

CIÊNCIA DE DADOS E INTELIGÊNCIA ARTIFICIAL

ENSEMBLE LEARNING

Prof. Anderson França




MINISTÉRIO DA
SAÚDE



MATRIZ CURRICULAR

Essa disciplina explora conceitos fundamentais da ciência de dados, incluindo técnicas de coleta, limpeza e análise de dados. Serão utilizados **algoritmos de aprendizado de máquina**, como regressão, árvores de decisão e redes neurais, aplicados à previsão e classificação. Além disso, serão utilizados métodos de automação de processos utilizando ferramentas modernas e técnicas avançadas para otimizar e escalonar soluções analíticas.



1. Introdução à análise e ciência de dados	44
2. Estatística aplicada I – Cultura orientada a dados e transformação digital	90
3. Estatística aplicada II – Ciência de dados, aprendizado de máquinas e otimização	136
4. <i>Deep learning</i> e inteligência artificial	50
5. Inteligência artificial aplicada	80

OBJETIVO DA AULA

O objetivo desta aula é capacitar os alunos a **compreender e aplicar técnicas de Ensemble Learning**, explorando modelos que combinam múltiplos classificadores para melhorar a precisão e robustez das previsões.

Ao final da aula, os alunos serão capazes de:

- **Entender os conceitos de Ensemble Learning**, incluindo Bagging e Boosting.
- **Explicar e aplicar o Random Forest e Extra Trees** como exemplos de Bagging.
- **Compreender o funcionamento do Boosting e suas variações**, incluindo AdaBoost, Gradient Boosting, XGBoost e LightGBM.
- **Comparar as diferenças entre Bagging e Boosting** e saber quando utilizar cada um.
- **Implementar modelos de Ensemble Learning no Python usando scikit-learn.**
- **Ajustar hiperparâmetros para otimizar o desempenho dos modelos ensemble.**
- **Analisar a importância das variáveis (Feature Importance)** para melhorar a interpretabilidade dos modelos.

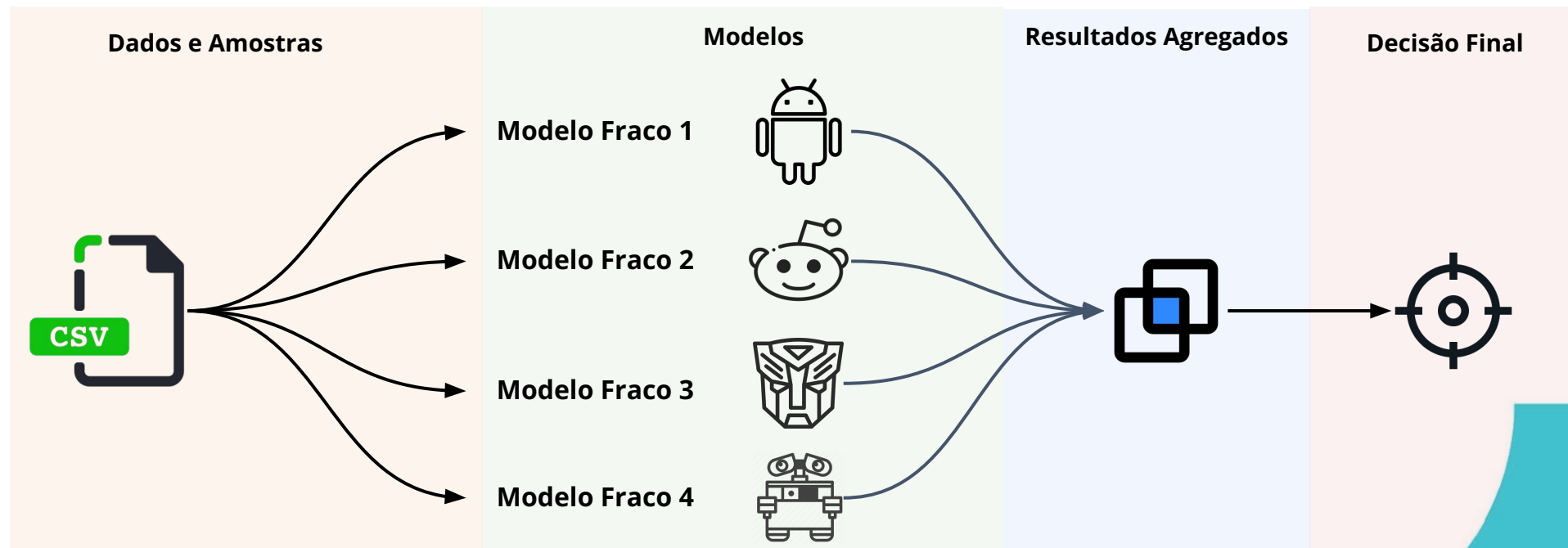
CIÊNCIA DE DADOS E INTELIGÊNCIA ARTIFICIAL

ENSEMBLE DE MODELOS

ENSEMBLE DE MODELOS

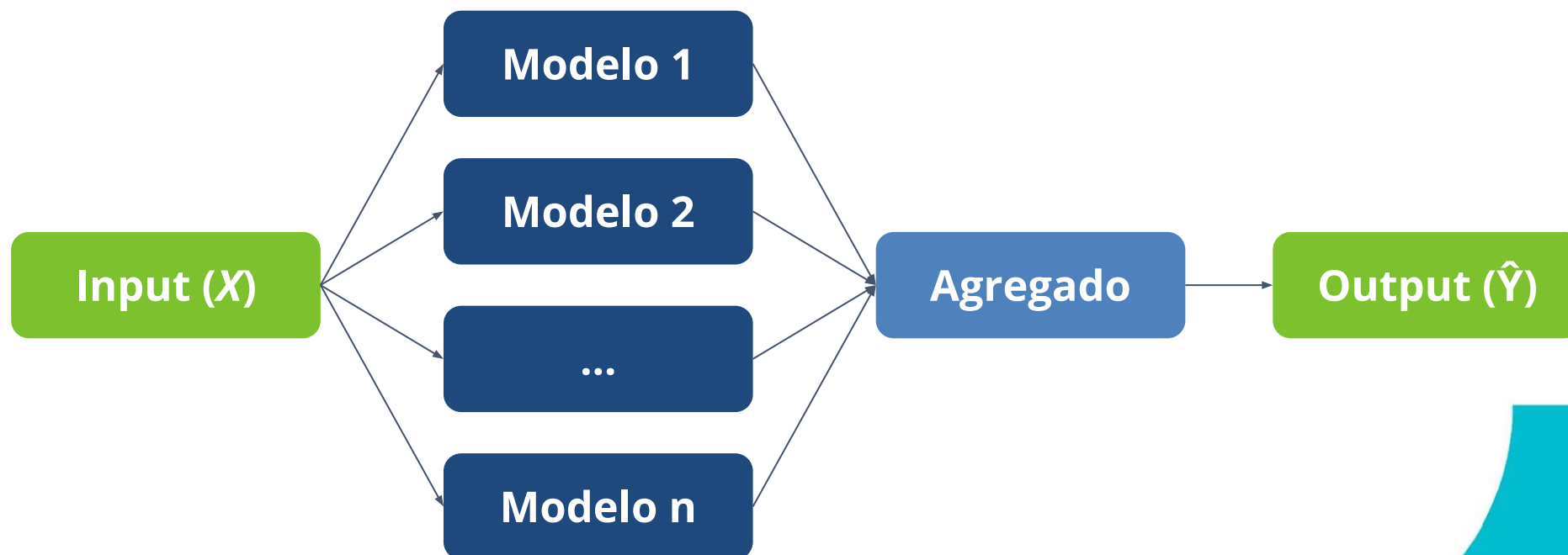
Até agora, exploramos modelos individuais de classificação, como **Regressão Logística** e **Árvores de Decisão**, que apresentam bons resultados, mas podem ter limitações dependendo do problema. Agora, vamos aprofundar um conceito avançado do Machine Learning:

Ensemble de Modelos



ENSEMBLE DE MODELOS

Os modelos ensemble **combinam múltiplos modelos** individuais para melhorar a precisão das previsões e aumentar a robustez da solução. Em vez de confiar em um único modelo, essa abordagem permite que diferentes modelos colaborem, reduzindo os erros e aumentando a generalização.



PRINCIPAIS TÉCNICAS

Existem vários tipos de modelos ensemble, incluindo:

- **Bagging (Bootstrap Aggregating):** Cria várias versões do modelo treinadas em subconjuntos diferentes dos dados e combina suas previsões. O **Random Forest** é uma implementação aplicada a árvores de decisão.
- **Boosting:** os modelos são treinados sequencialmente, e cada modelo se concentra em corrigir os erros cometidos pelos modelos anteriores, ajustando os pesos dos exemplos de dados mal previstos.
- **Stacking:** Usa diferentes modelos base para gerar previsões, que são combinadas por um modelo final de agregação.

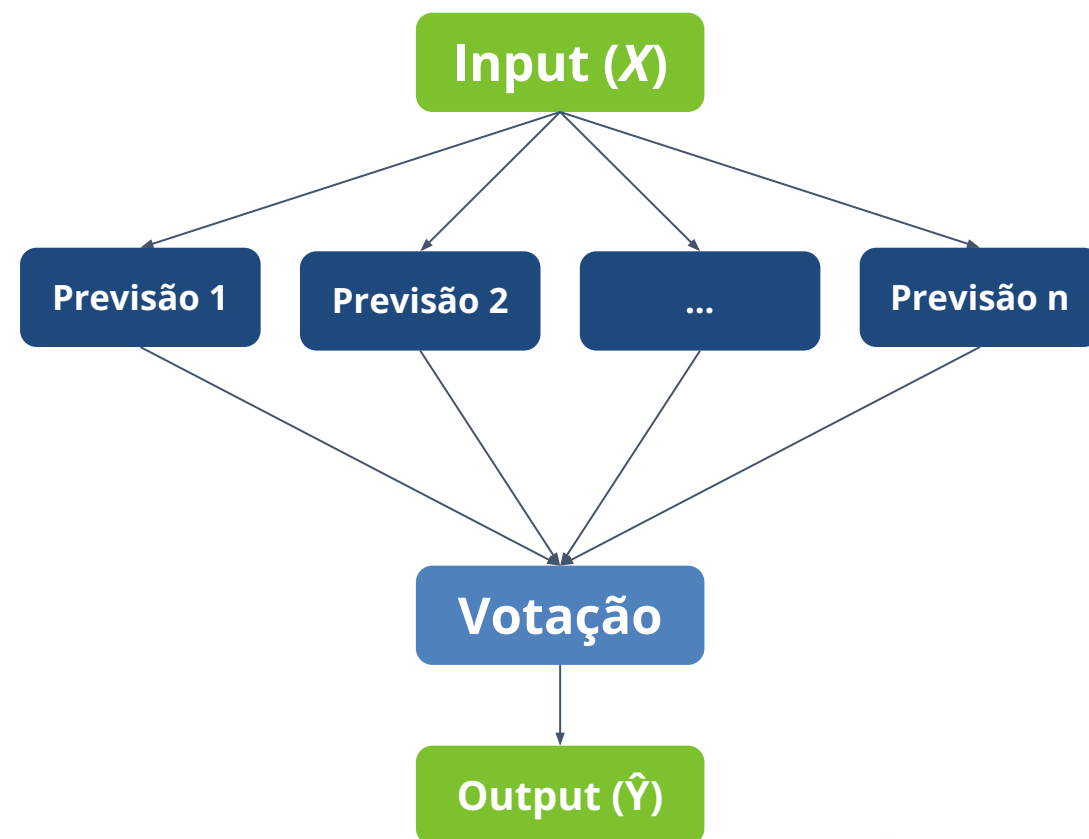
CIÊNCIA DE DADOS E INTELIGÊNCIA ARTIFICIAL

