# Árvores de Decisão

## Introdução à Pruning
## Evitando Overfitting

Tiago Oliveira Weber
2024

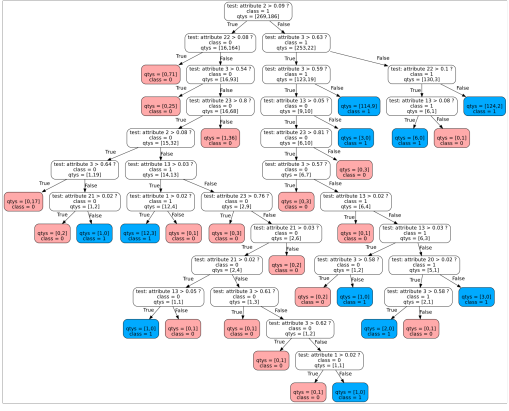
# Tópicos

- Introdução à Árvores de Decisão
- Impureza Gini
- Escolha de Atributo para Divisão
- Estratégias para Evitar Overfitting
  - Pruning
  - *Ensemble Methods* (Métodos em Conjunto ou Agrupamento)

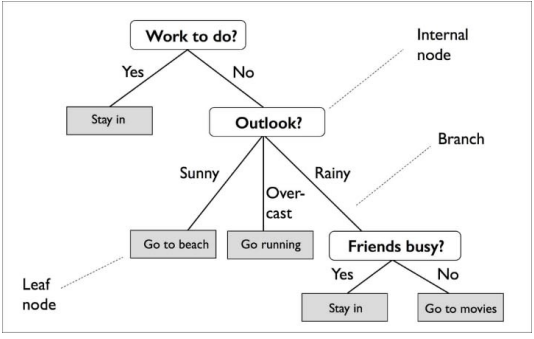
O que foi visto nas últimas aulas

- apresentação da disciplina
  - com intro a uso de HW em Inteligência computacional;
- noções iniciais de aprendizado de máquina;
- KNN e K-means

- um dos métodos mais simples, mas ainda assim poderoso de aprendizado de máquina
- a árvore chega a sua decisão através de uma sequência de testes



- cada nó representa o teste do valor de um dos atributos de entrada ($A_i$)
- os ramos do nós são rotulados com os possíveis valores do atributo ($v_{ik}$)
  - em uma árvore binária, verdadeiro ou falso
- as folhas (nós sem filhos) representam os valores a serem retornados pela função




Fonte: imagem extraída de "Classification and Regression Analysis with Decision Trees"por Lorraine, disponível em https://dev.to/nexttech/classification-and-regression-analysis-with-decision-trees-jgp

- Exemplo simples: XOR

- Árvores de Decisão - Exemplo

- análise da criação de uma árvore de decisão booleana (saída = verdeiro ou falso)
- objetivo: criar árvore que seja:
  - consistente com os exemplos
  - tão pequena quanto possível
- estratégia de dividir e conquistar:
  - testar o atributo mais importante primeiro
  - a partir do resultado, subdidivir o problema em problemas menores
  - resolvê-los recursivamente

Possíveis casos (observando os resultados do teste do atributo em um nó):

- exemplos restantes do nó são todos positivos ou todos negativos. Nesse caso, este é um nó folha
- há alguns exemplos positivos e outros negativos. Nesse caso, escolher melhor atributo e dividí-los (repetir o processo)
- sem exemplos restantes. Significa que não há amostras para essa combinação de atributos. Retornar valor baseado na classificação da pluralidade dos exemplos do nó pai.
- sem atributos restantes (todos já foram testados), mas ainda há exemplos positivos e negativos. Significa estes exemplos têm a mesma descrição, mas classificações diferentes. Retornar valor baseado na classificação da pluralidade dos exemplos do nó pai.
  - Possíveis causas: ruído ou falta de atributo importante

- Pseudocódigo

```
1:  função APRENDIZADOÁRVOREDEDECISÃO(exemplos, atributos, exemplos paternos)
2:      se exemplos está vazio então
3:          retorna ValorPluralidade(exemplos paternos)
4:      senão se todos exemplos têm mesma classificação então
5:          retorna classificação
6:      senão se atributos está vazio então
7:          retorna ValorPluralidade(exemplos paternos)
8:      senão
9:          A ← argmax_{a∈atributos} importância(a, exemplos)
10:         para cada valor v_k de A faça
11:             exs ← {ex : ex ∈ exemplos e ex.A = v_K}
12:             subárvore ←AprendizadoÁrvoreDeDecisão(exs,atributos-A,exemplos)
13:             adicionar ramo à árvore com rótulo A = v_k e subárvore subárvore
14:         fim para
15:         retorna árvore
```

argmax: ponto do domínio no qual o valor da função é máximo.

