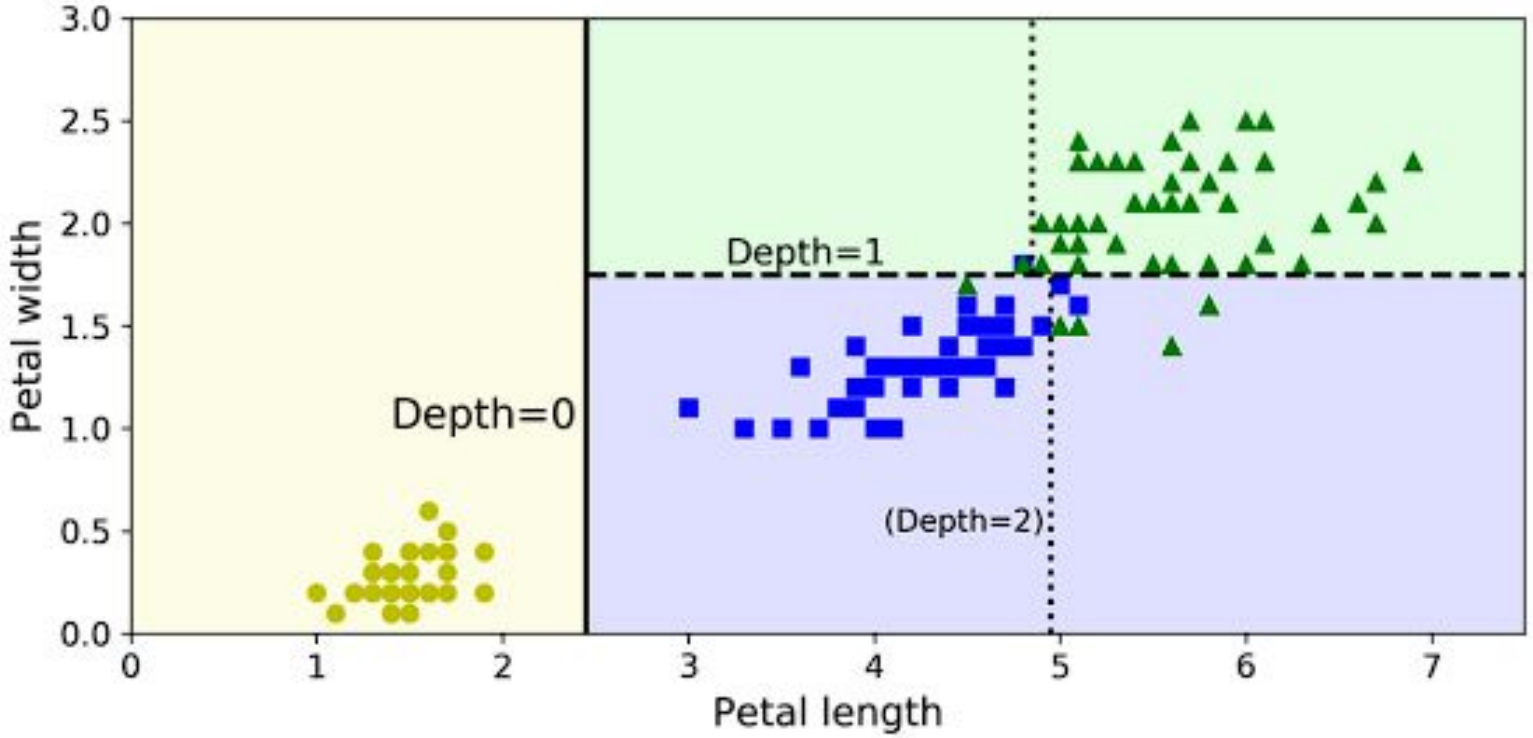
- *Samples*: número de instâncias do conjunto inicial de treino alcançam esse nó durante a criação da Árvore.

- *Value*: o número de instâncias de cada classe do conjunto de treino presente neste nó.

- *Gini*: atributo que mede o grau de impureza do nó. Um nó é totalmente puro (*gini = 0*) quando todas as instâncias de treino se aplicam a mesma classe.