BAGGING (Bootstrap Aggregating)

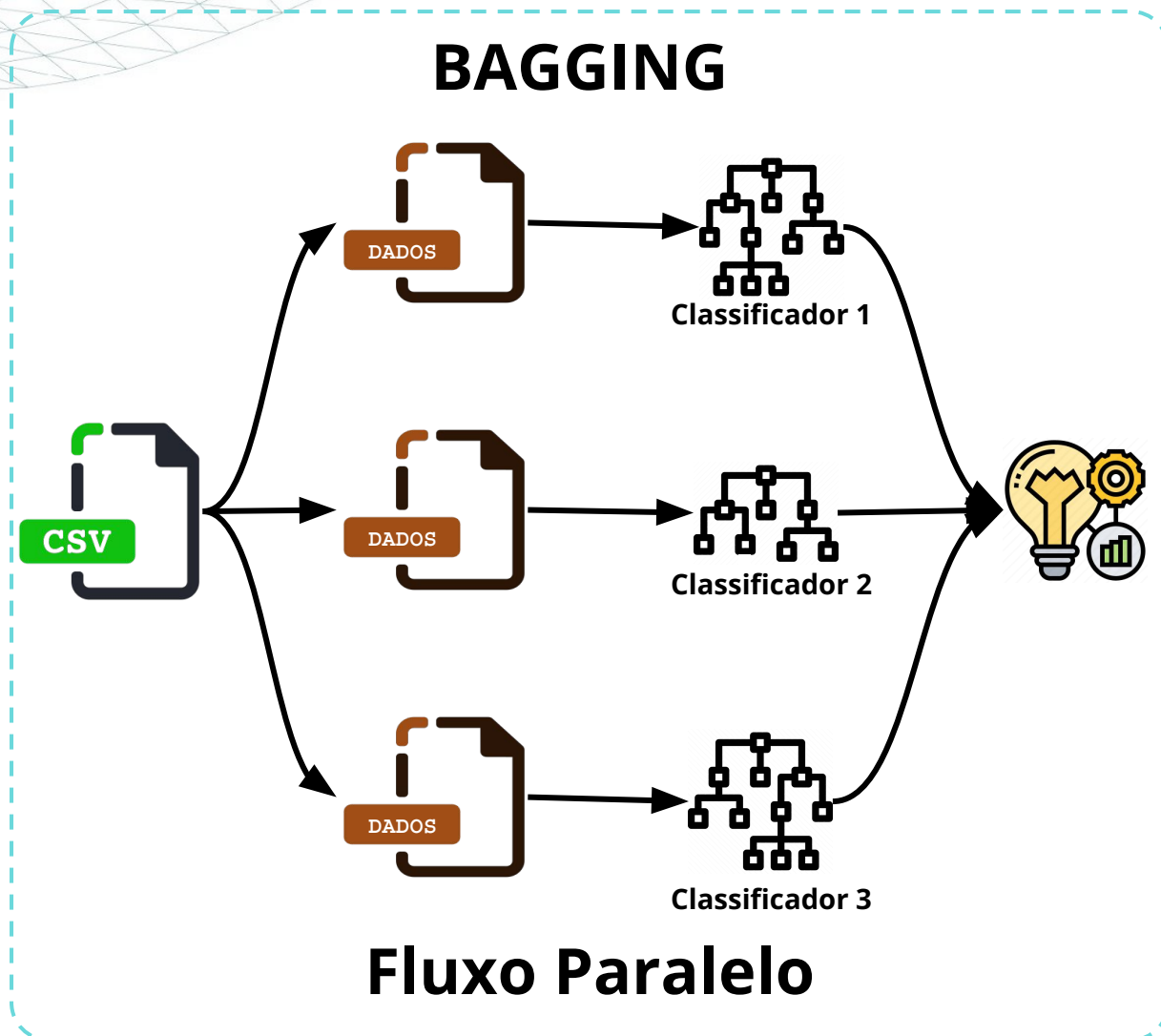
BAGGING

Bagging (Bootstrap Aggregating) combina múltiplos modelos de aprendizado de máquina treinados em amostras aleatórias dos dados de treinamento, com o **objetivo de melhorar a precisão e a robustez das previsões** e reduzindo o risco de *overfitting*.

O processo de bagging envolve a criação de várias amostras de bootstrap (amostras com reposição) dos dados originais. Cada amostra é usada para treinar um modelo independente. As previsões desses modelos são combinadas para produzir uma previsão final.



COMO FUNCIONA O BAGGING?

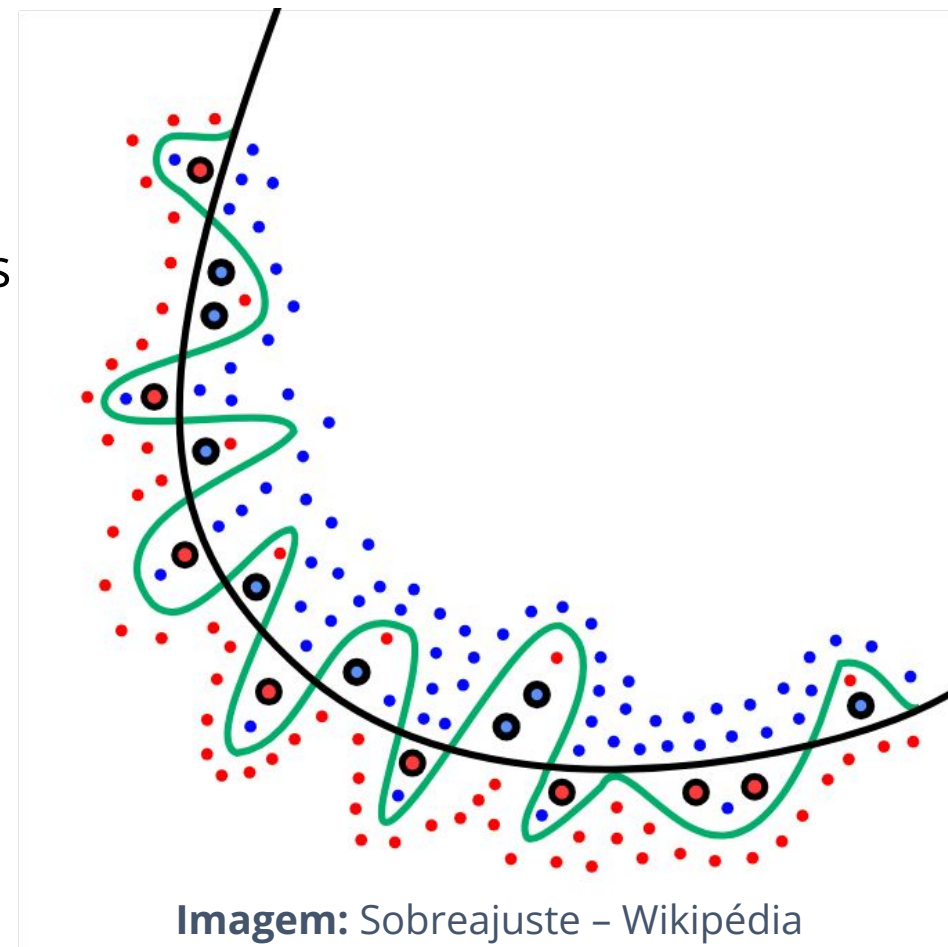


O **Bagging** cria um conjunto de modelos a partir de diferentes amostras de um mesmo conjunto de dados. Podemos dividir o modelo em três etapas principais:

- **Criar subconjuntos de dados:** Gera várias amostras do conjunto de treinamento usando amostragem com reposição (bootstrap).
- **Treinar múltiplos modelos:** Cada subconjunto de dados treina um modelo independente, geralmente do mesmo tipo.
- **Combinar previsões:**
 - **Classificação:** votação majoritária
 - **Regressão:** média das previsões.

VANTAGENS DO BAGGING

- **Redução do overfitting** – Torna o modelo mais generalizável e robusto.
- **Aumento da estabilidade das previsões** – Pequenas variações nos dados não impactam fortemente o resultado.
- **Aplicável a diferentes algoritmos** – Pode ser usado com **Árvores de Decisão, SVMs e Redes Neurais**.
- **Resistência a outliers** – Modelos treinados separadamente reduzem o impacto de valores extremos.



PONTOS A CONSIDERAR

- **Maior custo computacional:** Treinar múltiplos modelos exige mais processamento.
- **Interpretação mais complexa:** Combinando várias previsões, torna-se mais difícil entender cada decisão individual.
- **Necessidade de dados suficientes:** Como os dados são reamostrados, conjuntos muito pequenos podem não se beneficiar tanto.