baseado em RUSSEL, S.; NORVIG, P. Artificial Intelligence: A Modern Approach. 3rd edition. Prentice Hall Press, 2009

# Teste de Atributos (função "importância")

Formas mais usais:

- baseado no ganho de informação, definido em termos de entropia
  - Entropia: medida de incerteza de uma variável aleatória
  - quanto mais informação, menos entropia
- baseado na medida de impureza da divisão
  - índice GINI, que mede a "impureza"de um nó

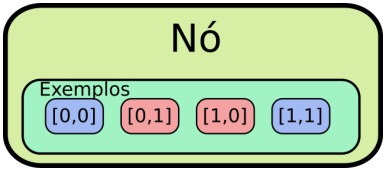
- mede a impureza de um nó
- nó que tenha exemplos restantes de apenas uma classe:
  - nó puro (gini = 0)

$$G_i = 1 - \sum_{k=1}^{n} p_k^2$$

onde:

- $G_i$ é o índice Gini
- $n$ é o número total de classes de saída do problema

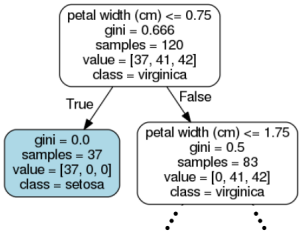
$$p_k = \frac{\text{exemplos de classe k presentes no nó}}{\text{total de exemplos do nó}}$$




**Cálculo**

$$G_i = 1 - \left[ \left( \frac{2}{4} \right)^2 + \left( \frac{2}{4} \right)^2 \right]$$

$$G_i = 1 - [0,25 + 0,25] = 0,5$$




## Cálculos

$$G_i = 1 - \left[ \left( \frac{37}{120} \right)^2 + \left( \frac{41}{120} \right)^2 + \left( \frac{42}{120} \right)^2 \right] = 0,666$$

$$G_i = 1 - \left[ \left( \frac{0}{83} \right)^2 + \left( \frac{41}{83} \right)^2 + \left( \frac{42}{83} \right)^2 \right] = 0,500$$

$$G_i = 1 - \left[ \left( \frac{37}{37} \right)^2 + \left( \frac{0}{37} \right)^2 + \left( \frac{0}{37} \right)^2 \right] = 0,000$$

- Pode-se usar a impureza para avaliar qual é o melhor atributo para fazer determinada separação na criação da árvore de decisão
  - Queremos divisões que resultem em baixa impureza;

**Escolha do Atributo em Nó através do Índice Gini**

- Calcula-se o Gini para cada possível atributo restante;
- Para cada atributo, faz-se a média ponderada da impureza Gini para os nós-filho da divisão;

$$MP = \sum_{i=0}^{K} w_i \cdot Gi_i$$

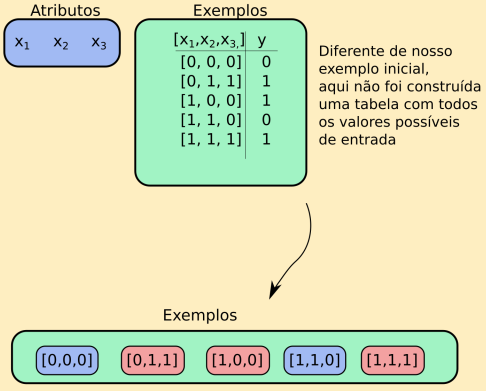
onde:

- $w_i = \frac{\text{total de exemplos do nó filho}}{\text{total de exemplos paternos}}$
- K é o número de valores de saída do atributo (número de nós-filhos)

- Escolhe-se a divisão de atributo que gera a menor média ponderada

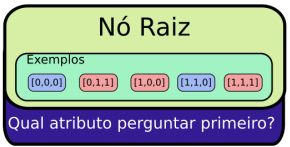
# Exemplo da Escolha do Atributo

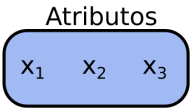
## Instâncias usadas neste problema de criação de árvore



