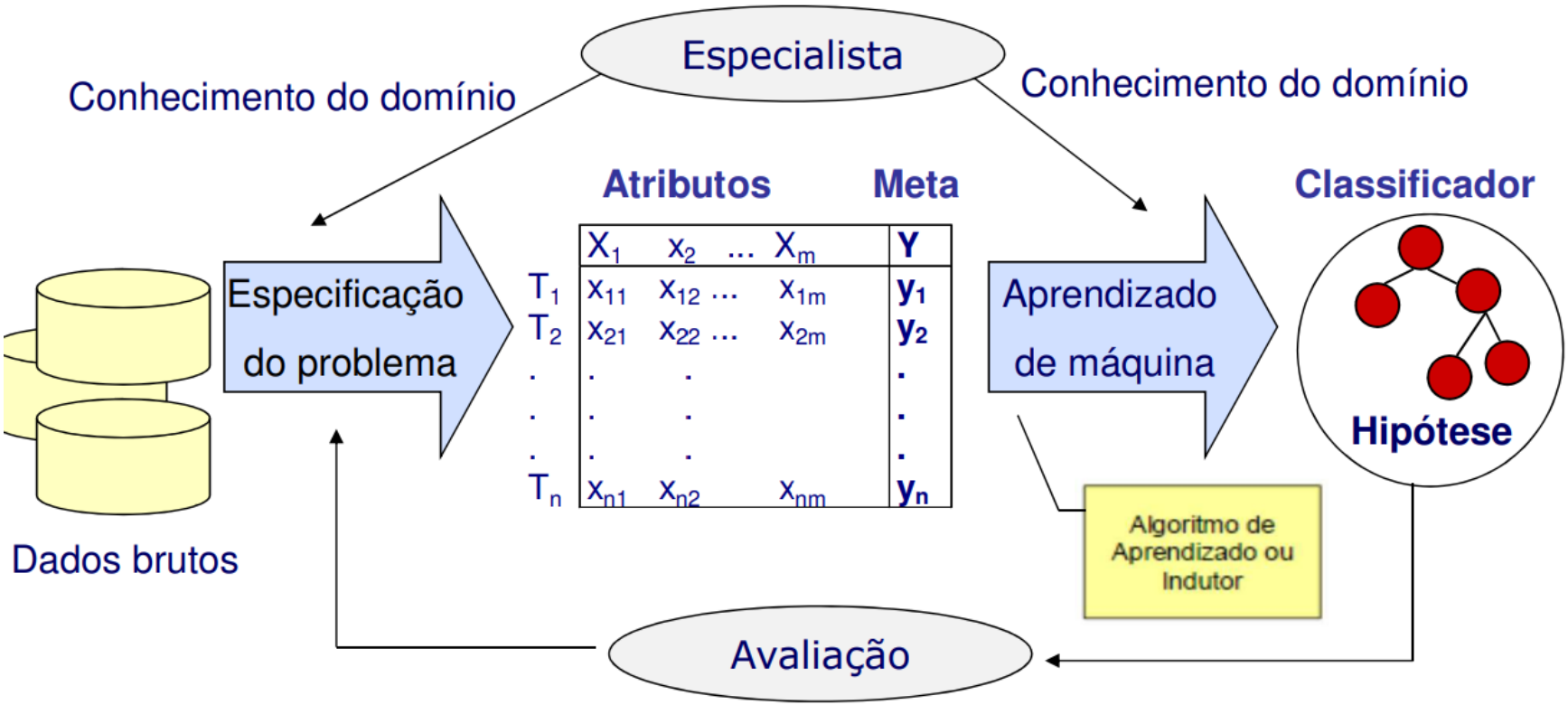
Árvores de Decisão

Cesar de Oliveira F. Silva

Setembro / 2026

Roteiro

Processo de classificação



O que é uma árvore de decisão?

- Estrutura em forma de **fluxograma**:
  - **Nó interno**: teste sobre um atributo (pergunta).
  - **Ramo**: resultado do teste (sim/não, $\leq$ / $>$, categoria etc.).
  - **Folha**: classe ou valor de saída (decisão).
- Podemos usar tanto para **classificação** quanto para **regressão**.
- Intuição: é como jogar “20 perguntas” até chegar a uma resposta.

Exemplo intuitivo

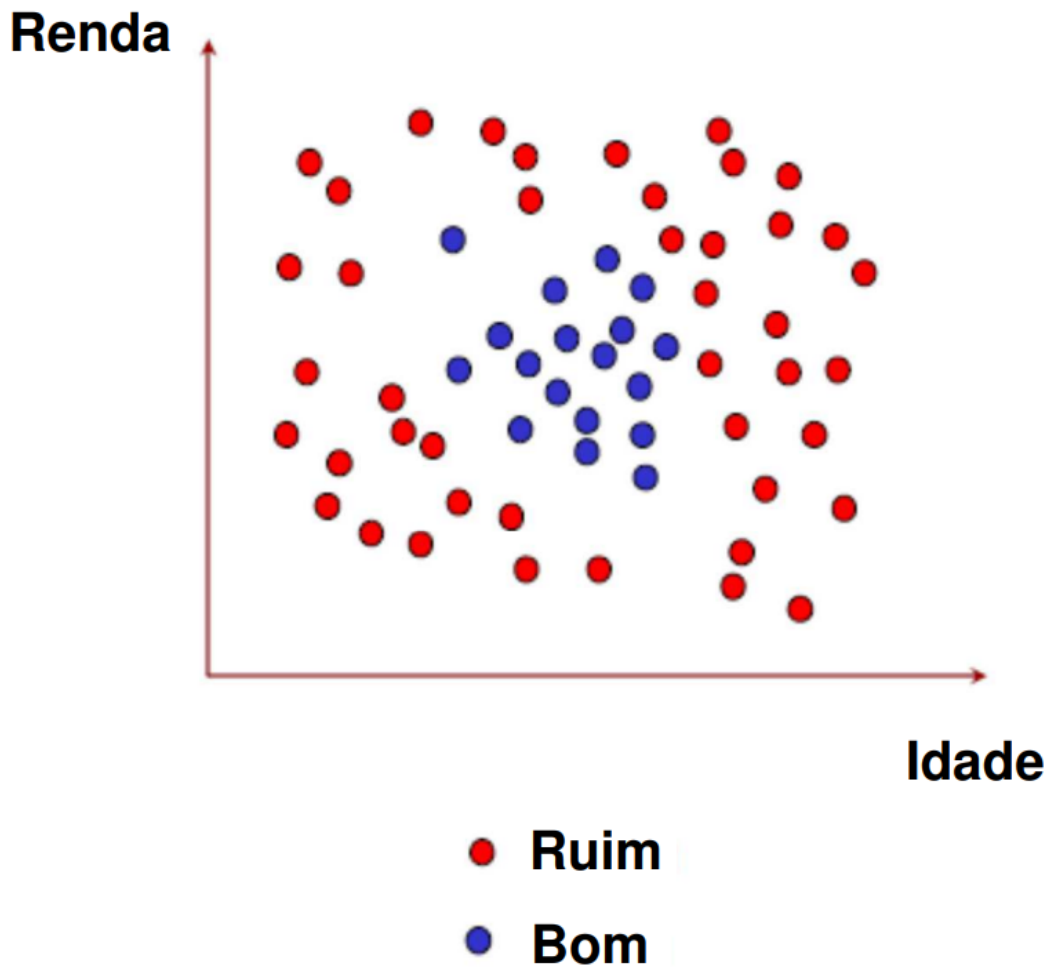
Queremos classificar se um dia é **bom para ir ao parque**.

- Nó raiz: **Chove?**
  - Se **sim**: decisão $\Rightarrow$ `Não ir`.
  - Se **não**: próximo teste.
- Segundo nó: **Temperatura** $>$ **25ºC?**
  - Se sim: `Ir à tarde`.
  - Se não: `Ir de manhã`.

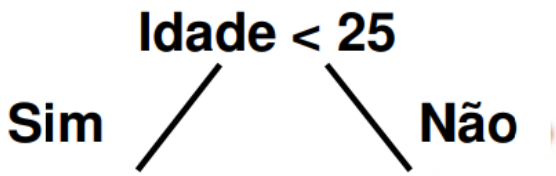
Ideia

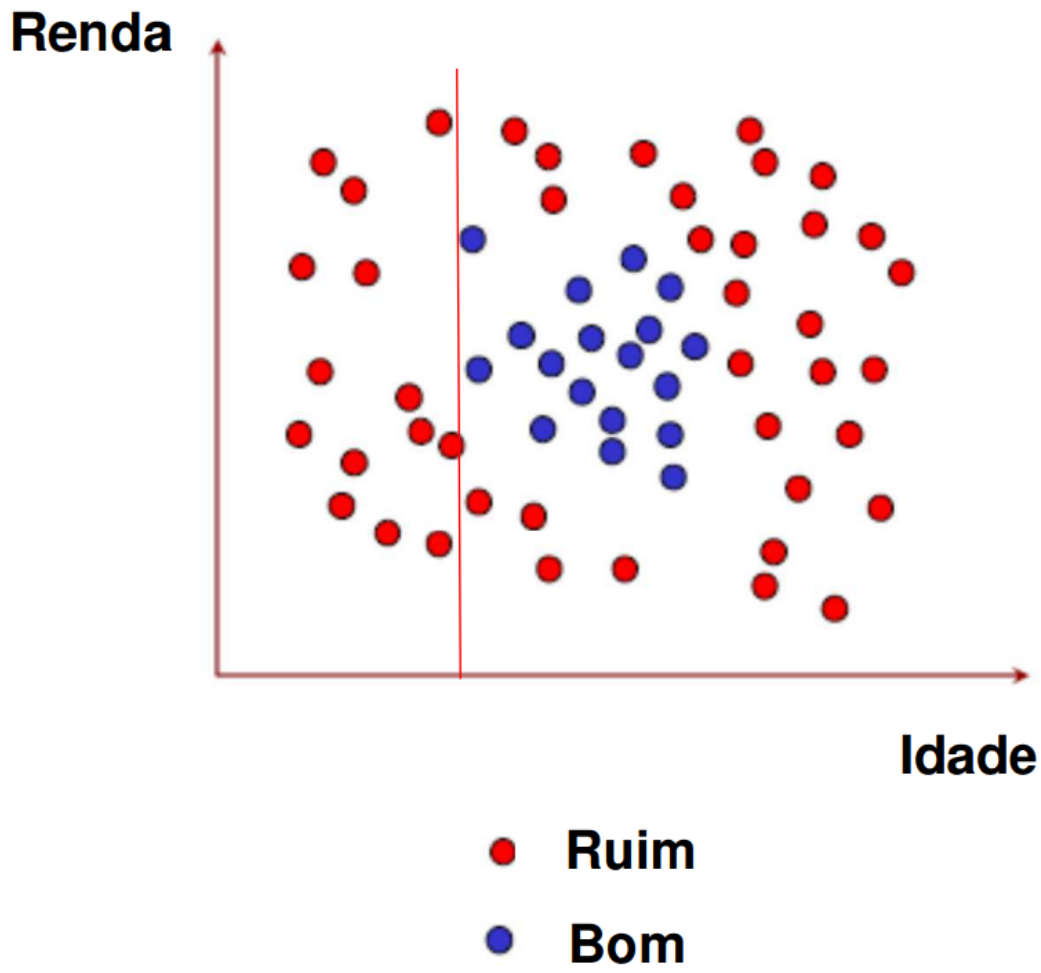
Cada pergunta tenta separar os casos em grupos **mais homogêneos** em relação à decisão final.