Imagem: Adobe Stock

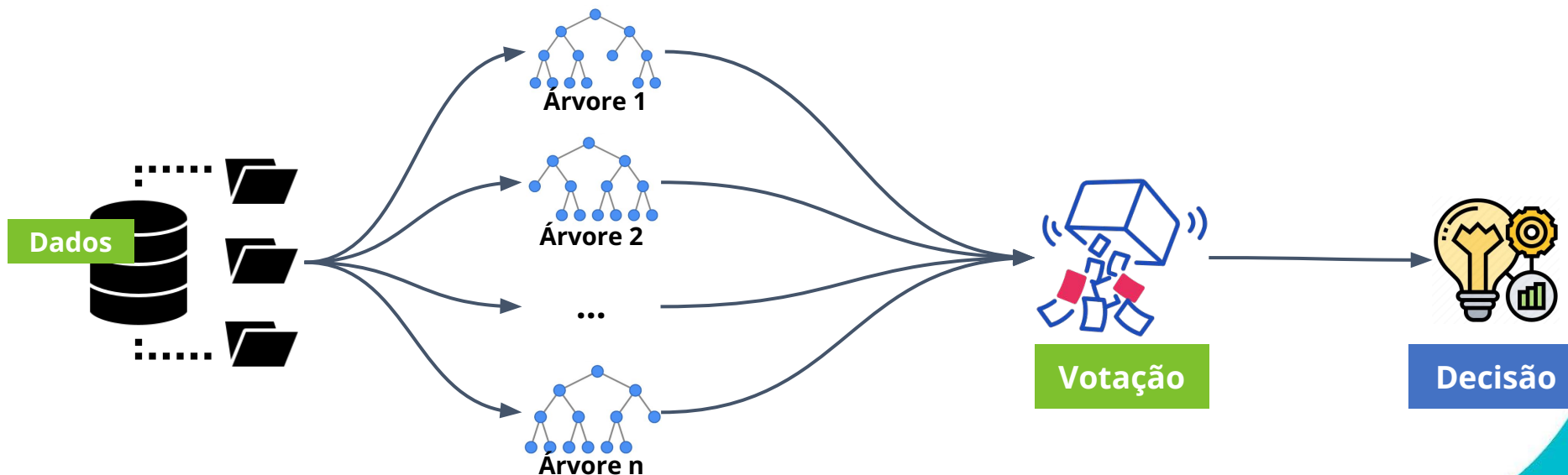
CIÊNCIA DE DADOS E INTELIGÊNCIA ARTIFICIAL

RANDOM FOREST

RANDOM FOREST

O **Random Forest** é um algoritmo de *ensemble learning* baseado em *bagging* que combina múltiplas árvores de decisão para obter maior precisão, estabilidade e generalização, reduzindo problemas como *overfitting* e alta variância.

Ele é projetado para melhorar a precisão das previsões e a robustez do modelo, corrigindo algumas limitações das árvores de decisão individuais, como o overfitting e sensibilidade a pequenas variações nos dados

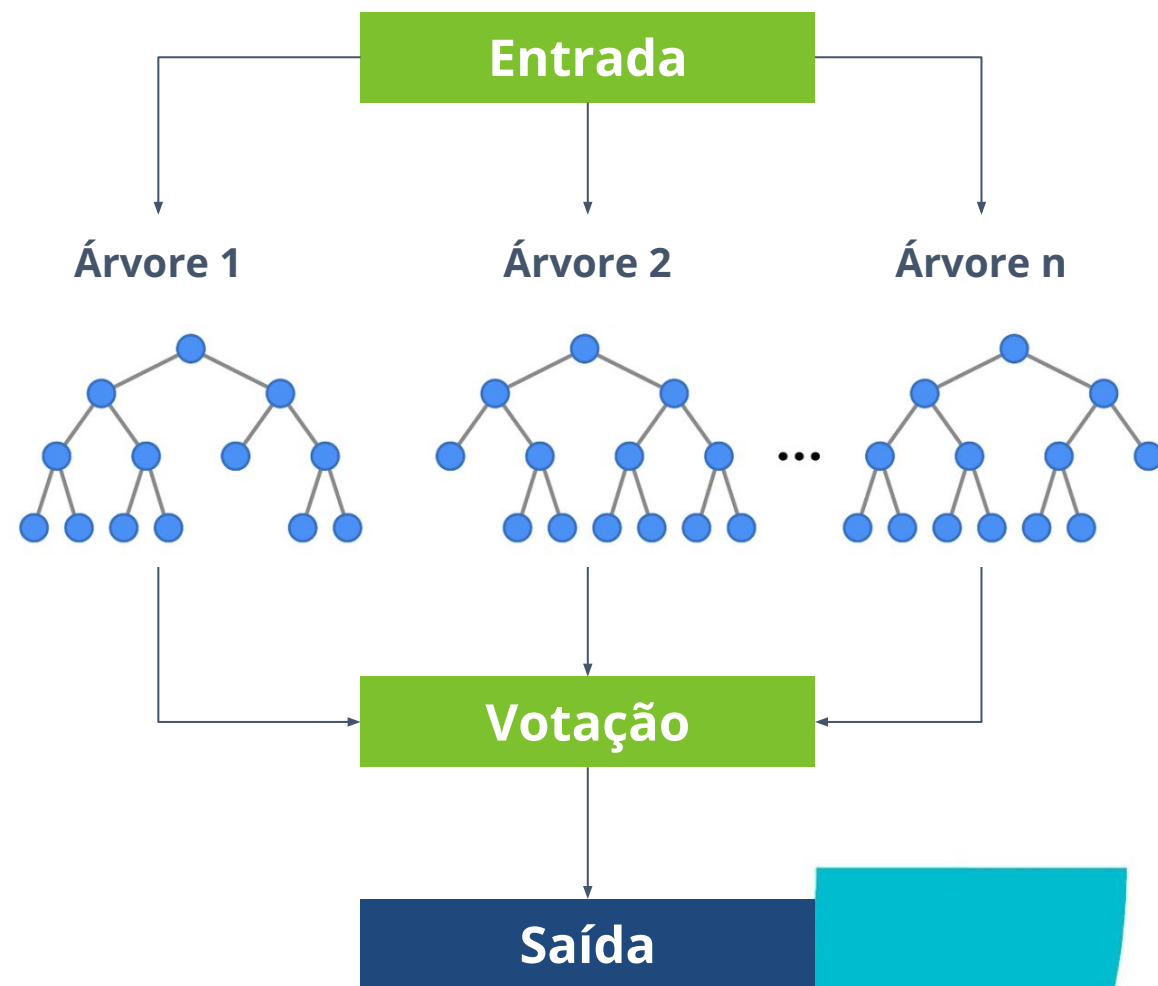


ETAPAS DO RANDOM FOREST

Construção das Árvores (Bootstrapping): O modelo gera várias amostras aleatórias da base usando o *bootstrap* e cada árvore é treinada de forma independente com cada amostra.

Seleção aleatória de variáveis: Diferente da árvore de decisão, onde podemos usar todas as variáveis, no Random Forest cada árvore escolhe um subconjunto aleatório de variáveis para decidir a melhor divisão dos nós.

Combinação das Previsões: As previsões das árvores são combinadas. Para classificação, usa-se votação majoritária, e para regressão, a média das previsões das árvores é utilizada.



VANTAGENS E DESVANTAGENS

VANTAGENS	DESVANTAGENS
Reduz overfitting – Ao combinar múltiplas árvores, evita que o modelo memorize os dados de treino.	Maior custo computacional – Exige mais tempo e memória do que uma única árvore.
Estabilidade e precisão – Pequenas variações nos dados não afetam tanto o modelo.	Menor interpretabilidade – Com muitas árvores, pode ser difícil entender a lógica da previsão.
Importância das variáveis – Permite avaliar quais variáveis influenciam mais as previsões.	Menos eficiente para inferência em tempo real – Consultar várias árvores pode tornar a previsão mais lenta.
Versatilidade – Funciona bem para classificação e regressão.	Pode ser menos eficiente em alta dimensionalidade – Se houver muitas variáveis, pode ser necessário aplicar redução de dimensionalidade (PCA, Seleção de Variáveis).

TREINAMENTO DO MODELO

É importante notar que o modelo de Random Forest pode ser computacionalmente intensivo, especialmente com grandes bases de dados. Além disso, a interpretabilidade do modelo pode ser limitada devido à complexidade das árvores de decisão individuais no modelo.

Para treinar o modelo, é necessário seguir as etapas de modelagem:

- Tratar e selecionar as melhores variáveis
- Dividir a base de dados
- Treinar o modelos
- Ajustar os hiperparâmetros
- Avaliar a performance

RANDOM FOREST NO PYTHON

No Python, o modelo de Random Forest pode ser implementado usando a biblioteca scikit-learn. A classe **RandomForestClassifier** é usada para problemas de classificação e a classe **RandomForestRegressor** é usada para problemas de regressão.

```
sklearn.ensemble.RandomForestClassifier(n_estimators=100, *, criterion='gini', max_depth=None, min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='sqrt', max_leaf_nodes=None, min_impurity_decrease=0.0, bootstrap=True, oob_score=False, n_jobs=None, random_state=None, verbose=0, warm_start=False, class_weight=None, ccp_alpha=0.0, max_samples=None)
```

Confira a definição completa dos hiperparâmetros em: [Scikitlearn - Random Forest](#)

TREINANDO O MODELO

```
#Carregar bibliotecas necessárias
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report

#Particionar base de dados
X_train, X_test, y_train, y_test = train_test_split(x, y,
                                                    test_size=0.2,
                                                    random_state=0)
```

HIPERPARÂMETROS DO MODELO

Os **hiperparâmetros** são configurações pré-definidas que controlam o funcionamento do algoritmo **antes do treinamento**. Diferente dos **parâmetros** do modelo (como os pesos em uma regressão linear ou os splits de uma árvore de decisão, que são aprendidos durante o treinamento), os hiperparâmetros precisam ser **ajustados manualmente** ou por técnicas de otimização para obter o melhor desempenho.

```
modelo = RandomForestClassifier(n_estimators=100, random_state=0)
```

Parâmetro	Descrição	Impacto
n_estimators	Número de árvores na floresta.	Maior = Mais estabilidade, mas maior custo computacional.
max_depth	Profundidade máxima das árvores.	Limitar evita overfitting, mas reduzir demais pode causar underfitting.
max_features	Número máximo de variáveis consideradas por nó.	Menos variáveis = mais diversidade entre árvores.
min_samples_split	Mínimo de amostras para dividir um nó.	Valores muito pequenos podem gerar overfitting.
random_state	Semente aleatória para reprodutibilidade.	Garante que os resultados sejam sempre os mesmos.
min_samples_leaf	Número mínimo de amostras necessárias para formar uma folha.	Valores menores tornam a árvore mais sensível aos dados, enquanto valores maiores reduzem overfitting.
bootstrap	Define se a amostragem dos dados será feita com reposição (True) ou sem reposição (False).	Quando False, todas as árvores usam todos os dados, reduzindo a diversidade do ensemble.

TREINANDO O MODELO

```
#Ajustar o modelo
modelo.fit(X_train, y_train)

#Previsão na Base Teste
y_pred = modelo.predict(X_test)

#Visualizar resultados
print(classification_report(y_test, y_pred))
```

O ajuste correto dos hiperparâmetros permite evitar overfitting, melhorar a precisão e otimizar o tempo de treinamento. Métodos como Grid Search e Random Search ajudam a encontrar as melhores combinações.