- *Class*: se refere à classe mais comum entre as amostras do nó. Em um nó folha, representa qual a classificação do item que chegou neste nó.

Limites da Árvore de Decisão



Estimativa de Probabilidades de Classe

Estimativa de Probabilidades de Classe

- Além de apenas classificar, a Árvore de Decisão pode averiguar a probabilidade de uma instância pertencer a cada uma das classes.

- Ao percorrer a Árvore até o nó folha referente a esta instância, retorna-se a razão do número de instâncias de cada classe pelo total de *samples* no nó.

- Com isso, é possível ter uma porcentagem que infere a probabilidade da instância pertencer a cada classe. Isso se torna útil em Árvores que possuem alguns nós com *Gini* um pouco elevado.

Algoritmo de Treinamento CART

Algoritmo de Treinamento CART

- Algoritmo utilizado pelo Scikit-Learn.
- Separa o *training set* em dois *subsets*, utilizando de uma *feature k* e um *threshold* t_k
- Escolhe a combinação de feature e threshold que produzem os subsets mais puros possíveis.
- Após separar o training set sem dois subsets, separa os subsets utilizando da mesma lógica, até atingir a profundidade máxima ou uma divisão que não diminui a impureza.
- O Algoritmo CART é um Algoritmo Guloso.

Equações do Algoritmo de CART

Função de custo de classificação:

$$J\left(k, t_k\right) = \frac{m_{left}}{m} G_{left} + \frac{m_{right}}{m} G_{right}$$

onde:

- k é a *feature*
- t_k é o *threshold*
- m é o número de instâncias
- G é o grau de impureza *Gini*

Medidas de Impureza

Medidas de Impureza

Gini

Índice de Gini é uma métrica que mede a impureza de um conjunto de dados. Ele indica a probabilidade de uma amostra ser classificada incorretamente se for aleatoriamente rotulada de acordo com a distribuição das classes no conjunto de dados.

Fórmula

$$G_i = 1 - \sum_{k=1}^{n} p_{i,k}2$$

Onde $p_{i,k}$ é a razão das instâncias da classe k dentre as instâncias de treino do nó i.

Medidas de Impureza

Entropia

Entropia é uma medida de incerteza ou impureza de um conjunto de dados. É originada da teoria da informação e quantifica a quantidade de incerteza envolvida na previsão da classe de uma amostra aleatória do conjunto.

Fórmula

$$H_i = - \sum_{\substack{k=1 \\ p_{i,k} \neq 0}}^{n} p_{i,k} \log_2 \left(p_{i,k}\right)$$

Onde $p_{i,k}$ é a razão das instâncias da classe k dentre as instâncias de treino do nó i.

Características

Gini

- Um Gini de 0 indica que todas as amostras pertencem a uma única classe (nó puro).
- Um Gini próximo de 0 indica alta pureza.
- Um Gini de 0.5 indica máxima impureza em um problema de duas classes (distribuição uniforme).

Entropia

- Uma entropia de 0 indica que todas as amostras pertencem a uma única classe (nó puro).
- Uma entropia alta indica maior incerteza ou impureza.
- Para um problema de duas classes, a entropia máxima é 1 (distribuição uniforme).

Qual usar ?

- O Algoritmo de CART utiliza por padrão Gini, porém é possível utilizar a Entropia modificando os hiperparâmetros de critério.

- Em muitos casos a escolha acaba não fazendo muita diferença, pois ambas técnicas acabam gerando Árvores muito similares.

- Por não envolver cálculos com logaritmos, Gini tende a ser mais rápido de computar, tornando-o um bom parâmetro *default*.

- Gini tende a isolar a classe mais frequente no seu próprio galho da Árvore, enquanto a Entropia tende a produzir Árvores mais balanceadas.

Hiperparâmetros de Regularização

O Problema de Overfitting

- Árvores de Decisão são *Modelos Não Paramétricos*, ou seja, não possuem um número definido de parâmetros. Por mais que isso aumente a liberdade do modelo para se adaptar aos dados fornecidos, o risco de *overfitting* aumenta de maneira significativa.