Construindo uma árvore de decisão

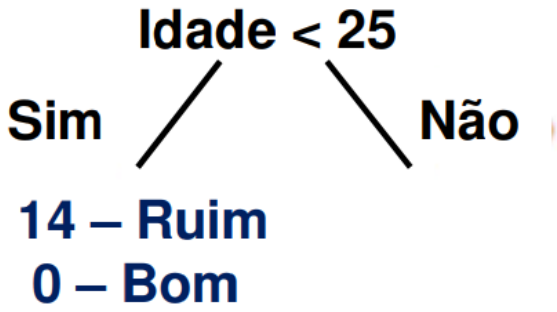


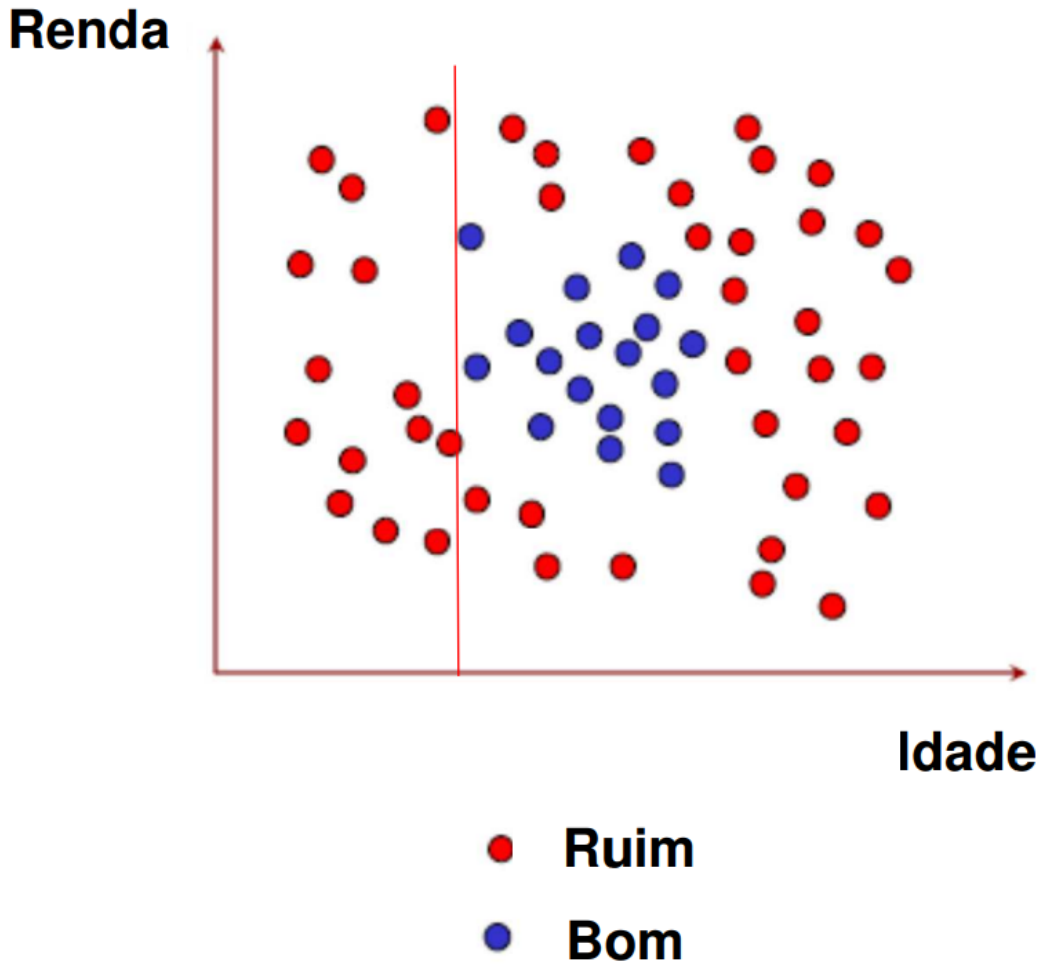
Construindo uma árvore de decisão



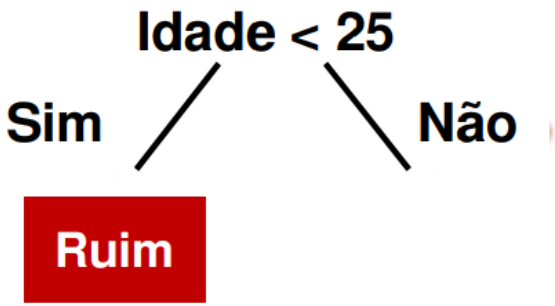


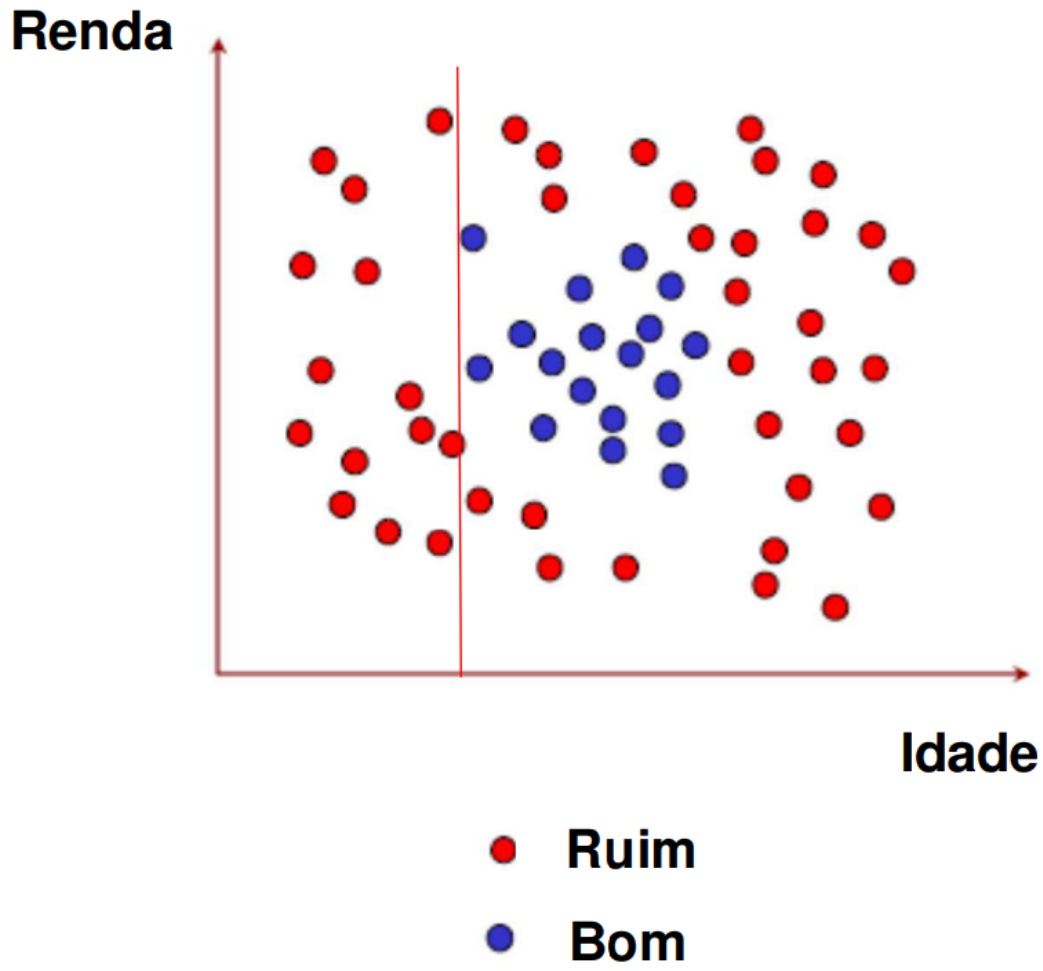
Construindo uma árvore de decisão



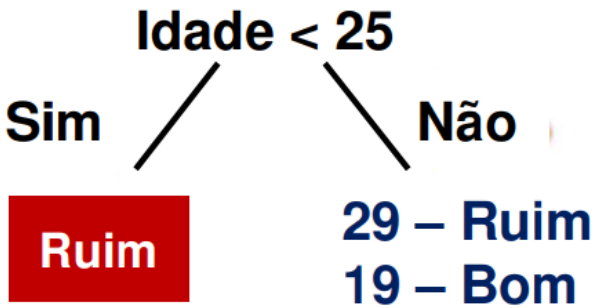


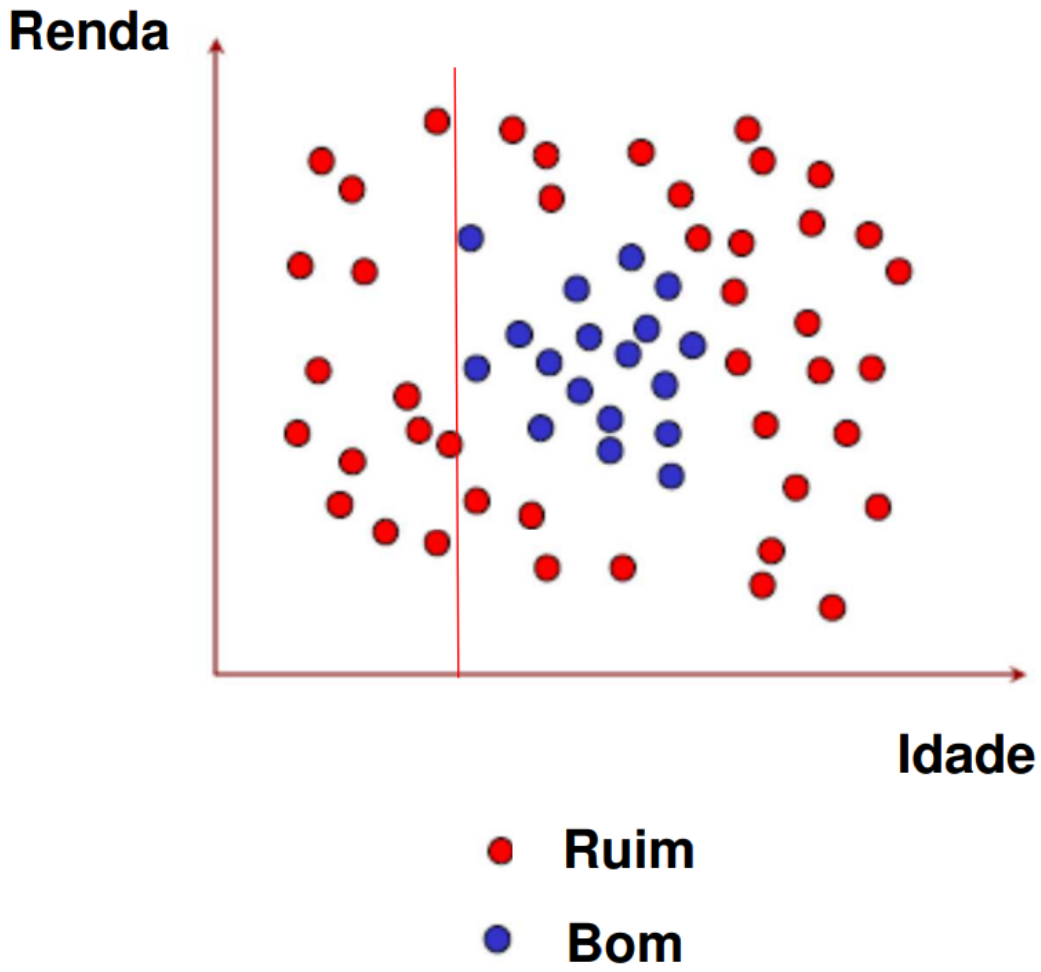
Construindo uma árvore de decisão