Os hiperparâmetros controlam como o modelo aprende, enquanto os parâmetros são aprendidos pelo modelo durante o treinamento.

AJUSTES: MODELO COM OVERFITTING

O modelo tem **alta precisão no treino**, mas **baixo desempenho no teste**, indicando que está "memorizando" os dados de treino e não generalizando bem para novos exemplos.

Solução: Reduzir a complexidade do modelo para evitar que memorize os dados de treino.

Hiperparâmetro	Sugestão	Impacto esperado
max_depth	Reduzir (ex: 5-15)	Impede que as árvores cresçam demais e memorizem os dados.
min_samples_split	Aumentar (ex: 5-10)	Evita divisões excessivas e reduz a profundidade das árvores.
min_samples_leaf	Aumentar (ex: 2-10)	Faz com que folhas tenham mais exemplos, reduzindo overfitting.
max_features	Reduzir (ex: "sqrt" ou "log2")	Aumenta a diversidade das árvores e reduz correlação entre elas.
ccp_alpha	Ajustar (ex: 0.01-0.1)	Ativa poda (pruning) para eliminar ramos irrelevantes.

AJUSTES: MODELO COM UNDERFITTING

O modelo tem **desempenho ruim tanto no treino quanto no teste**, sugerindo que está muito simples para capturar padrões complexos nos dados.

Solução: Aumentar a complexidade do modelo para capturar melhor os padrões dos dados.

Hiperparâmetro	Sugestão	Impacto esperado
n_estimators	Aumentar (ex: 200-500)	Mais árvores reduzem viés e aumentam estabilidade.
max_depth	Aumentar (ex: 20+)	Permite que as árvores capturem padrões mais complexos.
min_samples_split	Diminuir (ex: 2-4)	Permite que as árvores dividam mais os nós e capturem mais detalhes.
min_samples_leaf	Diminuir (ex: 1)	Garante que o modelo capture padrões sutis nos dados.
max_features	Aumentar (ex: "auto" ou um valor alto)	Usa mais variáveis por nó, aumentando a capacidade preditiva.

AJUSTES: TEMPO DE TREINAMENTO ALTO

O tempo para treinar o modelo está **excessivamente longo**, tornando inviável ajustes frequentes ou aplicações em **grandes volumes de dados**.

Solução: Reduzir a carga computacional sem comprometer muito a precisão.

Hiperparâmetro	Sugestão	Impacto esperado
n_estimators	Reduzir (ex: 50-100)	Diminui o número de árvores e acelera o treinamento.
max_depth	Limitar (ex: 10-20)	Reduz o tamanho das árvores, tornando-as mais eficientes.
max_features	Reduzir (ex: "sqrt")	Usa menos variáveis em cada divisão, acelerando cálculos.
n_jobs	Ajustar para -1	Usa todos os núcleos do processador para treinar mais rápido.
bootstrap	Manter True	Mantém o processo de amostragem, garantindo diversidade entre as árvores.

AJUSTES: DADOS DESBALANCEADOS

O modelo tem dificuldade em aprender classes minoritárias, resultando em precisões muito baixas para algumas categorias.

Solução: Ajustar hiperparâmetros para garantir que o modelo aprenda as classes minoritárias.

Hiperparâmetro	Sugestão	Impacto esperado
class_weight	"balanced"	Ajusta pesos das classes automaticamente para evitar viés para a maioria.
min_samples_leaf	Aumentar (ex: 3-10)	Garante que as folhas tenham exemplos de ambas as classes.
n_estimators	Aumentar (ex: 200-300)	Maior número de árvores melhora estabilidade em classes minoritárias.
max_depth	Ajustar (ex: 5-15)	Mantém um equilíbrio entre complexidade e generalização.

AJUSTES: MELHORAR A INTERPRETABILIDADE

O modelo faz boas previsões, mas é difícil entender como ele chegou a cada decisão, dificultando a análise por especialistas.

Solução: Ajustar o modelo para ser mais fácil de interpretar e visualizar.

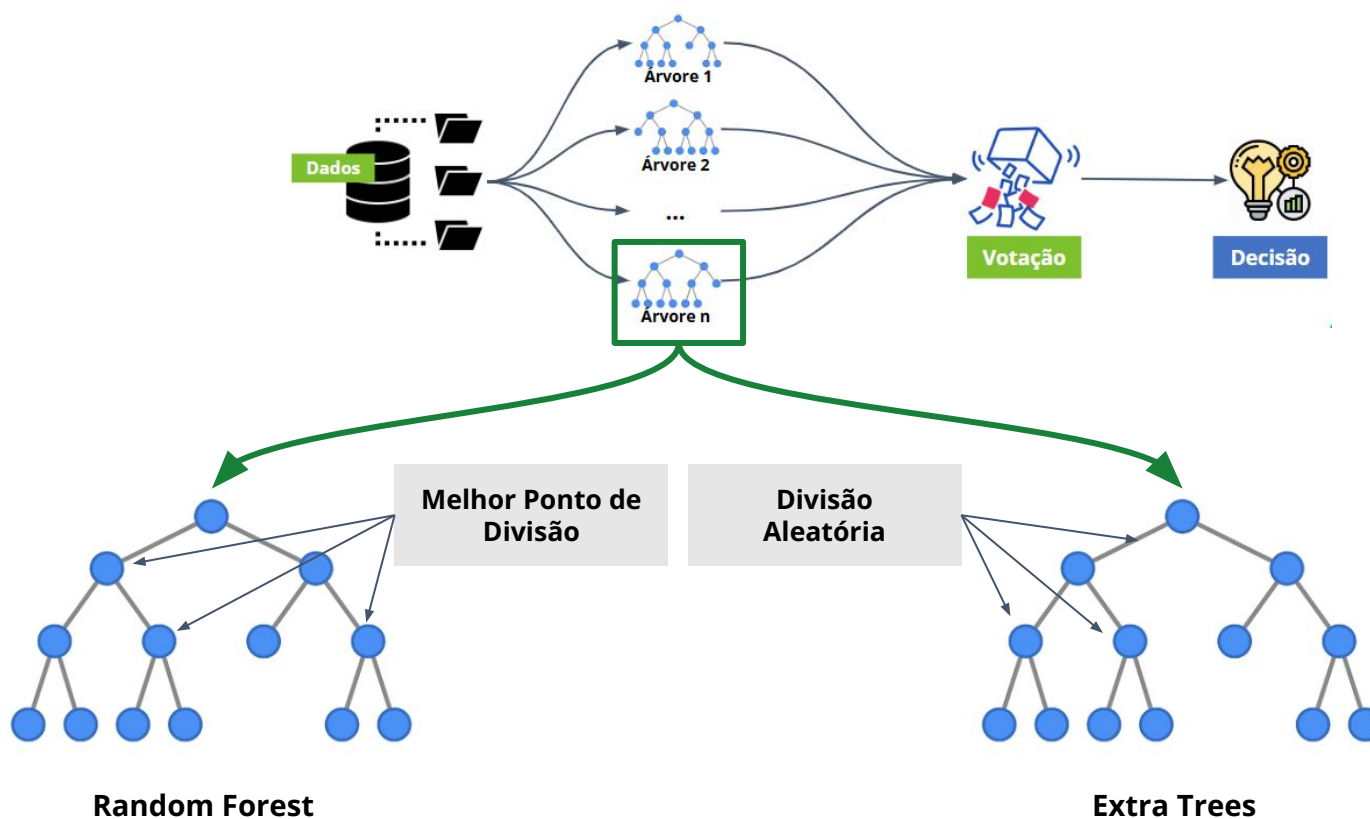
Hiperparâmetro	Sugestão	Impacto esperado
max_depth	Reduzir (ex: 5-10)	Mantém as árvores menores e mais interpretáveis.
n_estimators	Reduzir (ex: 50-100)	Evita um modelo excessivamente complexo.
oob_score	Ativar (True)	Fornece uma estimativa de erro sem necessidade de conjunto de teste.
feature_importances_	Analisar	Ajuda a visualizar quais variáveis são mais relevantes para o modelo.

CIÊNCIA DE DADOS E INTELIGÊNCIA ARTIFICIAL

EXTRA TREES (Extremely Randomized Trees)

EXTRA TREES

O **Extra Trees (Extremely Randomized Trees)** é uma variação mais rápida do algoritmo Random Forest que também usa múltiplas árvores de decisão, mas com um nível maior de aleatoriedade durante o processo de construção das árvores.