- A fim de evitar o *overfitting*, é necessário limitar essa liberdade durante o treinamento. Esse processo é chamado de Regularização.

- Para limitar a liberdade da Árvore de Decisão, utilizamos de certos Hiperparâmetros. Estes Hiperparâmetros dependem de qual algoritmo foi utilizado, porém seguem funcionalidades parecidas.

Hiperparâmetros do Scikit-Learn

- ***max_depth***: Controla a profundidade máxima da Árvore, impedindo que ela cresça excessivamente. É um dos Hiperparâmetros mais comumente utilizados.

- ***min_samples_split***: Define o número mínimo de amostras necessárias para dividir um nó, evitando nós com poucas instâncias.

- ***min_samples_leaf***: Define um número mínimo de amostras em um nó, evitando padronizar casos específicos do set de treinamento.

- ***min_weight_fraction_leaf***: Similar ao *min_samples_leaf*, porém utiliza-se de uma porcentagem do subset original ao invés de um número.

- ***max_leaf_node****:* Define um número máximo de nós folhas, reduzindo a complexidade da Árvore.

- ***max_features***: Número máximo de *features* que são avaliadas para a divisão de cada nó.

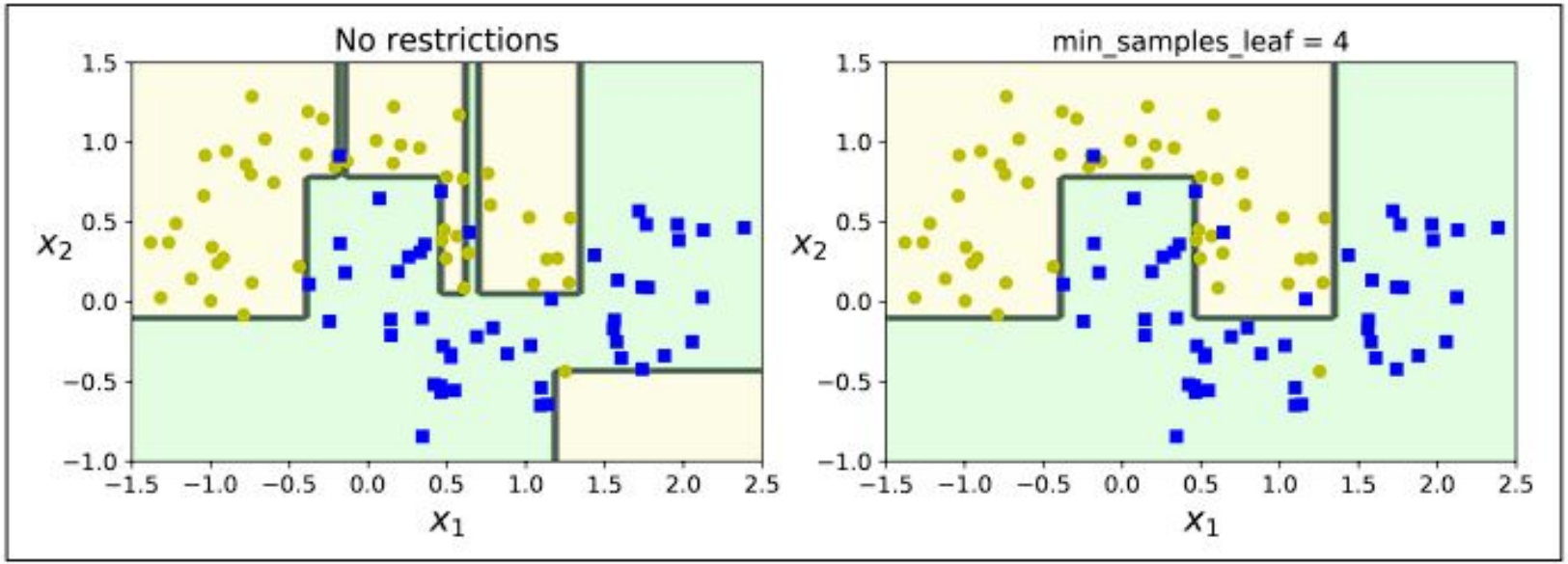
Como Regularizar a Árvore ?