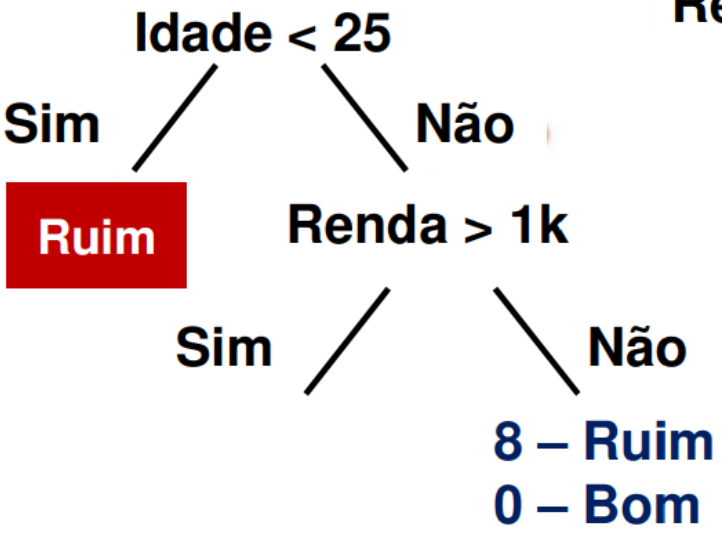


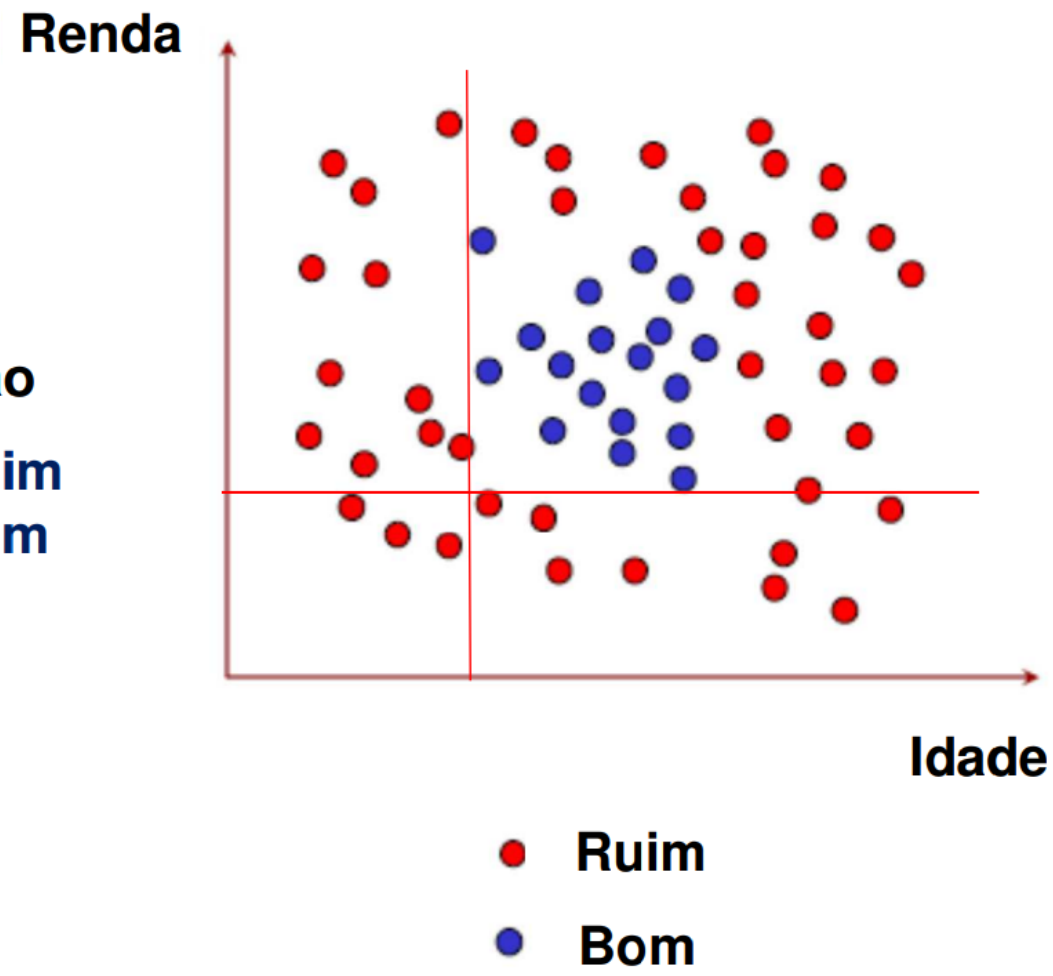
Construindo uma árvore de decisão



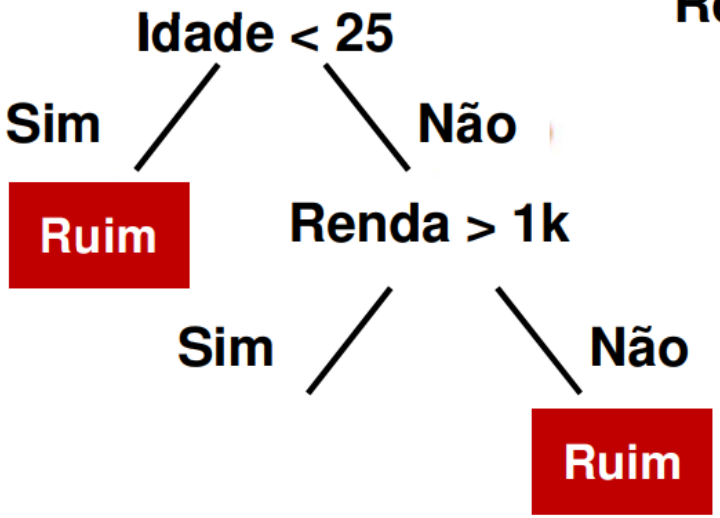


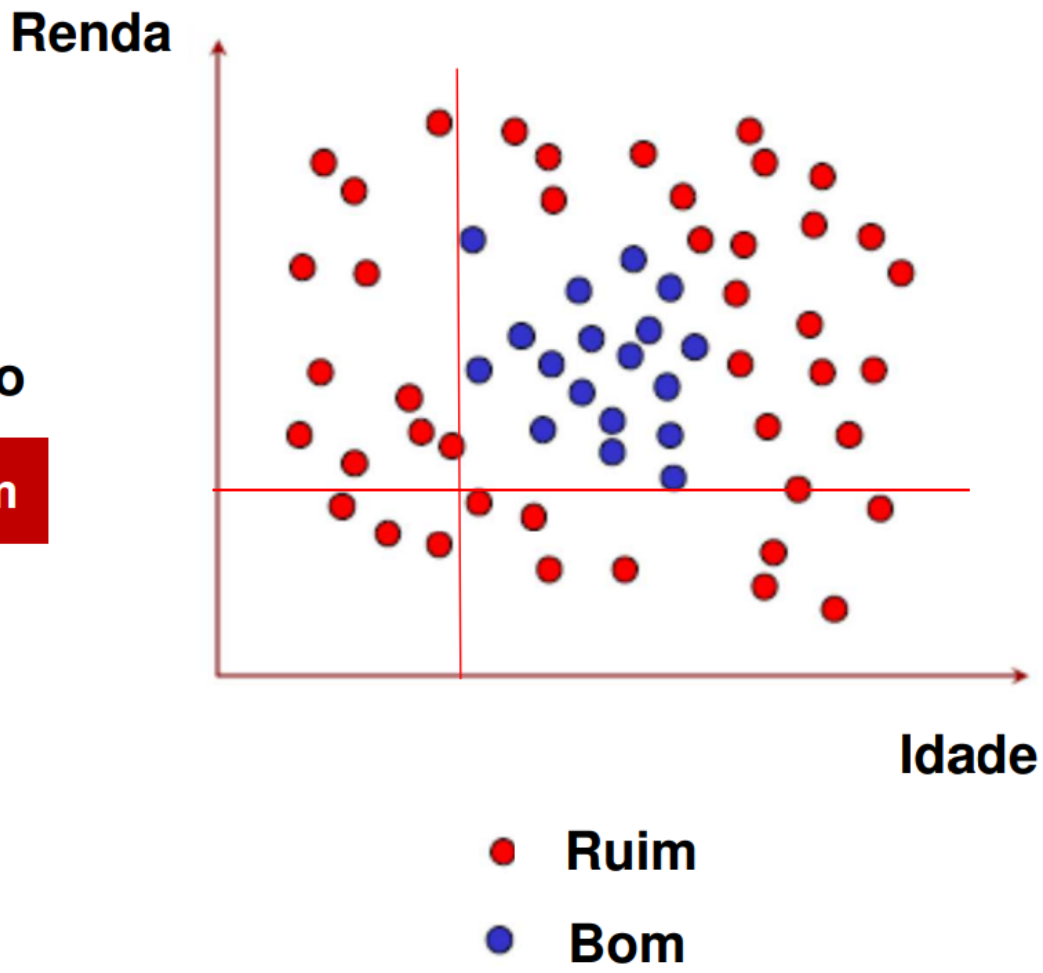
Construindo uma árvore de decisão



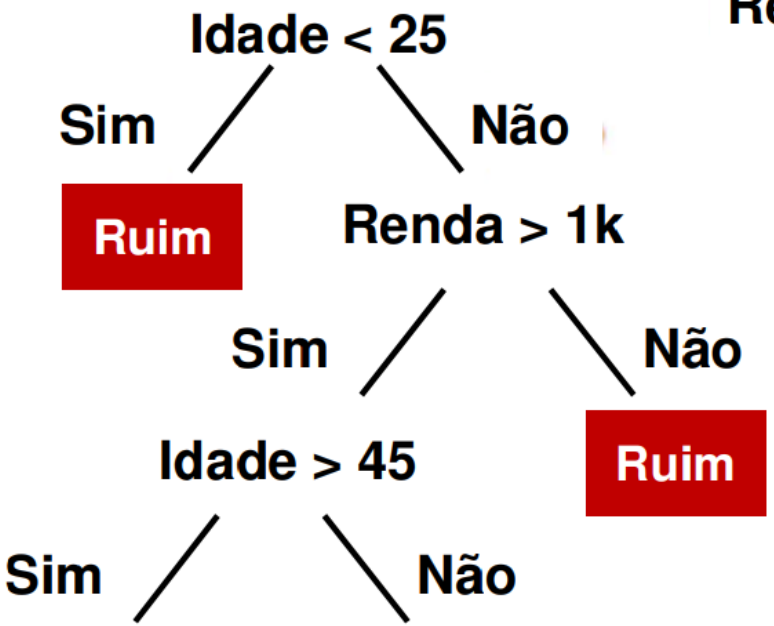


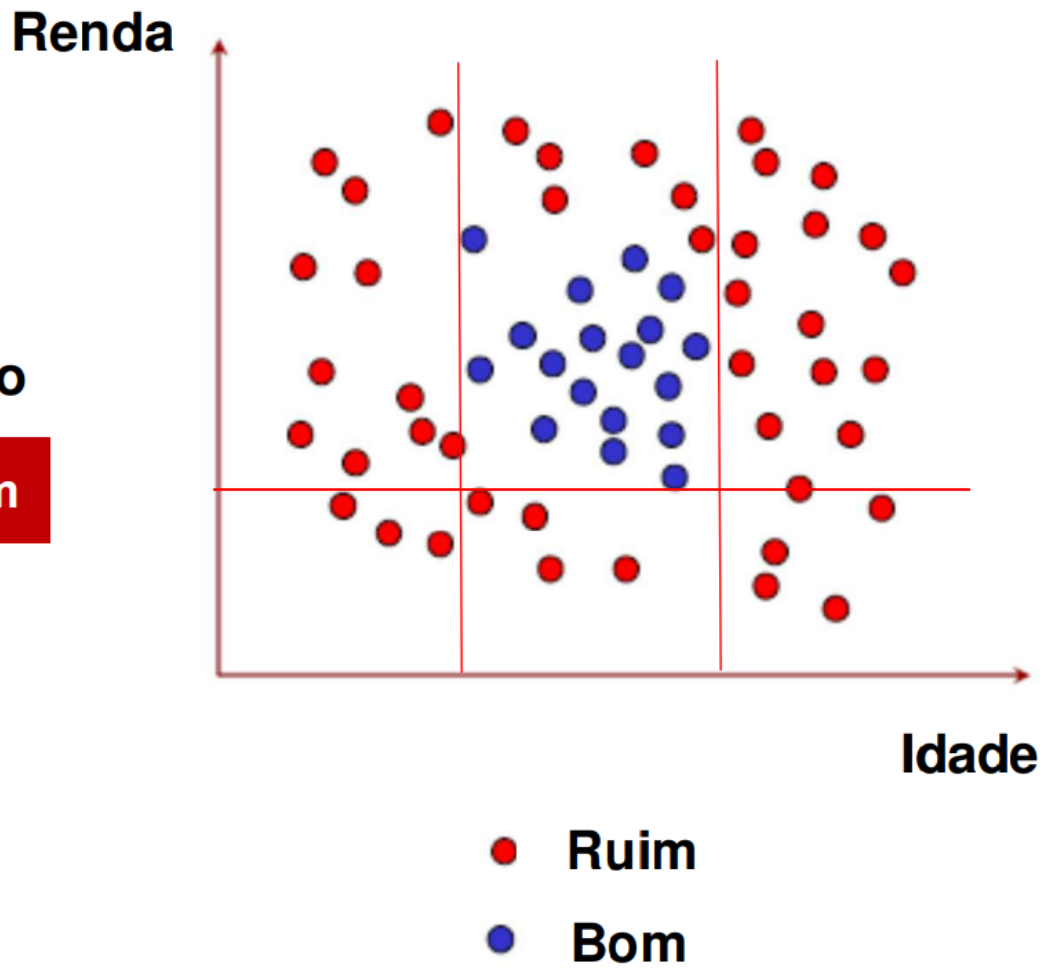
Construindo uma árvore de decisão



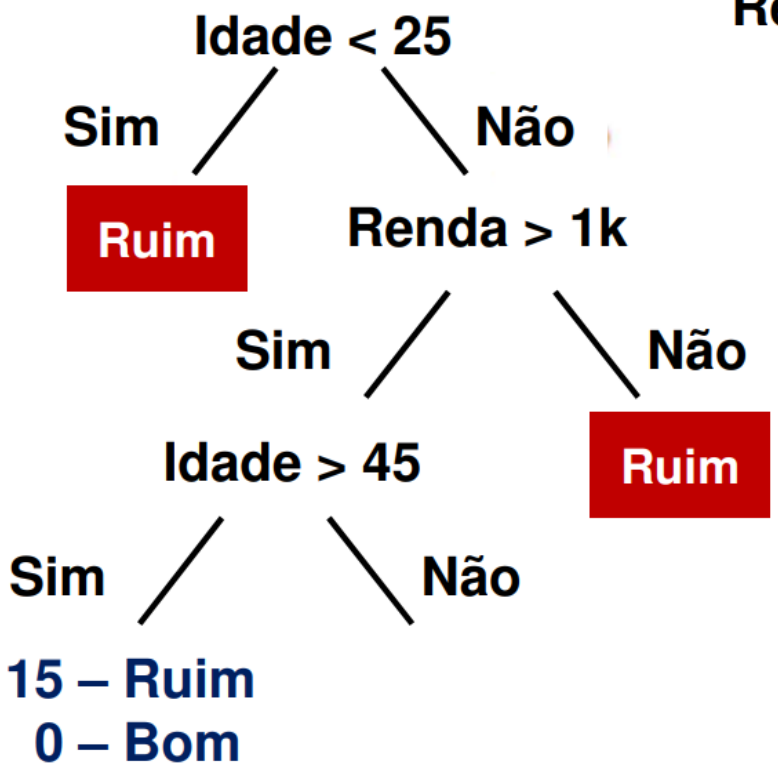