# Exemplo da Escolha do Atributo



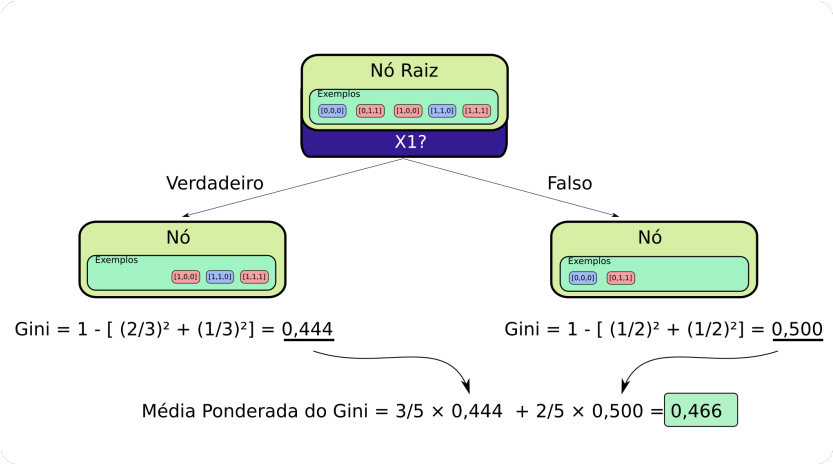



Vamos analisar o efeito de cada atributo

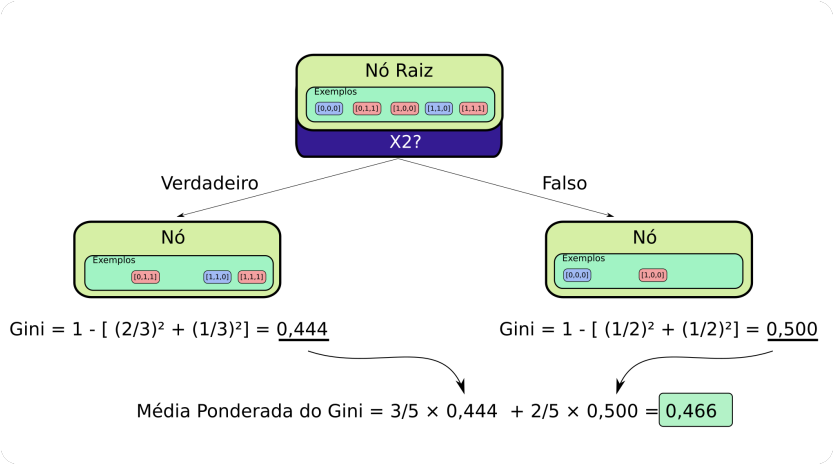
# Exemplo da Escolha do Atributo





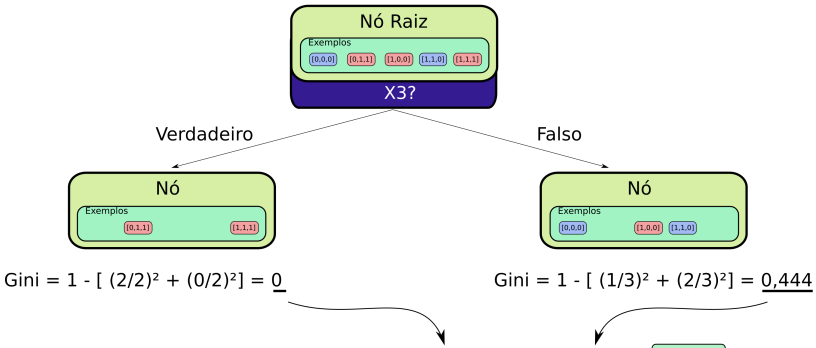
# Exemplo da Escolha do Atributo





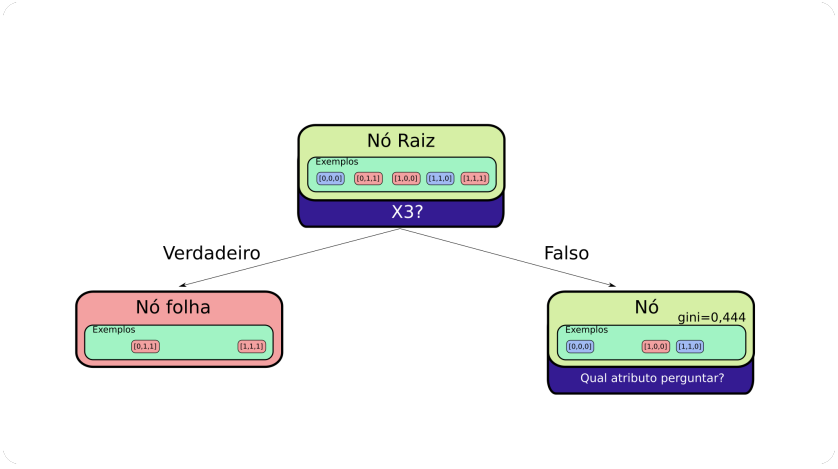
# Exemplo da Escolha do Atributo





# Exemplo da Escolha do Atributo






- continua o processo até finalizar a árvore

# Exemplo da Escolha do Atributo



Comparando com resultados obtidos em Python

- para maior parte dos casos os resultados não são significativamente diferentes

**Nos casos que há diferença:**

- Gini é mais rápido de calcular
- Entropia tende a ter resultados mais balanceados

**O que é?**

Penalizar a criação de hipóteses muito complexas, a fim de aumentar o poder de generalização do algoritmo

- em alguns casos, o padrão subjacente nos dados de entrada pode ser difícil de encontrar (ou pode ser que não exista);
  - isso pode levar a overfitting
- será tentado encontrar uma regra muito específica para o conjunto de dados de treinamento, mas que não realmente encontra o padrão subjacente aos dados.

**Exemplo em que os dados não estão correlacionados com a saída**

| ibovespa subiu no dia anterior? | número de acesso de alunos ao Moodle no dia anterior | choveu no dia seguinte? |
|---|---|---|
| sim | 10 | sim |
| não | 5 | não |
| sim | 6 | sim |
| . | . | . |
| . | . | . |
| . | . | . |
| sim | 8 | não |

- uma árvore de decisão capturaria apenas "ruído"ao tentar prever o padrão nos dados
- mesmo em uma situação com correlação dos dados de entrada com a saída, ocorrerá *overfitting* caso não sejam tomados cuidados;

# Estratégias para Evitar Overfitting

**Desempenho**

Caso o conjunto de dados de treinamento contenha ruído, *pruning* melhora o desempenho (no conjunto teste) e reduz o tamanho da árvore