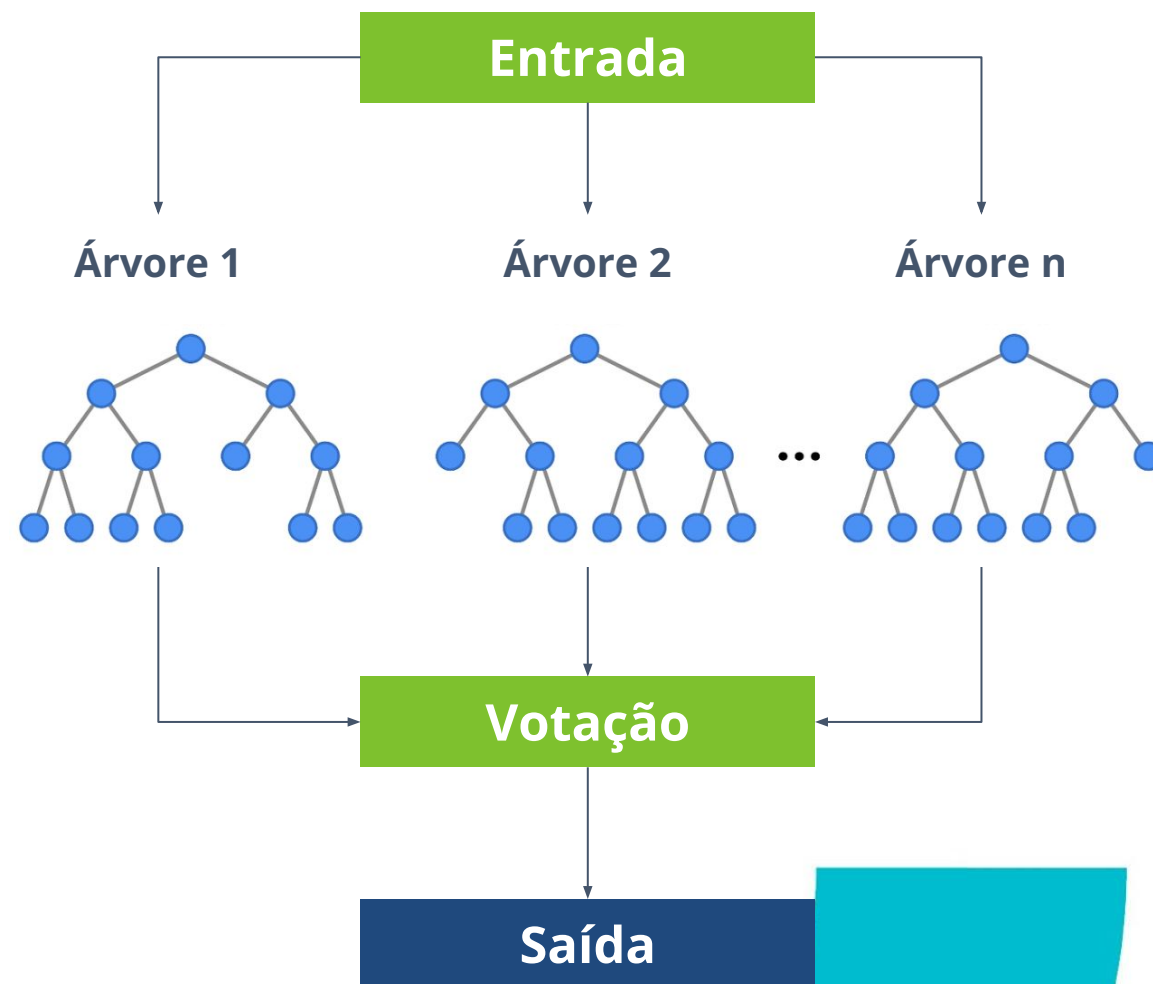
ETAPAS DO EXTRA TREES

Construção das Árvores (Bootstrapping Opcional): O modelo gera várias amostras aleatórias da base. Por padrão, ele usa todos os dados (não faz bootstrap), mas essa opção pode ser ativada.

Seleção aleatória de variáveis: Ao dividir os nós, os pontos de corte são escolhidos de forma completamente aleatória, em vez de buscar a melhor divisão.

Combinação das Previsões: As previsões das árvores são combinadas. Para classificação, usa-se votação majoritária, e para regressão, a média das previsões das árvores é utilizada.

Obs. Esse modelo tende a ser mais rápido e mais robusto contra overfitting do que o Random Forest



OBSERVAÇÕES

Ser mais rápido e mais robusto contra overfitting não significa que o Extra Trees seja sempre melhor que o Random Forest. Tudo depende do contexto e do tipo de problema.

Extra Trees pode ser uma escolha melhor em alguns cenários, como:

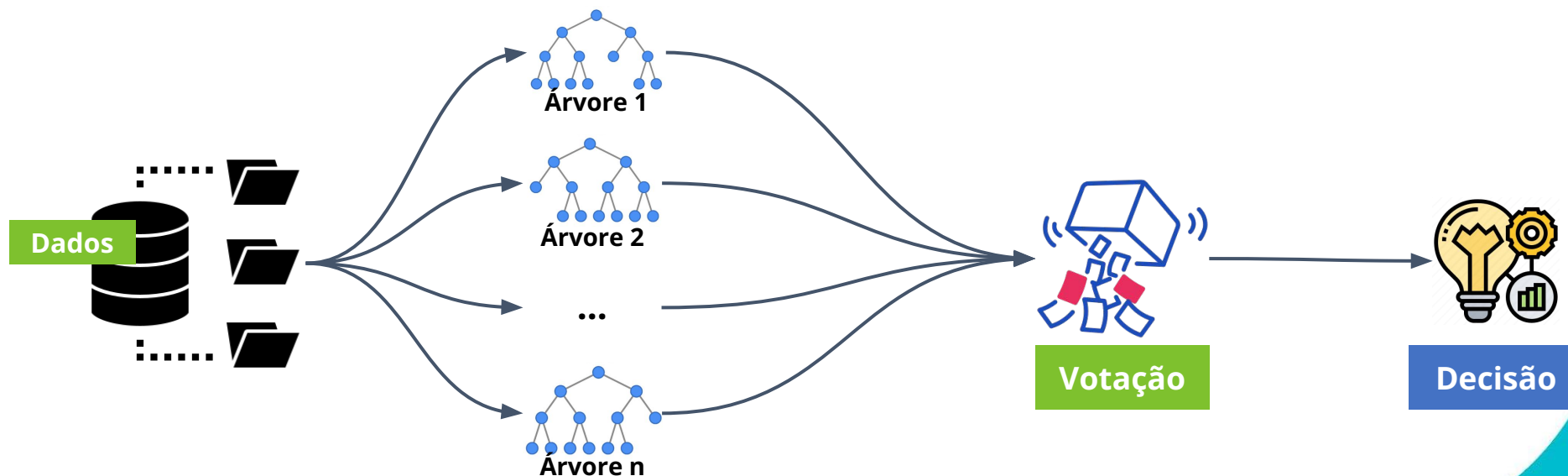
- Quando o dataset é **grande e ruidoso**, pois sua aleatoriedade ajuda a evitar que o modelo memorize padrões irrelevantes.
- Quando o **tempo de treinamento** é um fator crítico, já que ele não precisa calcular o melhor ponto de divisão para cada nó, acelerando o aprendizado.
- Quando há muito *overfitting* em um Random Forest, pois a escolha aleatória dos *splits* pode ajudar na generalização

Extra Trees pode ser inferior ao Random Forest em certos casos, como:

- **Se os dados forem pequenos ou estruturados**, onde encontrar os melhores pontos de divisão pode ser mais vantajoso.
- **Se a precisão for a principal preocupação**, pois o Random Forest pode obter melhores *splits* e, conseqüentemente, previsões mais precisas.

VANTAGENS E DESVANTAGENS

VANTAGENS	DESVANTAGENS
Treinamento mais rápido – Não precisa calcular o melhor ponto de divisão.	Pode ser menos preciso – Como os splits são totalmente aleatórios, pode gerar divisões menos ideais.
Menos propenso a overfitting – A aleatoriedade reduz o risco de memorizar padrões irrelevantes.	Menos interpretável – A aleatoriedade torna mais difícil entender as decisões individuais.
Útil para bases grandes e ruidosas – Modelos mais rápidos e robustos para grandes conjuntos de dados.	Menos eficiente para bases pequenas – Pode não capturar padrões complexos tão bem quanto o Random Forest.



TREINAR EXTRA TREES

```
#Carregar biblioteca ExtraTreesClassifier
from sklearn.ensemble import ExtraTreesClassifier

# Cria o modelo ExtraTreesClassifier
modelo = ExtraTreesClassifier(n_estimators=100, random_state=0)

# Treina o modelo com o conjunto de treinamento
modelo.fit(X_train, y_train)

# Faz as previsões com o conjunto de teste
y_pred = modelo.predict(X_test)
```


HIPERPARÂMETROS DO MODELO

Os hiperparâmetros do Extra Trees são similares aos do Random Forest, mas com algumas diferenças importantes:

```
modelo = ExtraTreesClassifier(n_estimators=100, random_state=0)
```

Hiperparâmetro	Random Forest	Extra Trees	Impacto
criterion	"gini" ou "entropy" (classificação) / "mse" (regressão)	Mesmo critério	Não muda entre os dois
max_features	Escolhe aleatoriamente um subconjunto de variáveis para cada nó	Mesmo comportamento	Reduz a correlação entre as árvores.
bootstrap	True por padrão (usa amostragem com reposição)	False por padrão (usa todos os dados)	No Extra Trees, ativar bootstrap pode reduzir o viés, mas normalmente não é necessário.
splitter	Escolhe o melhor ponto de divisão para cada nó	Escolhe um ponto aleatório dentro dos limites da variável	Extra Trees treina mais rápido, mas pode ser menos preciso.
n_estimators	Número de árvores a serem treinadas	Mesmo comportamento	Mais árvores = modelo mais estável
max_depth	Controla a profundidade máxima das árvores	Mesmo comportamento	Impacta overfitting e capacidade do modelo
min_samples_split	Número mínimo de amostras para dividir um nó	Mesmo comportamento	Define o tamanho mínimo dos grupos antes da divisão
min_samples_leaf	Número mínimo de amostras em uma folha	Mesmo comportamento	Pode evitar divisões muito específicas

CIÊNCIA DE DADOS E INTELIGÊNCIA ARTIFICIAL

FEATURE IMPORTANCE

QUAIS VARIÁVEIS IMPACTAM O MODELO?

Nem todas as variáveis contribuem igualmente para uma previsão. Algumas são essenciais, enquanto outras podem ser irrelevantes ou até prejudiciais. Como identificar quais realmente importam?

É aqui que entra a **Feature Importance**!

Entender a *Feature Importance* permite simplificar o modelo, aumentar a eficiência e, o mais importante, melhorar a compreensão sobre **quais variáveis realmente importam** no processo de tomada de decisão.

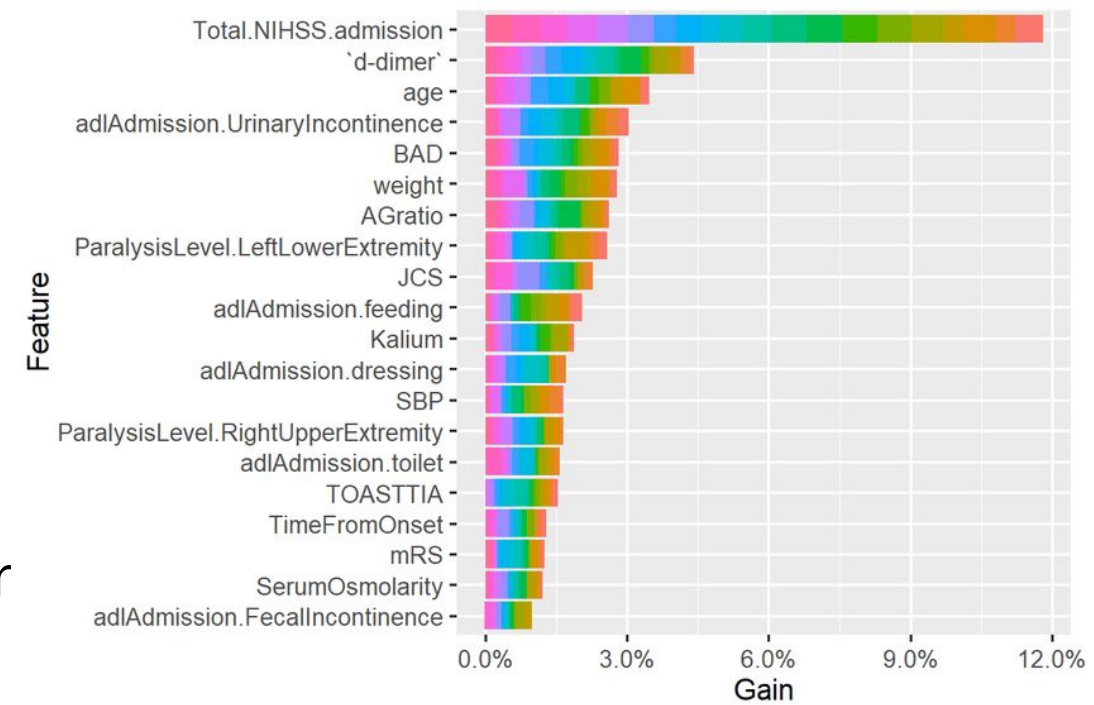


Imagem: [Feature Importance](#)

FEATURE IMPORTANCE

O Feature Importance é usado para **avaliar a relevância** ou **importância de cada variável** (*feature*) na construção de um modelo. Ele ajuda a identificar quais variáveis têm maior impacto nas previsões do modelo e permitindo a compreensão de quais variáveis estão influenciando mais o resultado.