- Ao utilizar os Hiperparâmetros, evitar o *overfitting* se torna mais fácil. Aumentando os Hiperparâmetros iniciados por *min* e diminuindo os Hiperparâmetros iniciados por *max*, grande parte dos problemas serão neutralizados.

- Existem outras formas de lidar com o *overfitting* além dos Hiperparâmetros. A principal sendo a técnica da *poda*, que envolve remover partes da Árvore que possuem pouca importância para o resultado final.

- Outro método muito utilizado é o Cross-Validation, que tem como base dividir o set de treinamento em várias partes, utilizando certas partes para treino e outras para validação, alternando entre elas. Com isso, a Árvore fica mais generalizada.

A Importância da Regularização



Árvores de Decisão para Regressão

Árvores de Decisão para Regressão

- Diferentemente das Árvores de Classificação, as Árvores de Regressão tem como objetivo prever um valor contínuo ao invés de uma classe discreta.

- Possuem uma estrutura semelhante, porém em vez de uma classe, cada folha contém um valor contínuo que é a média (ou mediana) dos valores dos dados que chegam a essa folha.

- Oferecem uma abordagem intuitiva e flexível para modelar relações não lineares entre variáveis.

Construção da Árvore de Regressão

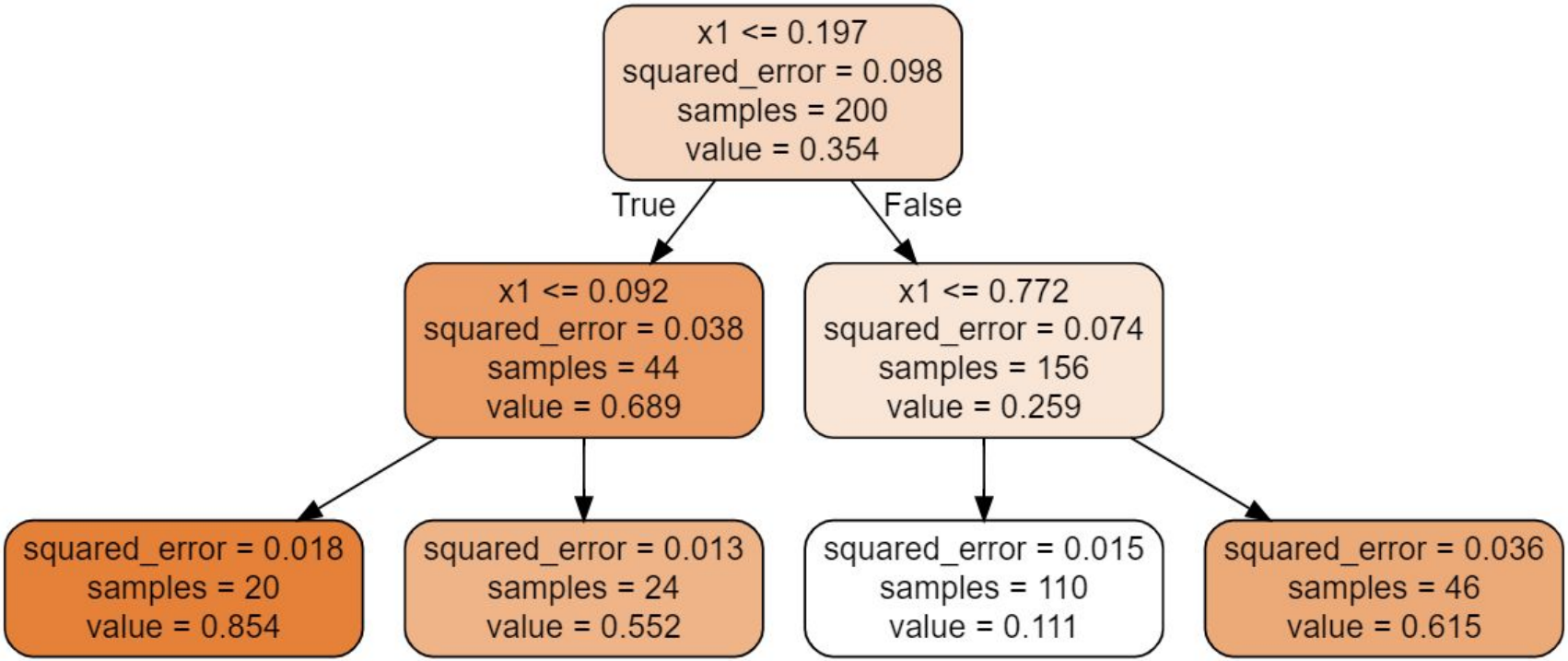
- O Algoritmo CART funciona de maneira similar à Classificação, porém ao invés de separar o set de treinamento de uma maneira que diminua a impureza, agora tentamos diminuir o MSE.