**Quando**

Pode ser durante ou depois da execução do algoritmo de criação:

- pre-pruning
- post-pruning

# Estratégias para Evitar Overfitting

**Pre-Pruning**

- Evita que a árvore cresça demasiadamente em situações que potencialmente não agregam mais informação;
- Parada precoce ("*early-stopping*");
- corta *de cima para baixo*;
- um atributo que, sozinho, não oferece ganho de informação, é impedido de continuar na avaliação.

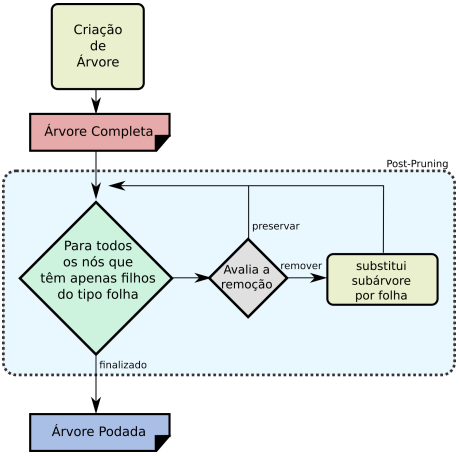
**Pre-Pruning**

Exemplo em Python

**Post-Pruning**

- Após a criação da árvore, realiza um processo de poda;
- corta *de baixo para cima*;
- permite analisar atributos mesmo que sozinhos estes não ofereceçam ganho de informação, permite análise de ganho resultante da combinação de fatores.

- substitui uma ou mais subárvores por um nó folha
- rotula o nó folha com o valor pluralidade dos exemplos deste nó (a classe mais comum dos exemplos que chegam até este nó)



**Formas de avaliar a remoção**

- Custo-complexidade
- Baseada em Estatística
- Reduced Error Pruning

**Custo-Complexidade**

- Atribui um custo para penalizar a complexidade das árvores

$$Custo = erro + \alpha \cdot L$$

onde:

- $\alpha$ é um parâmetro de ajuste que indica o impacto da complexidade na pontuação;
- L é a quantidade de folhas.

A escolha de $\alpha$ se dá por *cross-validation*, de forma que minimize o erro no conjunto validação.

**Custo-Complexidade**

Exemplo em Python

**Reduced Error Pruning (REP)**

- requer conjunto de dados separado para fazer análise de estimativa de erro;
- simples;
- após treinar a árvore no conjunto treinamento, avalia a árvore original e árvores podadas em um conjunto de dados separado (validação);
- para cada nó folha, avalia a remoção do mesmo:
  - continua fazendo o pruning até que o erro no conjunto validação seja maior para a árvore podada do que para a árvore original.