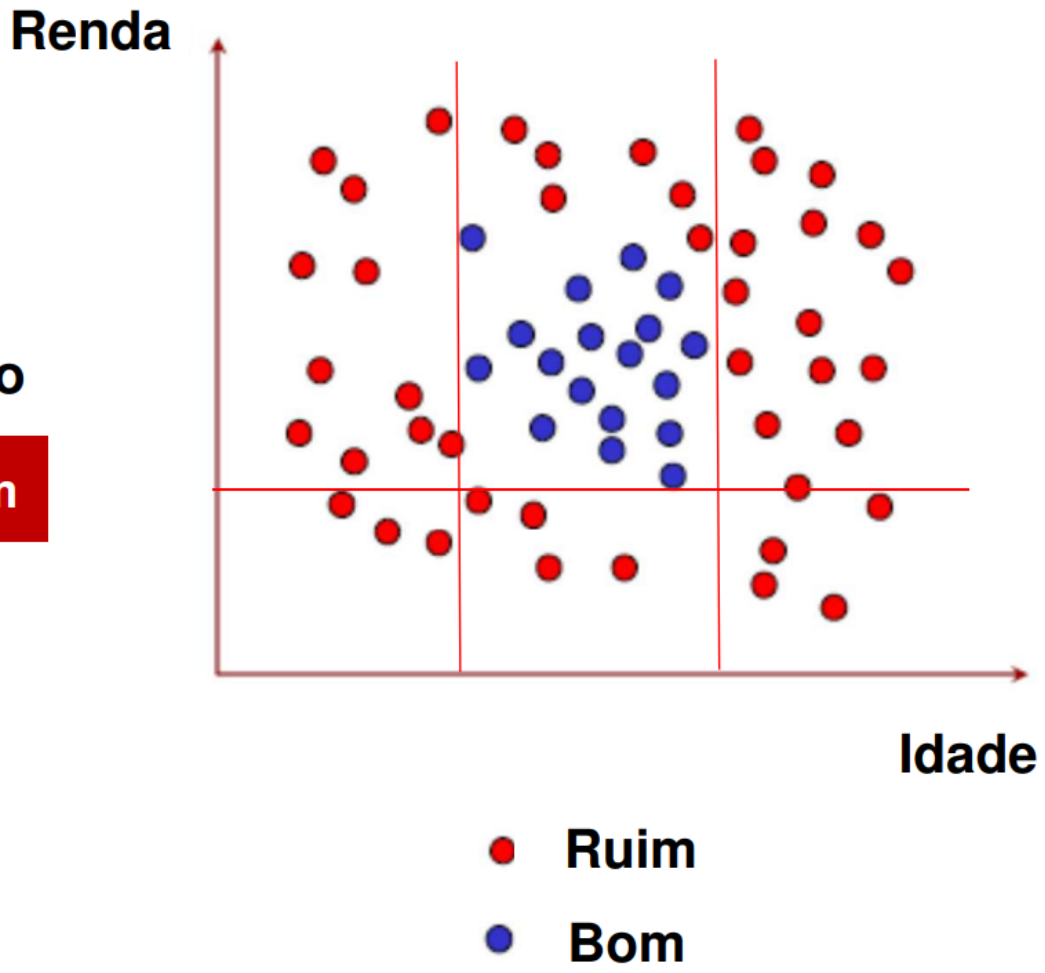
Construindo uma árvore de decisão



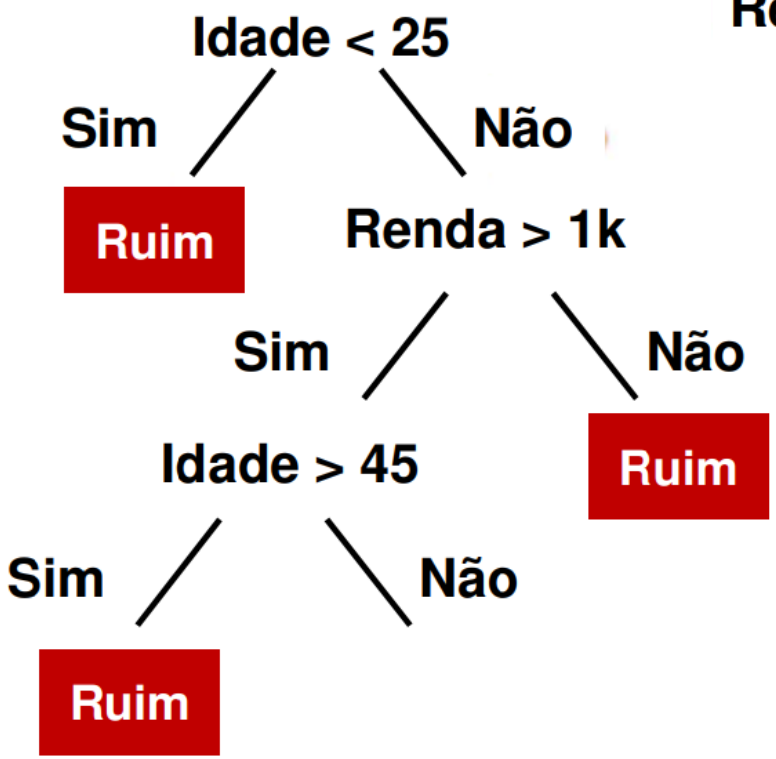


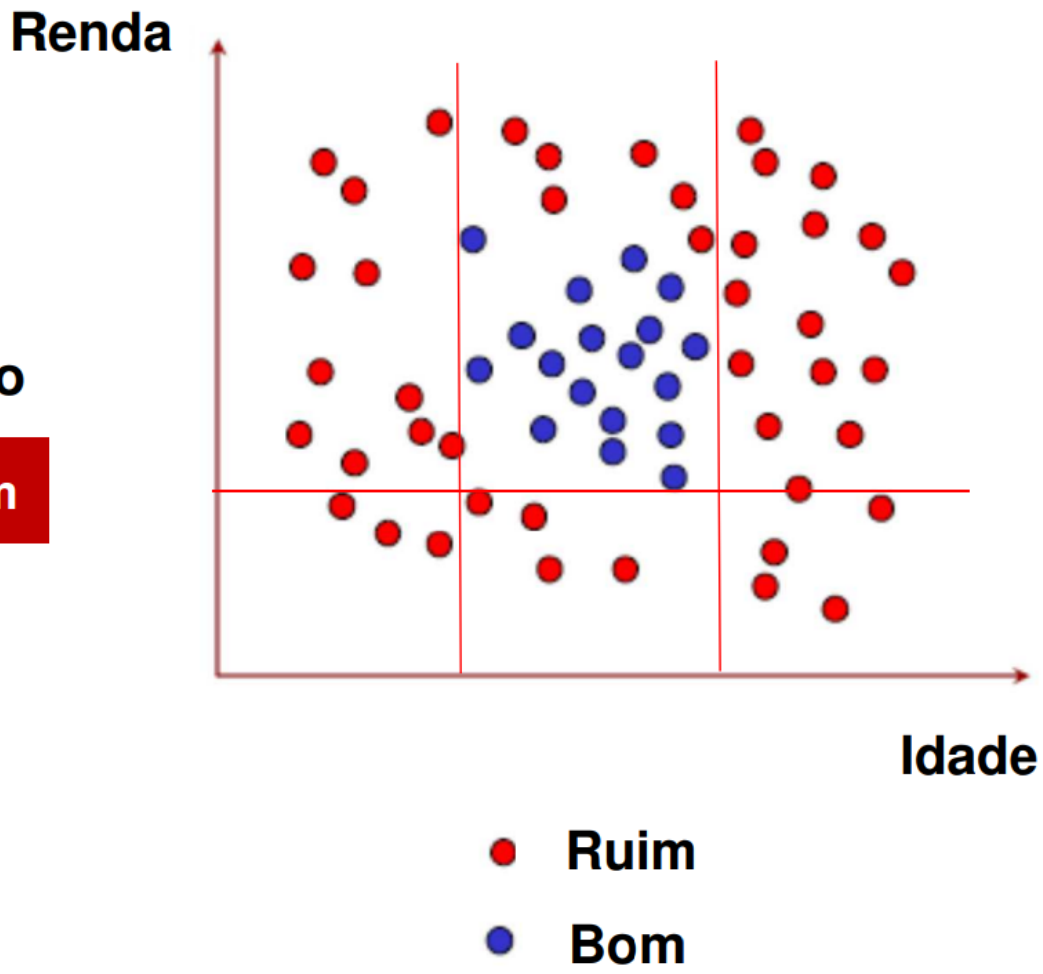
Construindo uma árvore de decisão



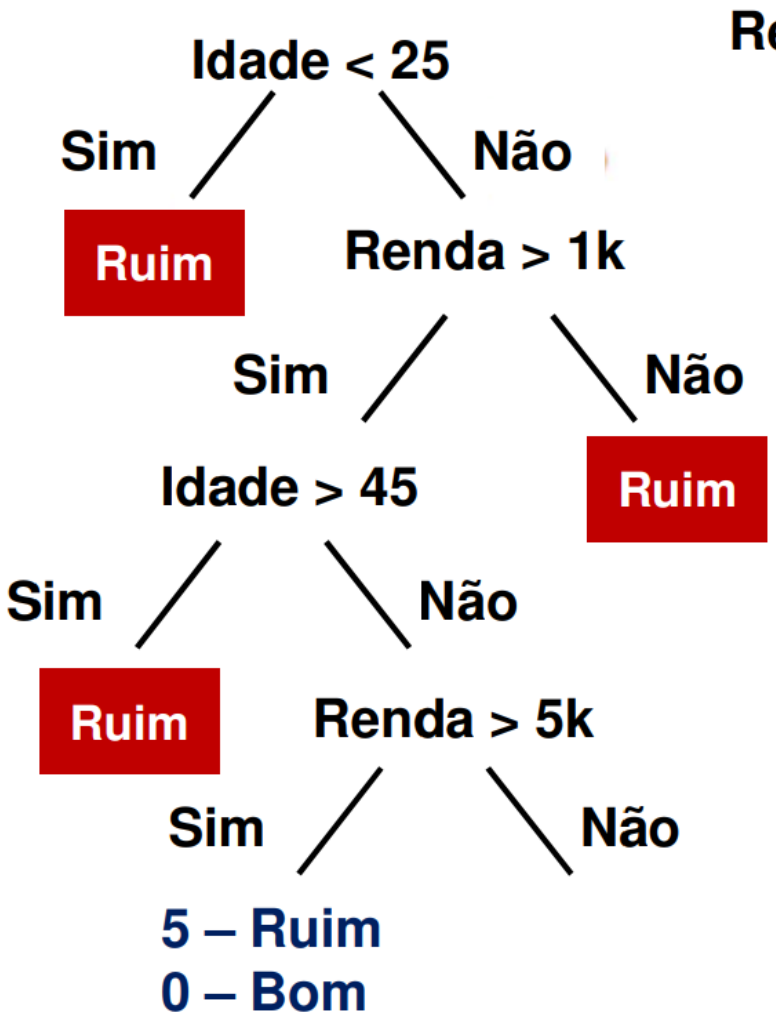


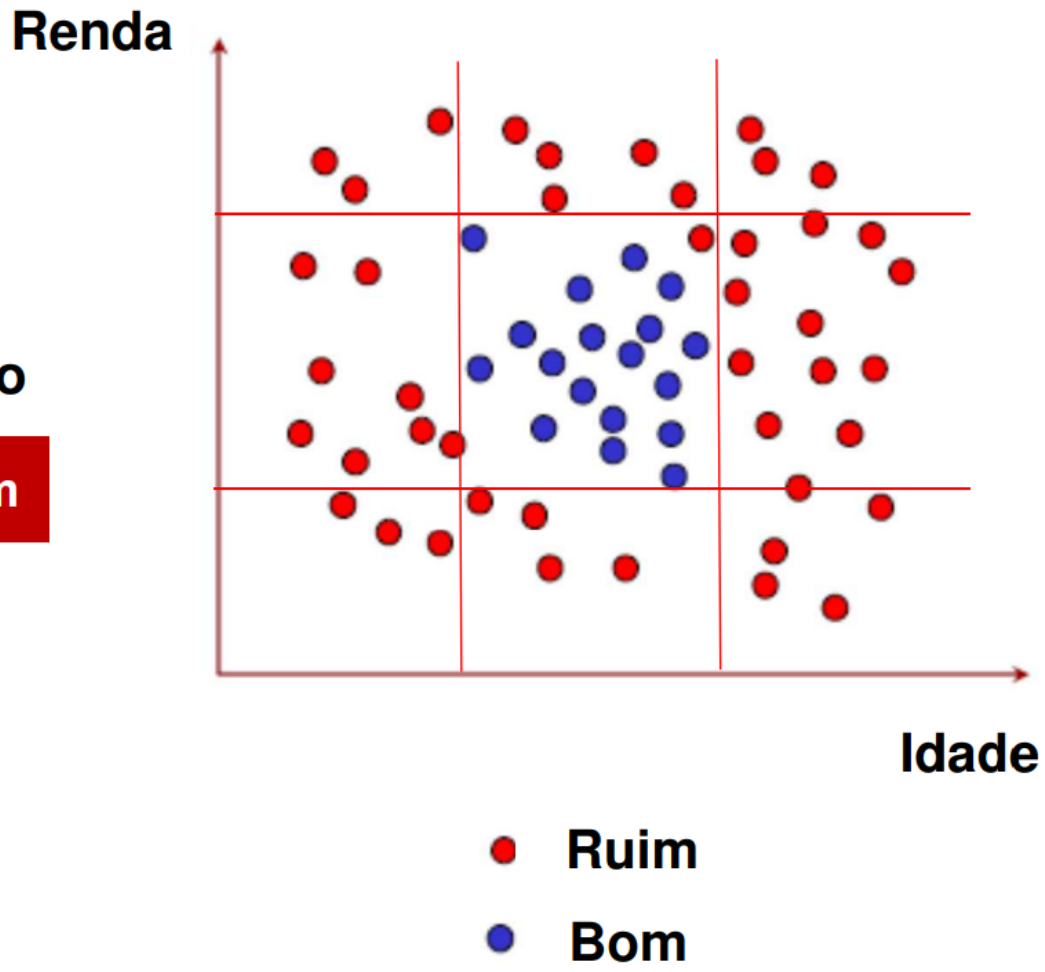
Construindo uma árvore de decisão



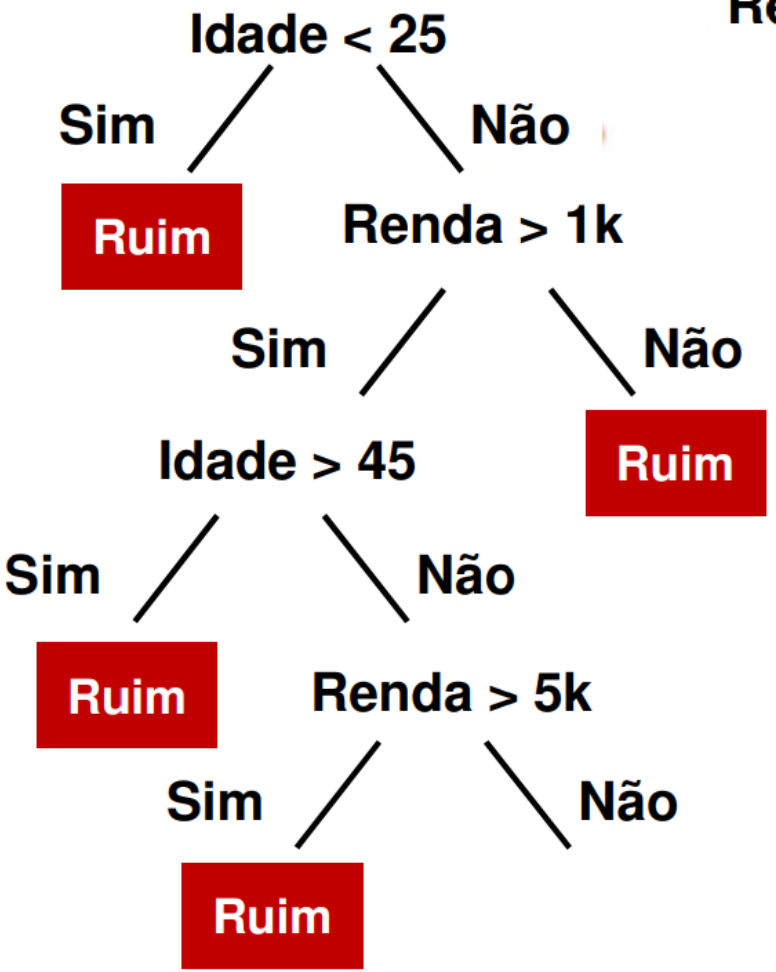


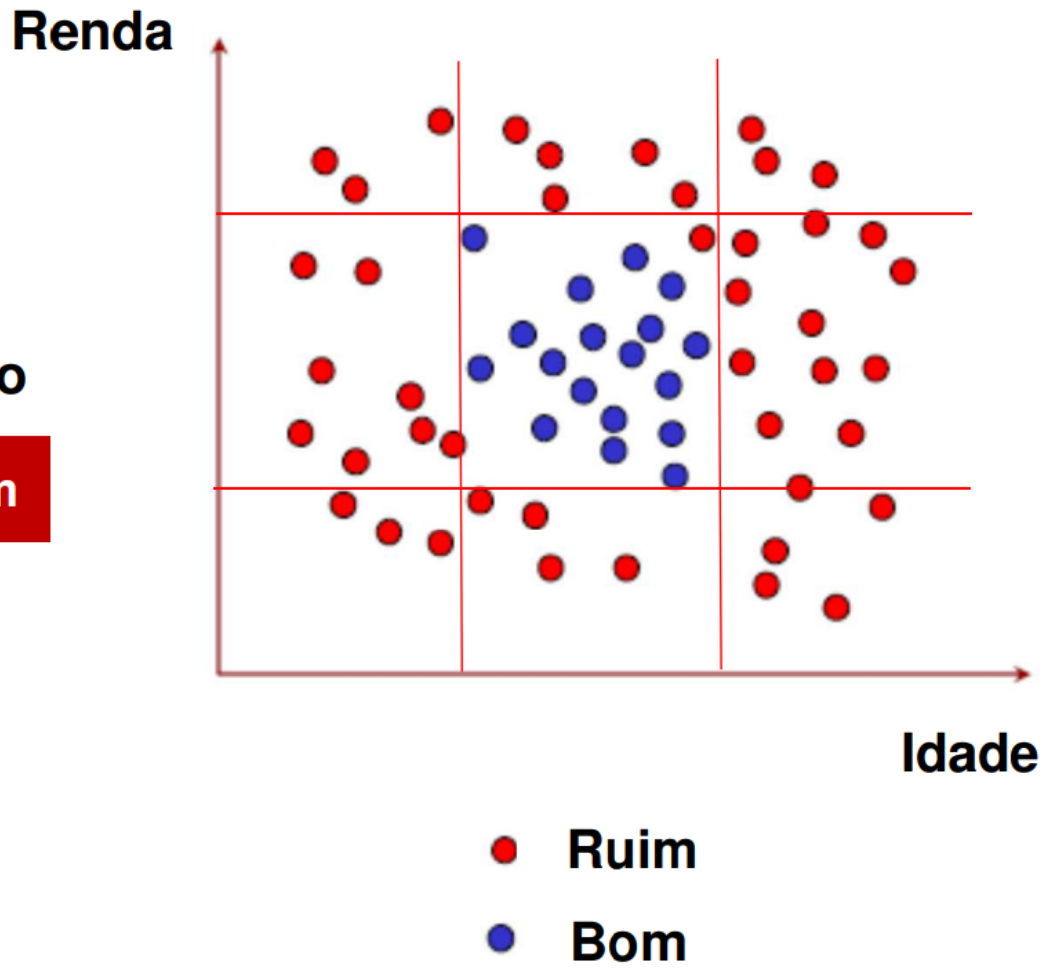