O Feature Importance pode ser utilizado principalmente para:

- **Selecionar as variáveis mais influentes**, removendo variáveis irrelevantes.
- **Melhorar a interpretação do modelo**, facilitando a explicação para tomadores de decisão.
- **Reduzir overfitting**, eliminando atributos redundantes e otimizando o modelo.

FUNCIONAMENTO DO *FEATURE IMPORTANCE*

Árvores de Decisão: Em algoritmos como Random Forest e Extra Trees, cada árvore de decisão realiza divisões nos dados com base nas variáveis que melhor separam as classes ou valores. A importância de uma variável (*feature importance*) é determinada pelo quanto ela reduz a impureza dos nós em um modelo de árvore, utilizando critérios como Gini ou Entropia.

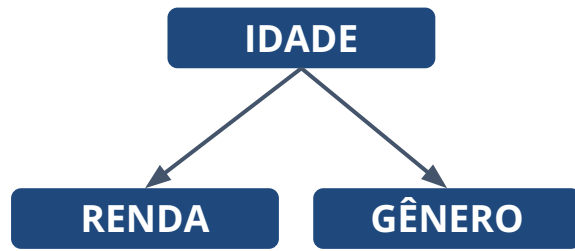
Medição do Impacto: O Feature Importance mede o impacto total de cada variável somando todas as reduções de impureza nos nós em que foi usada. Quanto mais vezes uma variável aparece e quanto maior a redução de impureza que ela causa, maior será sua importância no modelo.

Normalização: Para facilitar a análise, as importâncias das variáveis são normalizadas, de forma que a soma total seja igual a 1 (ou 100%). Isso permite uma comparação clara entre as variáveis, destacando quais são mais influentes na tomada de decisão do modelo

CÁLCULO FEATURE IMPORTANCE

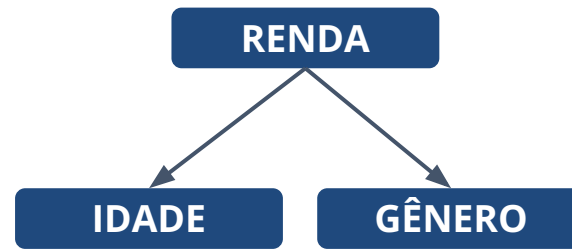
Temos um modelo de Random Forest com 3 árvores, e as variáveis disponíveis são: **Idade, Renda e Gênero**. Cada árvore faz divisões nos dados com base na redução de impureza (Gini, Entropia ou MSE).

Árvore 1



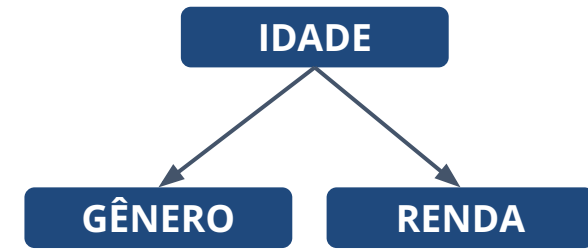
- Idade reduz impureza em 0.30
- Renda reduz impureza em 0.20
- Gênero reduz impureza em 0.10

Árvore 2



- Renda reduz impureza em 0.40
- Idade reduz impureza em 0.25
- Gênero reduz impureza em 0.05

Árvore 3



- Idade reduz impureza em 0.35
- Gênero reduz impureza em 0.15
- Renda reduz impureza em 0.25

Somamos a **redução de impureza** de cada variável ao longo das 3 árvores:

Variável	Árvore 1	Árvore 2	Árvore 3	Soma Total
Idade	0.30	0.25	0.35	0.90
Renda	0.20	0.40	0.25	0.85
Gênero	0.10	0.05	0.15	0.30

Normalizamos os valores para que a soma total seja 100%

Variável	Importância (%)
Idade	$(0.90/2.05) \times 100 = 43.9\%$
Renda	$(0.85/2.05) \times 100 = 41.5\%$
Gênero	$(0.30/2.05) \times 100 = 14.6\%$

FEATURE IMPORTANCE NO PYTHON

Após treinar o modelo de **Random Forest** ou **Extra Trees** no Python, é possível obter o **feature importance** diretamente.

```
# Gerar base de dados com features e importância
importancia_bd = pd.DataFrame({
    'feature': modelo.feature_names_in_,
    'importancia': modelo.feature_importances_
})

# Ordenar o dataframe pela importância das features
importancia_bd = importancia_bd.sort_values('importancia',
                                             ascending=False)
```

VISUALIZAR GRÁFICO DE FEATURE IMPORTANCE

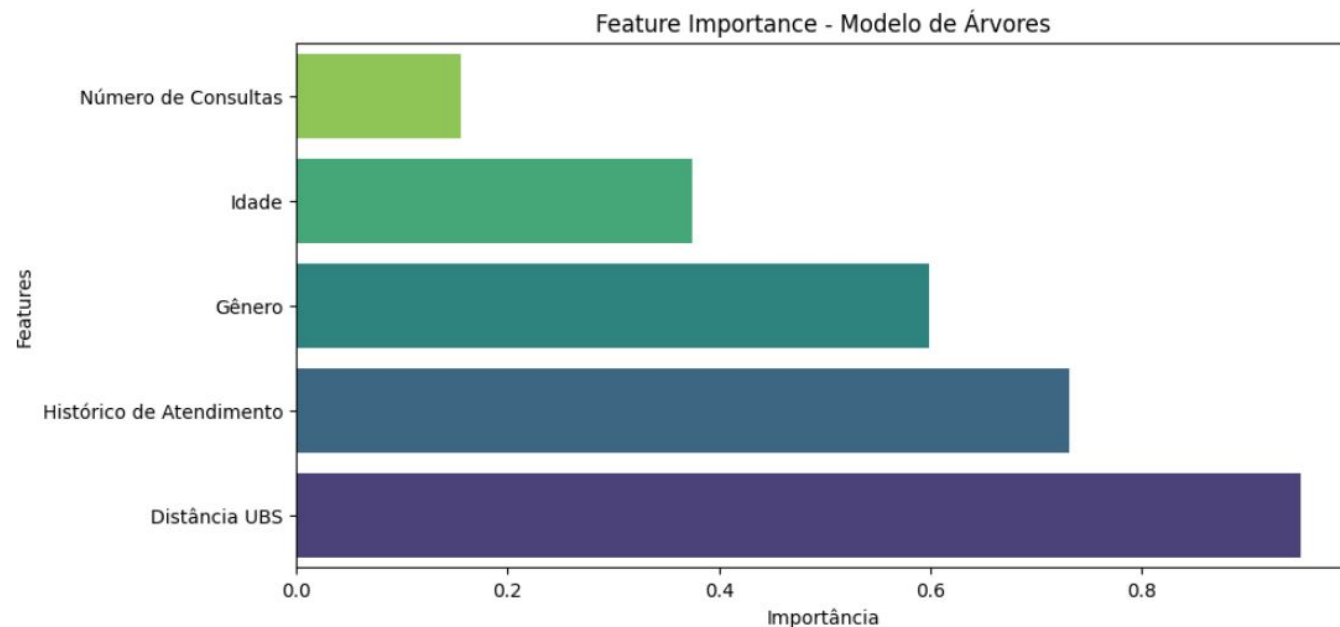
```
import matplotlib.pyplot as plt

# Criar gráfico de barras horizontais
plt.figure(figsize=(10, 5))
sns.barplot(x=importancia_bd["importancia"],
            y=importancia_bd["feature"],
            palette="viridis")

# Adicionar rótulos e título
plt.xlabel("Importância")
plt.ylabel("Features")
plt.title("Feature Importance - Modelo RF")

# Inverter eixo Y para ordenar corretamente
plt.gca().invert_yaxis()

# Mostrar gráfico
plt.show()
```