**Baseado em Estatística**

- Avalia se um nó é importante com base em teste de significância;
- Hipótese Nula ($H_0$): o atributo é irrelevante;

Assumindo que não há relevância no atributo:

$$\hat{p}_k = p \cdot \frac{p_k + n_k}{p + n}$$

$$\Delta = \sum_{k=1}^{d} \frac{(p_k - \hat{p}_k)^2}{\hat{p}_k} + \frac{(n_k - \hat{n}_k)^2}{\hat{n}_k}$$

- Na $H_0$: o valor de $\Delta$ é distribuído com uma distribuição $\chi^2$ com v-1 graus de liberdade, onde v = n+p;
- dado um determinado grau de significância, usamos tabela ou função para ver se $\Delta$ encontrado confirma a hipótese nula.

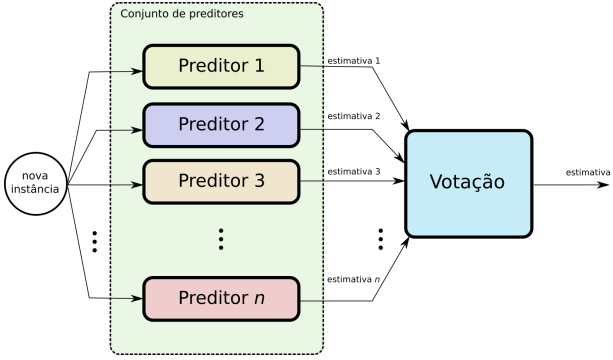
∗ baseado no livro de Stuart J. Russel, Peter Norvig. Artificial Intelligence: A Modern Approach (3rd. ed.). Prentice Hall Press, USA, 2009.

- Várias cabeças pensam melhor que uma;
- *Ensemble Method*: um método que usa vários preditores;
- Combinar a resposta de vários preditores resulta em melhores respostas que preditores individuais;
- Para o caso de:
  - classificador: realiza uma votação com base no resultado dos preditores. A resposta que recebe mais votos vira a saída do conjunto;
  - regressor: a saída do conjunto é a média da resposta dos preditores.

# Métodos Comuns

- Bagging e Pasting
- Boosting
- Stacking
- Random Forest

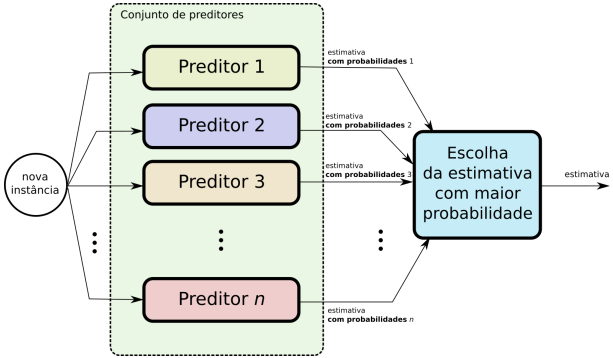
- O caso mais simples: *hard-voting classifier*
- a partir da solução de diversos classificadores, escolher a classe mais votada



Exemplo de *hard-voting classifier* em Python

Exemplo de retornar a probabilidade estimada da classe de resposta em Python

- *soft-voting classifier*;
- caso os preditores retorneme probabilidades de classe de saída, pode-se usar a resposta com maior probabilidade.



Exemplo de *soft-voting classifier* em Python

# Essência

- é preciso haver alguma independência entre os preditores:
  - caso os preditores cometam mesmos erros (erros para as mesmas entradas), sua eficácia em conjunto não é tão satisfatória
- são treinados bom base no mesmo *dataset* de origem, isso leva a haver *correlação*;
- estratégias são adotadas para *minimizar a correlação* entre os preditores;

*Bagging* e *Pasting*

- Suponha que o conjunto treinamento total seja $T = \{(x_1, y_1), (x_2, y_2), ..., (x_m, y_m)\}$
- Preditores são treinados em diferentes subconjuntos do conjunto total de treinamento;
- Bagging (bootstrap aggregation): sorteio com reposição;
- Pasting: sorteio sem reposição;

*Bagging* e *Pasting*

- cada preditor terá maior viés (bias) do que um predito quer usasse todos os dados de treinamento, mas votação o reduzirá o viés do conjunto;
- estas técnicas permitem uso de paralelismo.