Construindo uma árvore de decisão



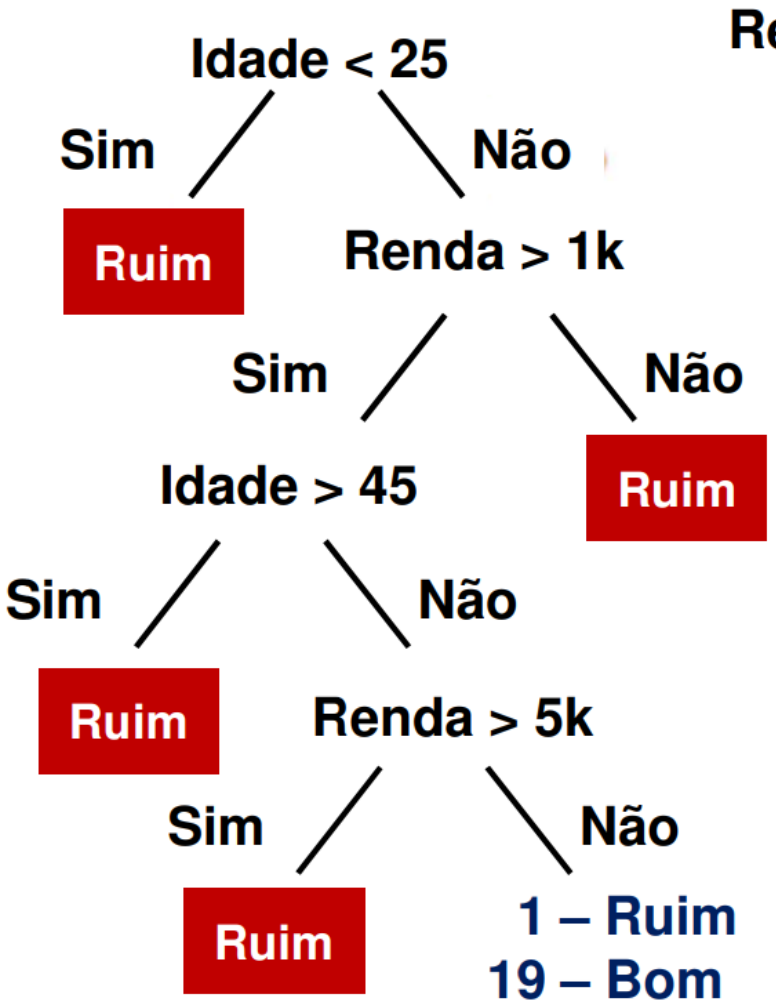


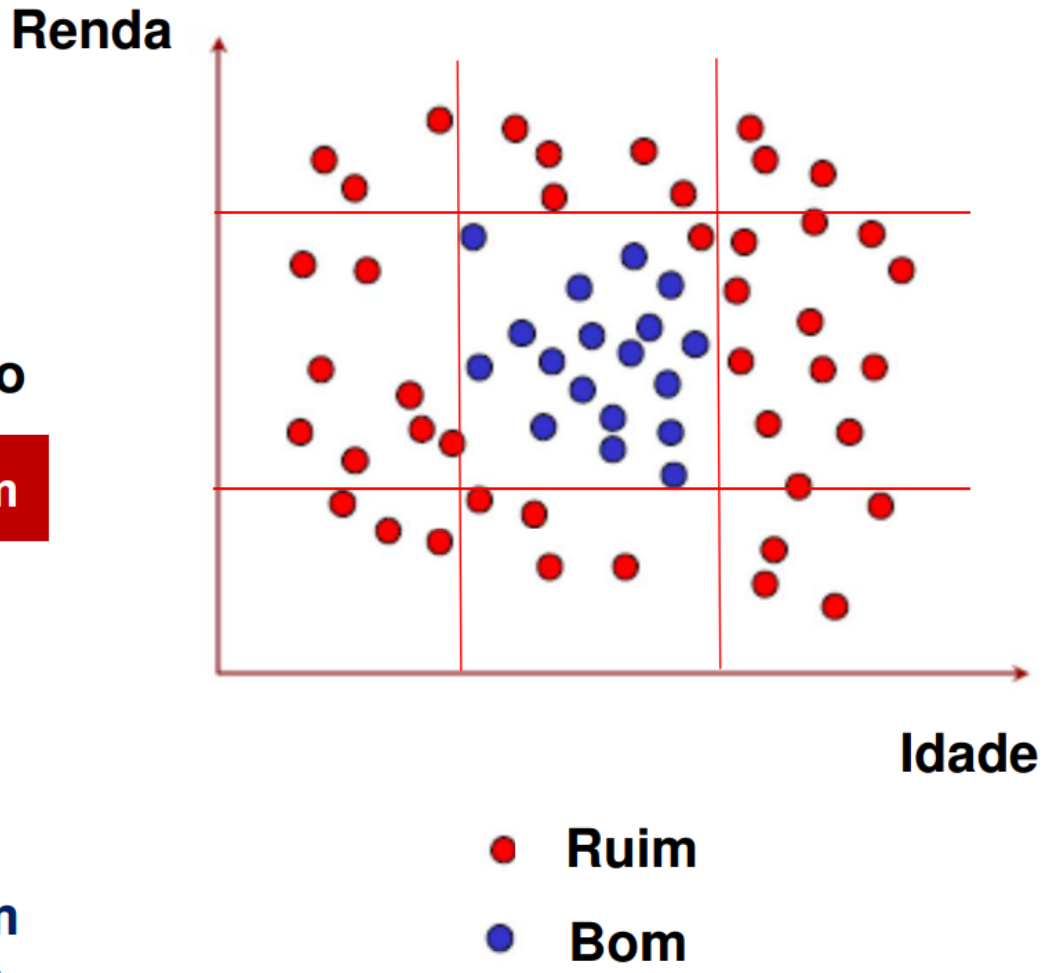
Construindo uma árvore de decisão



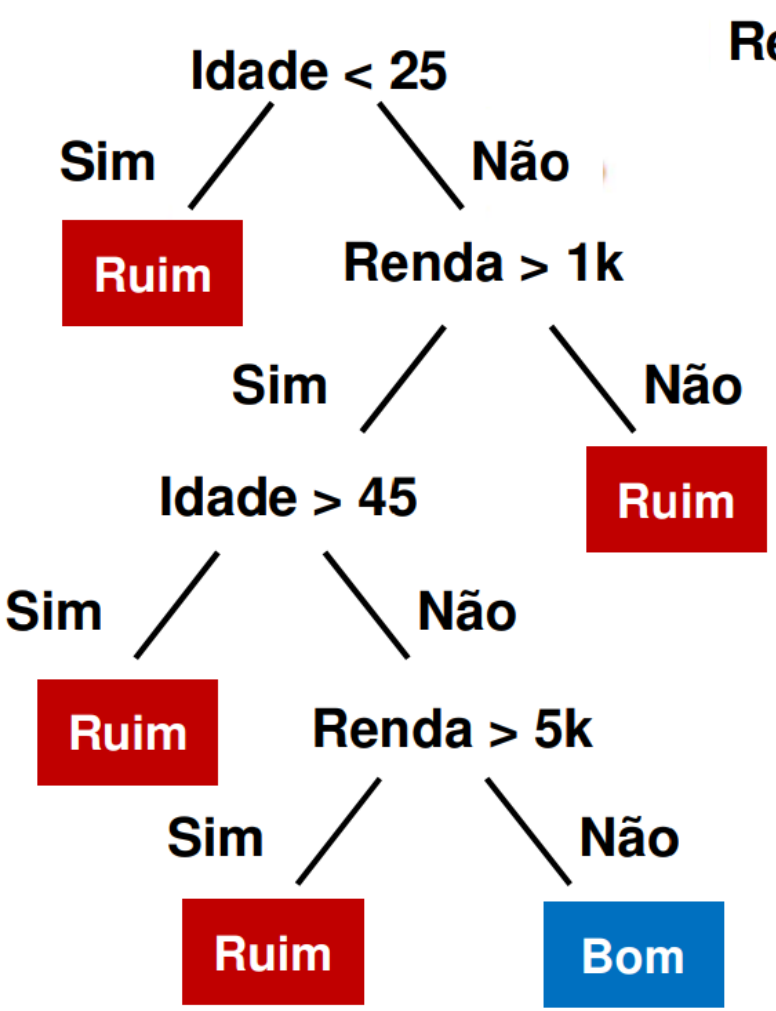


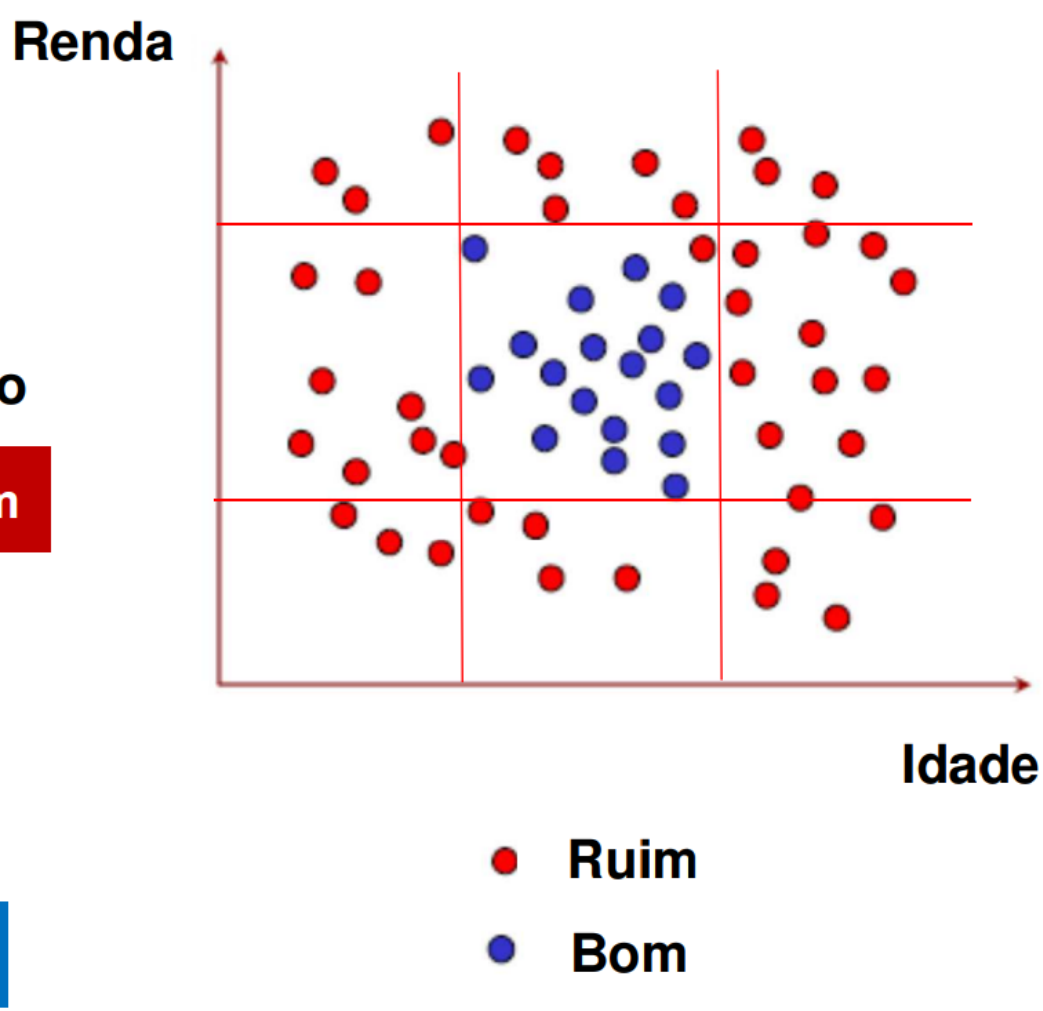
Construindo uma árvore de decisão





Construindo uma árvore de decisão





Fases principais

- **Fase 1 – Construção (growing)**
  - Particionamento recursivo dos dados.
  - Busca de um **bom atributo** para dividir o conjunto em cada nó.
- **Fase 2 – Poda (pruning)**
  - Remover ramos que capturam apenas **ruído** ou **outliers**.
  - Objetivo: melhorar a **generalização** em dados novos.
- **Uso da árvore**
  - Para classificar uma nova amostra, vamos percorrendo a árvore e respondendo às perguntas até chegar a uma folha.