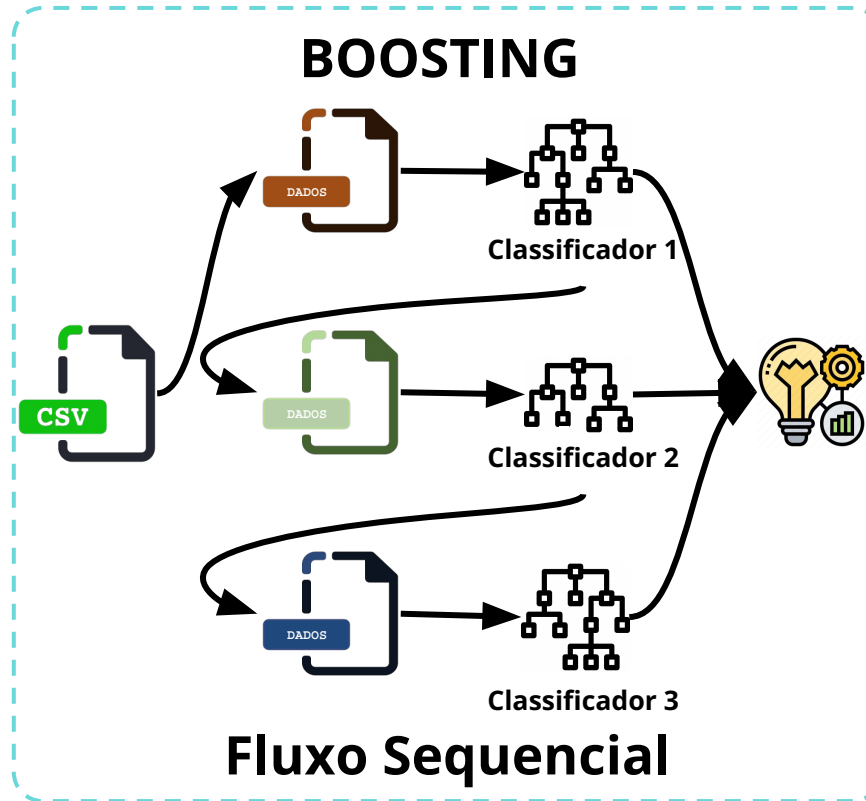
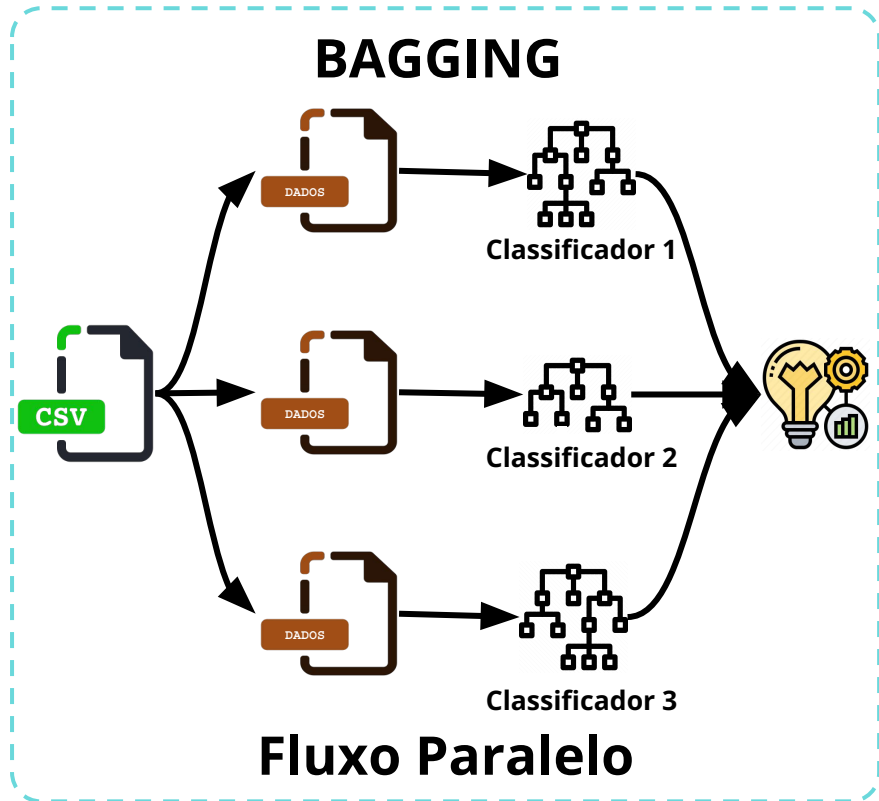


CIÊNCIA DE DADOS E INTELIGÊNCIA ARTIFICIAL

BOOSTING

BOOSTING

O **Boosting** é um método de ensemble que melhora o desempenho do modelo treinando modelos sequenciais, onde cada novo modelo tenta corrigir os erros do anterior. Diferente do Bagging, que cria modelos independentes em paralelo, o Boosting prioriza os exemplos mais difíceis, tornando o modelo mais preciso e robusto.



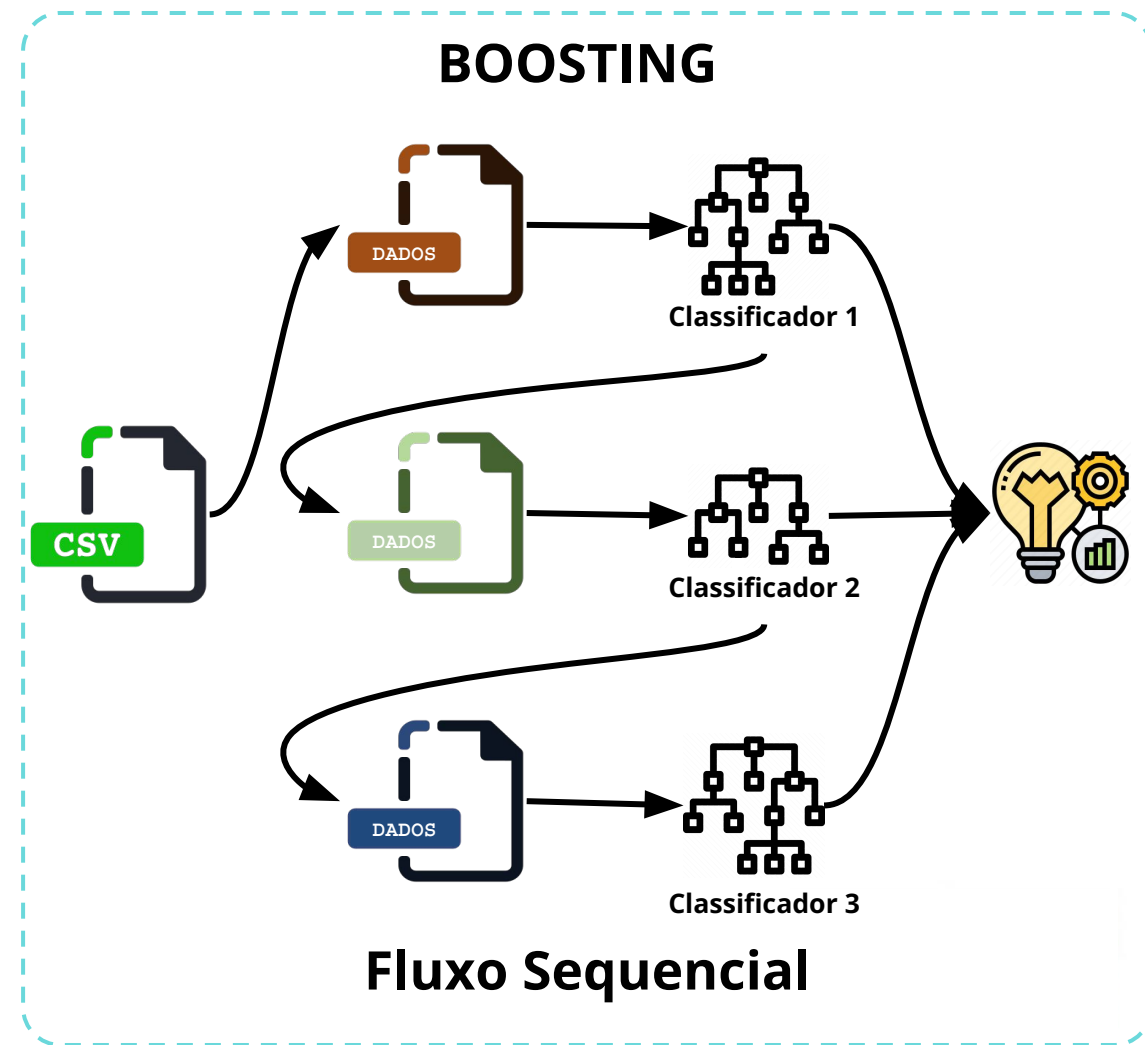
BOOSTING

O Boosting é especialmente eficaz em problemas de classificação com dados **desbalanceados**, onde uma classe é significativamente mais frequente que a outra.

Nessas situações, modelos tradicionais podem ter dificuldade em identificar padrões na classe minoritária. O Boosting resolve esse problema dando mais peso aos exemplos mal classificados, forçando o modelo a aprender melhor os padrões das classes menos representadas, resultando em um desempenho mais equilibrado.

FUNCIONAMENTO DO BOOSTING

- **Treinamento Sequencial:** O processo inicia treinando um modelo fraco (como uma árvore de decisão rasa). Em seguida, os erros desse modelo são analisados, e um novo modelo é treinado para corrigir os exemplos mal classificados. Esse ciclo se repete, com cada novo modelo aprendendo com os erros do anterior, tornando a previsão progressivamente mais precisa.
- **Ajuste de Pesos:** Os exemplos que foram mal classificados recebem um peso maior, forçando o próximo modelo a dar mais atenção a esses casos difíceis. Esse ajuste faz com que o Boosting aprenda de forma mais direcionada, corrigindo erros iterativamente.
- **Combinação Final:** Após múltiplas iterações, as previsões de todos os modelos são combinadas. A decisão final é feita por ponderação (dando mais peso aos modelos mais precisos) ou votação, resultando em um modelo mais robusto e confiável.



VANTAGENS E DESVANTAGENS

Vantagens	Desvantagens
Redução de viés: Resolve problemas de underfitting melhor que Bagging.	Computacionalmente mais caro: Treinamento sequencial pode ser lento.
Alta precisão: Especialmente eficaz para dados estruturados.	Mais sensível a ruído: Pode superajustar se houver muitos outliers.
Melhor para modelos simples: Modelos fracos (ex: árvores rasas) são transformados em modelos fortes.	Dependente de hiperparâmetros: Precisa de ajuste fino para evitar overfitting.

BOOSTING - ALGORITMOS

Existem vários modelos de Boosting disponíveis, cada um com suas características próprias. Os principais são:

Algoritmo	Quando usar?	Características
AdaBoost	Dados pequenos e simples	Reajusta pesos dos exemplos mal classificados a cada iteração.
Gradient Boosting	Quando alta precisão é necessária	Modelos sucessivos aprendem a reduzir o erro residual do anterior.
XGBoost	Bases grandes e competições de ML	Implementação otimizada do Gradient Boosting, com paralelização e regularização.
LightGBM	Treinamento extremamente rápido	Trabalha com folhas em vez de níveis, eficiente para grandes bases.
CatBoost	Quando há muitas variáveis categóricas	Processa variáveis categóricas automaticamente, reduzindo necessidade de pré-processamento.

RESUMO BAGGING E BOOSTING

Critério	Bagging (Random Forest, Extra Trees)	Boosting (AdaBoost, XGBoost, etc.)
Objetivo	Reduz variância (evita overfitting).	Reduz viés (melhora aprendizado em padrões difíceis).
Treinamento	Modelos treinados em paralelo.	Modelos treinados de forma sequencial, corrigindo erros.
Base ideal	Funciona bem com modelos complexos.	Funciona melhor com modelos fracos (ex: árvores rasas).
Computação	Treinamento rápido (paralelização).	Mais lento, pois é sequencial.
Sensibilidade a Ruído	Menos sensível a outliers.	Mais sensível a outliers, pode superajustar.
Dados desbalanceados	Pode precisar de técnicas extras para lidar com desbalanceamento.	Aprende melhor padrões da classe minoritária devido ao ajuste de pesos.