*Bagging* e *Pasting*

Exemplo em Python

**Avaliação Out-of-Bag**

Ao usar bagging:

- alguns exemplos vão aparecer múltiplas vezes em um determinado preditor.
- alguns exemplos não vão aparecer em um determinado preditor;
- Podemos utilizar as amostras que não foram sorteadas para fazer a avaliação;
- Probabilidade de selecionar uma amostra específica em 1 sorteio

$$\frac{1}{n}$$

- Probabilidade de não selecionar uma amostra específica em 1 sorteio

$$1 - \frac{1}{n}$$

**Avaliação Out-of-Bag**

- Probabilidade de não selecionar uma amostra específica em $n$ sorteios

$$\left(1 - \frac{1}{n}\right)^n$$

- Para valores grandes de $n$:

$$\left(1 - \frac{1}{n}\right)^n \approx \frac{1}{e} \approx 0,37$$

- Logo:
  - aproximadamente 63 % das amostras serão usadas pelo menos 1 vez;
  - aproximadamente 37 % das amostras não serão usadas (out-of-bag).
- não serão os mesmos 37 % para diferentes preditores.

**Fazendo a Avaliação Out-of-Bag para o Método**

- Para um determinado exemplo *i* do conjunto treinamento, utilizamos para avaliação apenas o preditores que não usaram o exemplo *i* no teste;
- Para o exemplo *i*, a estimativa do método para validação será a média da previsão de todos os preditores que não usaram este exemplo no treinamento;
- O erro no conjunto validação será o erro para o total das previsões realizadas utilizando estas estimativas.

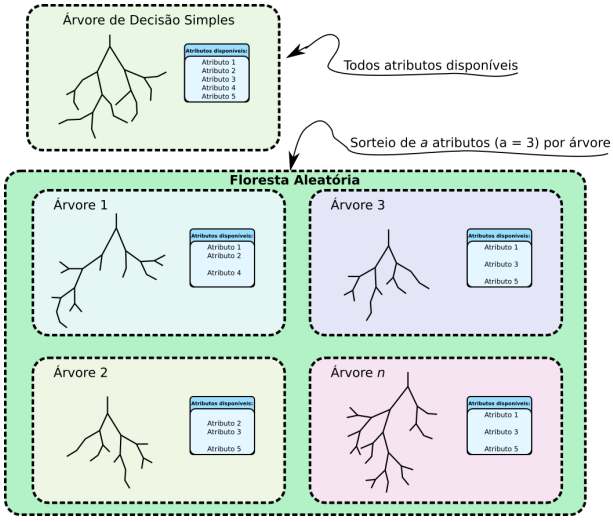
Avaliação Out-of-Bag

Exemplo em Python

**Random Forest**

- usa diversas árvores aleatórias, tal como o bagging;
- há restrição na escolha de atributos nas divisões durante a criação da árvores (feature bagging);
- resulta em maior diversidade entre os preditores.

- O número de atributos utilizado é um hiperparâmetro.



Exemplo em Python

# Extração de Regras (RE) *

- extrair regras de algoritmos black-box
- diversas aplicações: finanças, medicina, ecologia, ...
- h' aproxima h tanto quanto possível
  - exatidão x compeensibilidade

↑ regras ↑ exatidão ↓ compreensibilidade ↑ possibilidade de overfitting

- Forma Geral das Regras:

```
- SE (....) ENTÃO ....
- SE (....) ENTÃO ....
- SE (....) ENTÃO ....
- CASO CONTRÁRIO ( resposta padrão )
```

∗ baseado no artigo de Morteza Mashayekhi e Robin Gras. "Rule Extraction from Decision Trees Ensembles: New Algorithms based on Heuristic Search and Sparse Group Lasso Methods", International Journal of Information Technology Decision Making, Vol. 16, No. 06, pp. 1707-1727 (2017).

# Conclusão

**Tópicos estudados**

- Impureza Gini
- Escolha de Atributo para Divisão
- Estratégias para Evitar Overfitting
  - *Pruning*
  - *Ensemble Methods* (Métodos em Conjunto ou Agrupamento)

# Referências


- Stuart J. Russel, Peter Norvig. Artificial Intelligence: A Modern Approach (3rd. ed.). Prentice Hall Press, USA, 2009.

- Géron, Aurélien. Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems. OReilly Media, 2019.


- Kubat, Miroslav. An Introduction to Machine Learning. Springer International Publishing, second edition, 2017.