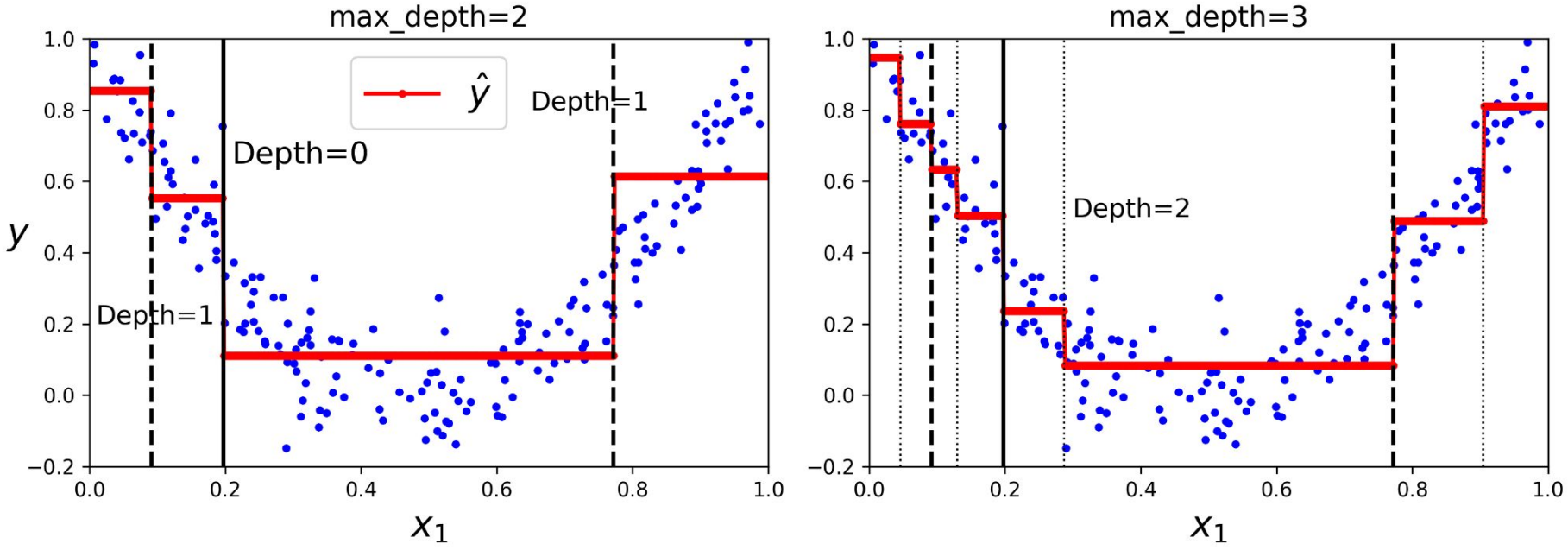
- MSE (Mean Squared Error): Soma dos Erros Quadráticos entre os valores previstos e os valores reais. Quanto menor seu valor, mais pura é a divisão.