Algoritmo guloso top-down

Ideia geral (ID3, C4.5, CART):

1. Começamos com todas as amostras no nó raiz.
2. Se todas pertencem à mesma classe $\Rightarrow$ cria folha.
3. Caso contrário:
   - Para cada atributo possível, calcula-se um **critério de qualidade de divisão**.
   - Escolhe-se o atributo com melhor valor do critério.
   - Divide-se o conjunto de dados em subconjuntos (filhos).
   - Repete-se o processo recursivamente em cada filho.
4. Paramos quando:
   - não há mais atributos para dividir; ou
   - o nó fica “pequeno demais”; ou
   - o ganho de qualidade da divisão é muito pequeno.

Critérios de divisão: entropia

Vamos medir quão "misturado" está um conjunto de exemplos.

Entropia de um conjunto S

$$H(S) = -\sum_{c=1}^{C} p_c \log_2 p_c$$

- C: número de classes;
- p_c: proporção de exemplos de S na classe c.

- Se todas as amostras são da mesma classe $\Rightarrow H(S) = 0$ (nó puro).
- Se as classes aparecem em proporções iguais $\Rightarrow H(S)$ é máximo (nó muito misturado).
- Intuição: entropia mede a **impureza** ou o grau de desorganização das classes em S.

Ganho de informação

Para um atributo A com valores (ou faixas) $v \in \mathcal{V}$:

Ganho de informação

$$\text{Gain}(S, A) = H(S) \; - \; \sum_{v \in \mathcal{V}} \frac{|S_v|}{|S|} H(S_v)$$

- S_v: subconjunto de exemplos em que o atributo A assume o valor (ou intervalo) v.

- Escolhemos como teste o atributo com **maior ganho de informação**.
- Intuição: buscamos a pergunta que mais **reduz a desorganização** das classes depois da divisão.
- Problema: o ganho de informação tende a favorecer atributos com muitos valores distintos $\Rightarrow$ C4.5 usa a **taxa de ganho** (gain ratio) para corrigir esse viés.

Atributos numéricos

Como dividir usando um atributo contínuo, por exemplo, `idade`?

- Em C4.5 / CART, busca-se um **ponto de corte** t tal que:

$$\text{teste} = \texttt{idade} \leq t \quad \text{vs.} \quad \texttt{idade} > t$$

- O algoritmo avalia possíveis valores de t (por exemplo, médias entre idades ordenadas), calcula o ganho de informação (ou índice Gini) para cada um, e escolhe o melhor.
- Resultado: cada divisão numérica gera apenas **dois ramos** (binária).

Overfitting e poda

- Se deixarmos a árvore crescer até explicar todos os exemplos de treino, ela pode aprender **ruído** e **exceções**.
- Consequência: ótimo desempenho no treino, desempenho ruim em novos dados $\Rightarrow$ **overfitting**.

Poda (pruning)

- **Pré-poda**: parar o crescimento antes (ex.: tamanho mínimo de nó, ganho mínimo).
- **Pós-poda**: primeiro cresce a árvore ao máximo, depois remove ramos que não melhoram a performance em validação.

- A poda busca o equilíbrio entre **completuide** (explicar os casos) e **simplicidade** (boa generalização).

Completuide e consistência

- **Completa**: a hipótese (árvore) consegue classificar **todos** os exemplos.
- **Consistente**: a árvore classifica **corretamente** todos os exemplos de treino.

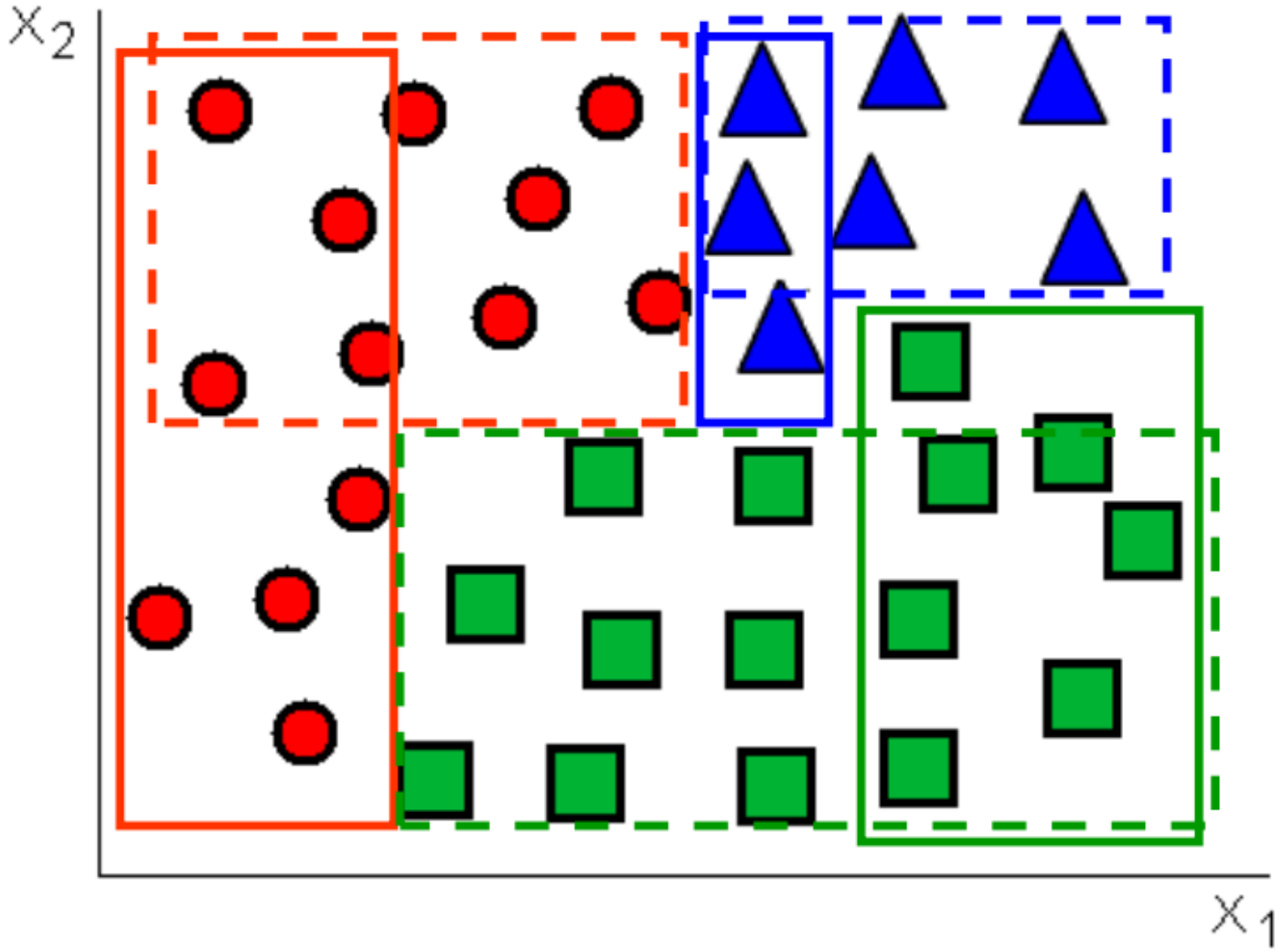
Possibilidades:

- Completa e consistente (caso ideal, mas pode ser overfitting).
- Incompleta e consistente (não cobre todos os casos).
- Completa e inconsistente (cobre tudo, mas com erros).
- Incompleta e inconsistente.

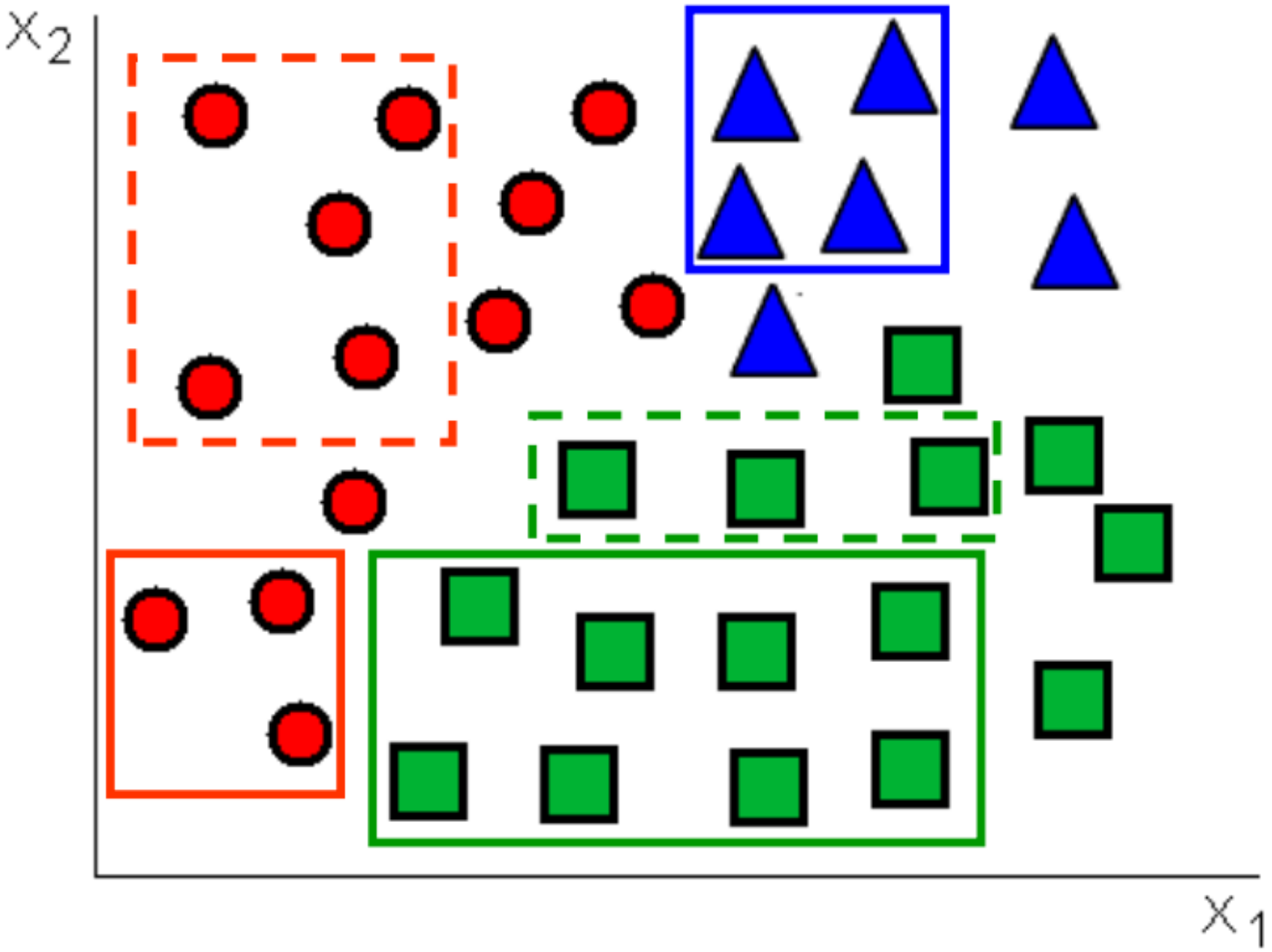
Na prática

Queremos árvores **simples** e com bom desempenho em dados não vistos, mesmo que não expliquem perfeitamente todo o conjunto de treino.

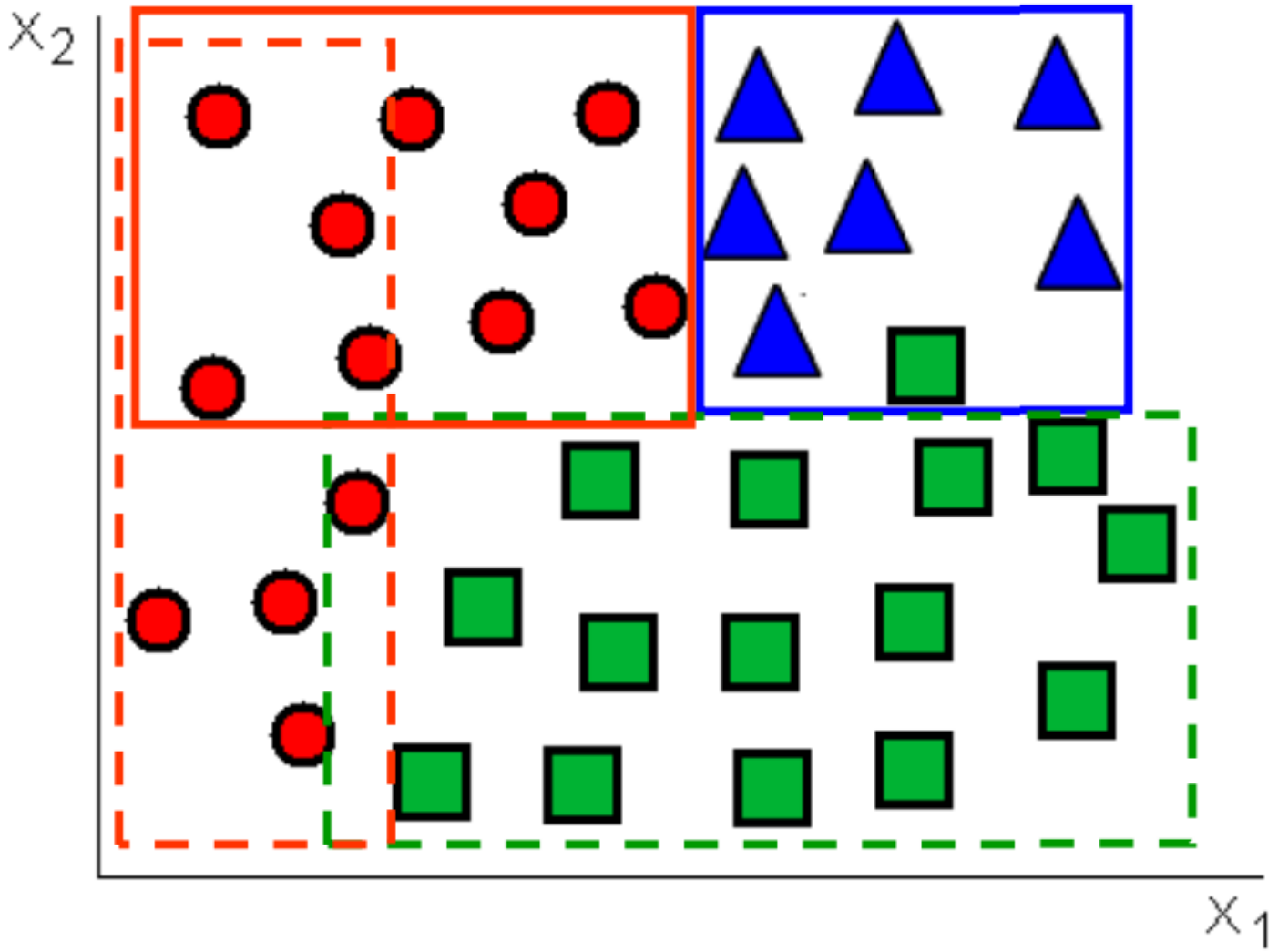
Completo e Consistente



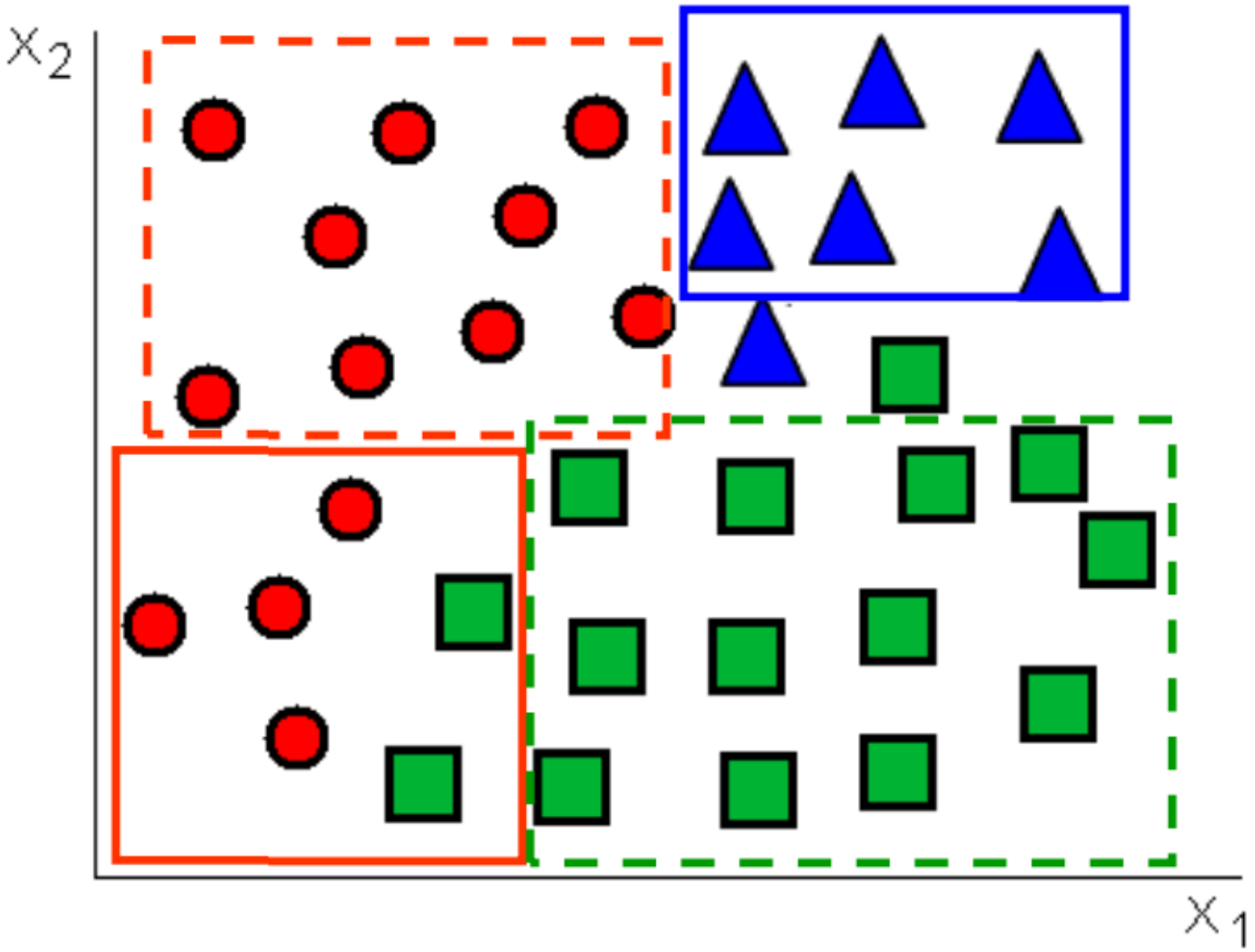
Incompleto e Consistente



Completo e Inconsistente



Incompleto e Inconsistente



Algoritmos clássicos

- **ID3** (Iterative Dichotomiser 3, Quinlan 1986)
  - Usa **ganho de informação** como critério de divisão.
  - Trabalha com atributos **categóricos**; atributos numéricos precisam ser discretizados antes.
- **C4.5** (J48 no Weka, Quinlan 1993)
  - Sucessor do ID3.
  - Suporta atributos **numéricos e categóricos**.
  - Usa **gain ratio** para reduzir o viés de atributos com muitos valores.
  - Possui regras de **poda pós-poda**.
- **CART** (Classification And Regression Trees, Breiman et al. 1984)
  - Gera árvores **binárias** (sempre 2 ramos por nó).
  - Critério típico: **índice Gini** para classificação.
  - Pode produzir **árvores de regressão** (saída numérica).

Heurísticas vs busca exaustiva

- Espaço de todas as árvores possíveis para um conjunto de atributos é gigantesco (**problema NP-completo**).
- Buscar exaustivamente a melhor árvore:
  - Custo cresce de forma exponencial com o número de atributos e exemplos.
  - Impraticável na prática.
- Solução: usar **heurísticas gulosas**:
  - Em cada passo, escolhe-se o melhor atributo localmente (maior ganho, menor Gini etc.).
  - Não há garantia de achar a árvore globalmente ótima.
  - Mas, na prática, obtém-se boas árvores com custo computacional aceitável.

Vantagens

- **Interpretação**: árvores pequenas são fáceis de entender e explicar.
- **Velocidade**:
  - Treinamento relativamente rápido em muitos casos.
  - Classificação de novos exemplos é muito rápida.
- **Pré-processamento simples**:
  - Pouca necessidade de normalizar atributos.
  - Lida bem com mistura de atributos numéricos e categóricos (C4.5, CART).
- **Capacidade não-linear**: pode aproximar fronteiras de decisão complexas.

Desvantagens

- Propensas a **overfitting** se não houver poda ou regularização.
- Pequenas variações nos dados podem gerar árvores bem diferentes (**instabilidade**).
- Árvores muito profundas tornam-se difíceis de interpretar.
- Para alta performance em problemas complexos, muitas vezes preferimos **ensembles** (Random Forest, Gradient Boosting).

Resumo

- Árvores de decisão representam decisões como uma sequência de **perguntas** sobre atributos.
- São construídas de forma **top-down**, usando critérios como entropia, ganho de informação ou índice Gini.
- A **poda** é fundamental para evitar modelos muito complexos e pouco generalizáveis.
- ID3, C4.5 e CART são algoritmos clássicos que diferem principalmente nos critérios de divisão e no tratamento de atributos.
- Apesar de limitações, são uma das ferramentas mais importantes e interpretáveis em aprendizado de máquina.