$$MSE = \frac{1}{n}\sum_{i=1}^{n}\left(y_i - \hat{y}\right)^2$$

onde:

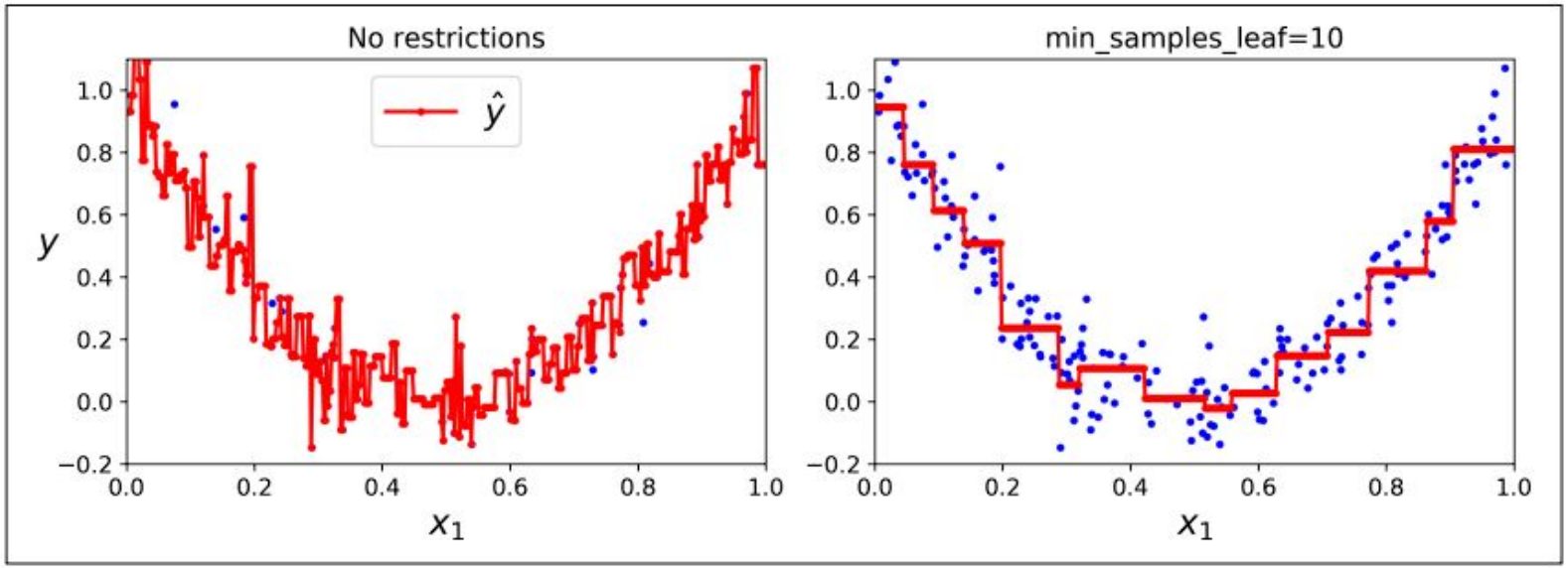
- n é o número de amostras no conjunto de dados.
- y_i é o valor real da i-ésima amostra.
- $\hat{y}_i$ é o valor previsto pelo modelo para a i-ésima amostra.


x1 <= 0.197
squared_error = 0.098
samples = 200
value = 0.354
True
False
x1 <= 0.092
squared_error = 0.038
samples = 44
value = 0.689
x1 <= 0.772
squared_error = 0.074
samples = 156
value = 0.259
squared_error = 0.018
samples = 20
value = 0.854
squared_error = 0.013
samples = 24
value = 0.552
squared_error = 0.015
samples = 110
value = 0.111
squared_error = 0.036
samples = 46
value = 0.615


max_depth=2
max_depth=3
$\hat{y}$
Depth=0
Depth=1
Depth=1
Depth=2
y
x_1
x_1
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
0.0
0.2
0.4
0.6
0.8
1.0

Construção da Árvore de Regressão

Assim como em Árvores de Classificação, as Árvores de Decisão também estão sujeitas ao *overfitting*, tornando sempre importante a aplicação da Regularização.



Limitações

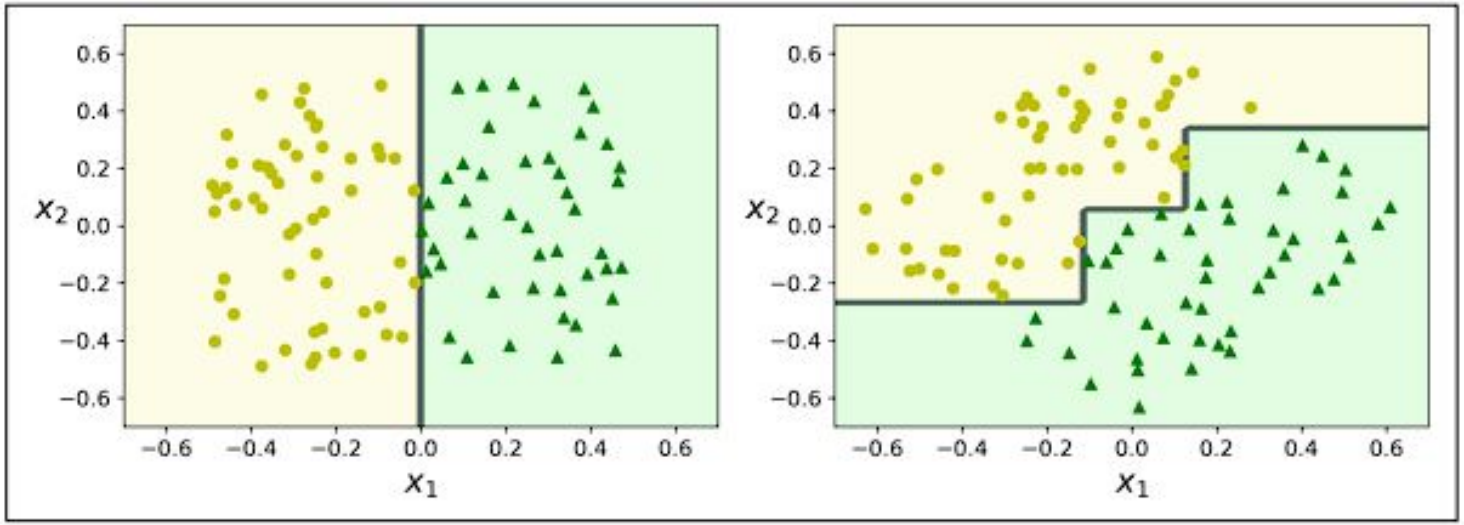
Limitações

- Possuem grande propensão ao *Overfitting*, se adaptando aos dados de treinamento. A principal técnica de mitigação é a Regularização da Árvore.

- Possuem grande sensibilidade a pequenas variações nos dados, podendo gerar Árvores completamente diferentes em uma pequena mudança. Para evitar isso, são utilizadas as Random Forests que reduzem a variabilidade.

- Como todas as divisões são perpendiculares a um dos eixos das *features*, um set de dados rotacionado pode aumentar a complexidade da Árvore. Uma maneira de mitigar esse problema é usar técnicas de transformação de dados como o PCA, que resultam em uma melhor orientação dos dados.

Sensitividade à Rotação do Set de Dados



EXERCÍCIO FINAL

7. Train and fine-tune a Decision Tree for the moons dataset.
   a. Generate a moons dataset using `make_moons(n_samples=10000, noise=0.4)`.
   b. Split it into a training set and a test set using `train_test_split()`.
   c. Use grid search with cross-validation (with the help of the `GridSearchCV` class) to find good hyperparameter values for a `DecisionTreeClassifier`. Hint: try various values for `max_leaf_nodes`.
   d. Train it on the full training set using these hyperparameters, and measure your model's performance on the test set. You should get roughly 85% to 87% accuracy.

Obrigado pela atenção.