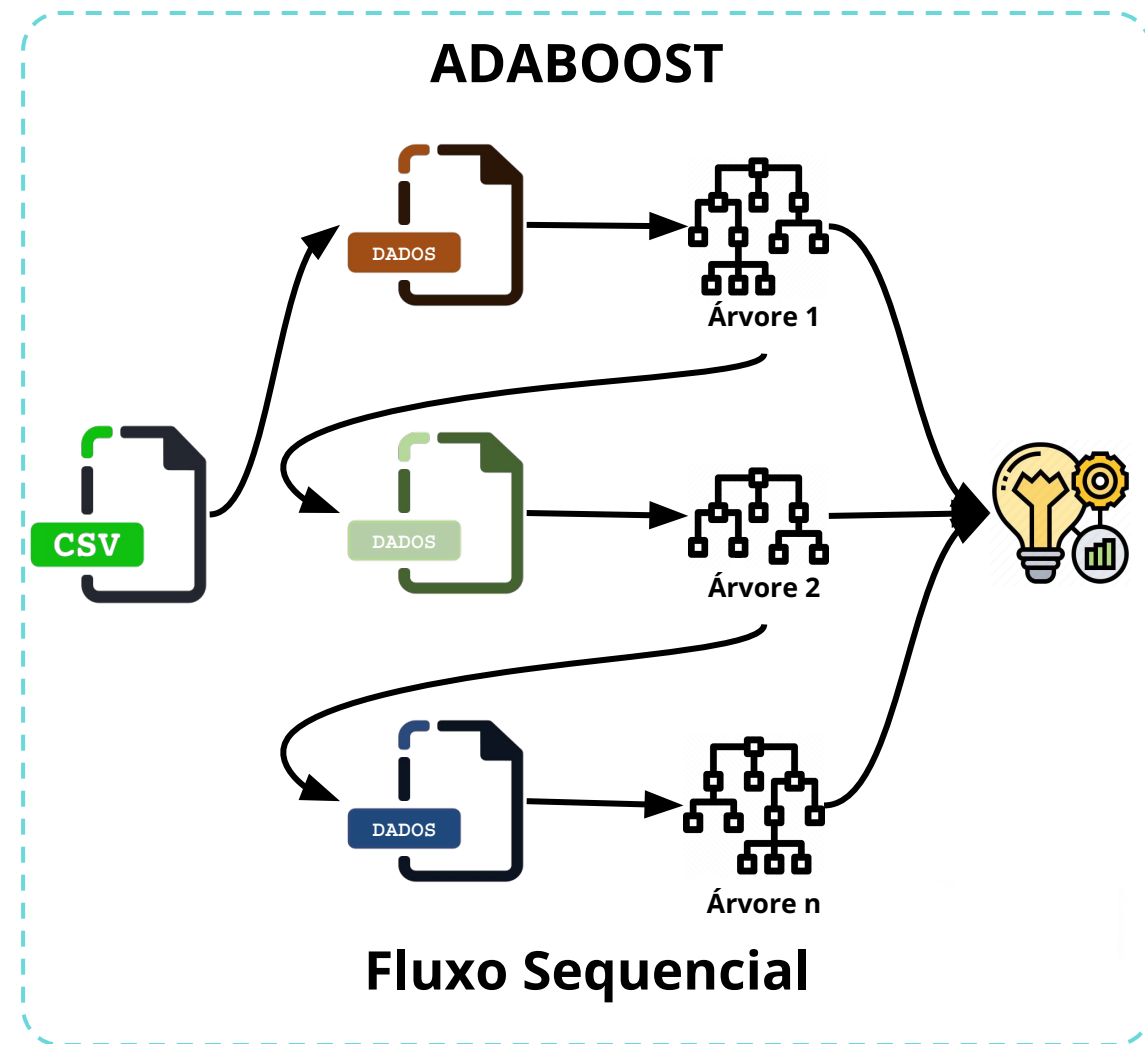
CIÊNCIA DE DADOS E INTELIGÊNCIA ARTIFICIAL

ADA BOOSTING (Adaptive Boosting)

ADABOOST

O **AdaBoost (Adaptive Boosting)** foi um dos primeiros algoritmos de Boosting e continua sendo uma das técnicas mais populares em aprendizado de máquina. Seu objetivo é combinar múltiplos "modelos fracos" (como árvores de decisão rasas) para formar um modelo forte e preciso.

Diferente de técnicas como Bagging, onde os modelos são treinados independentemente, **o AdaBoost treina modelos de forma sequencial**, onde cada novo modelo aprende com os erros do anterior



FUNCIONAMENTO

- **Treinamento Sequencial:**

- O AdaBoost começa treinando um modelo fraco (geralmente uma árvore de decisão com profundidade = 1, chamada "stump").
- Ele faz previsões nos dados e calcula os erros cometidos.

- **Ajuste de Pesos:**

- Os exemplos mal classificados recebem um peso maior.
- O próximo modelo será mais influenciado por esses exemplos difíceis.

- **Combinação dos Modelos:**


- Os modelos são combinados, dando mais peso aos modelos que tiveram melhor desempenho.
- A previsão final é feita por uma média ponderada ou votação ponderada no caso de classificação.

VANTAGENS E DESVANTAGENS

Vantagens	Desvantagens
Melhora a precisão: Corrige erros progressivamente.	Sensível a ruído: Pode superajustar se houver muitos outliers.
Fácil de interpretar: Baseado em árvores rasas, mais simples de entender.	Menos eficiente em grandes bases: Treinamento sequencial pode ser mais lento.
Funciona bem com dados desbalanceados: Aprende melhor padrões da classe minoritária.	Depende de bons modelos fracos: Se o modelo base for ruim, o AdaBoost não funcionará bem.

QUANDO USAR ADABOOST?

(ao invés de Bagging)

- **Se o modelo base tiver alto viés (underfitting)**
 - O Bagging ajuda quando o modelo está overfitting.
 - O AdaBoost é melhor se o modelo estiver underfitting, pois foca nos erros e ajusta progressivamente.
 - **Quando os dados são desbalanceados**
 - O AdaBoost ajusta os pesos dos exemplos mal classificados, dando mais atenção às classes menos frequentes.
 - O Bagging trata todas as amostras de forma igual, então pode precisar de técnicas extras (como oversampling).
 - **Quando a interpretabilidade é importante**
 - O AdaBoost usa modelos fracos (como árvores de decisão rasas), o que torna suas previsões mais explicáveis do que uma floresta densa de árvores no Random Forest.
 - **Se houver poucos dados disponíveis**
 - O AdaBoost pode funcionar melhor em bases menores, pois foca nas amostras mais difíceis.
 - O Bagging precisa de mais dados para generalizar bem.
 - **Quando for necessário um modelo menor e eficiente**
 - O AdaBoost tende a criar um modelo menor, pois usa árvores mais rasas.
 - O Random Forest gera muitas árvores profundas, o que pode aumentar o custo computacional.
- 

QUANDO NÃO USAR ADABOOST

(E optar pelo Bagging)

- **Se os dados tiverem muito ruído ou outliers:** O AdaBoost pode superajustar exemplos errados, pois dá muito peso aos outliers.
- **Se o problema for de alto risco computacional:** O Bagging é mais rápido, pois treina modelos em paralelo, enquanto o AdaBoost é sequencial.
- **Se o modelo base já for forte:** O AdaBoost funciona melhor com modelos fracos; se o modelo base for complexo, pode não melhorar tanto.

Cenário	Alternativa
Dados ruidosos com muitos outliers	Random Forest ou Extra Trees (menos sensíveis a ruído).
Base de dados muito grande	XGBoost ou LightGBM (implementações mais eficientes de Boosting).
Modelo base já é muito forte	Não usar Boosting, pois ele funciona melhor com modelos fracos.

ADABOOST NO PYTHON

```
from sklearn.ensemble import AdaBoostClassifier
from sklearn.tree import DecisionTreeClassifier

# Criando um classificador base (modelo fraco)
base_estimator = DecisionTreeClassifier (max_depth=1)

# Criando o modelo AdaBoost
ada_boost = AdaBoostClassifier (base_estimator=base_estimator ,
                                n_estimators= 50,
                                random_state= 0)

# Treinando o modelo
ada_boost.fit (X_train, y_train)

# Fazendo previsões no conjunto de teste
y_pred = ada_boost.predict (X_test)
```

HIPERPARÂMETROS DO MODELO

O AdaBoost possui alguns hiperparâmetros importantes que controlam seu desempenho e capacidade de generalização. Os principais são:

```
base_estimator = DecisionTreeClassifier (max_depth=1)
```

Hiperparâmetro	Descrição	Impacto
n_estimators	Número de modelos fracos treinados no ensemble.	Valores altos aumentam a precisão, mas podem levar a overfitting. Valores baixos podem causar underfitting.
learning_rate	Controla o peso dado a cada novo modelo adicionado.	Valores altos aceleram o aprendizado, mas podem superajustar. Valores baixos exigem mais estimadores para alcançar boa performance.
base_estimator	Modelo base usado no Boosting (por padrão, uma árvore rasa).	Árvores muito rasas podem exigir mais iterações. Modelos mais complexos aprendem mais rápido, mas podem overfittar.
algorithm	Define a variação do AdaBoost para classificação (SAMME ou SAMME.R).	"SAMME.R" é mais eficiente e baseado em probabilidades (padrão). "SAMME" é usado para classificação multiclasse.
random_state	Define a semente aleatória para reprodutibilidade dos resultados.	Se definido, garante que os resultados sejam sempre os mesmos ao repetir o treinamento.

CIÊNCIA DE DADOS E INTELIGÊNCIA ARTIFICIAL

RESOLUÇÃO DE PROBLEMAS

EXERCÍCIOS: NÃO COMPARECIMENTO

Imagine que você trabalha na equipe de gestão de um hospital e recebeu um desafio importante: **reduzir o número de pacientes que marcam consultas, mas não comparecem.**

As ausências impactam diretamente o funcionamento da unidade de saúde, gerando desperdício de recursos, aumento no tempo de espera para outros pacientes e dificuldades no planejamento da equipe médica.

Agora, surge a pergunta: **é possível prever quais pacientes têm maior risco de faltar à consulta?**

O conjunto de dados **naocomparecimento.csv**, contém 110.527 registros de consultas médicas, com 14 variáveis associadas a cada paciente e sua consulta. A principal variável a ser analisada indica se o paciente compareceu ou não à consulta.

BASE NÃO COMPARECIMENTO

A variável alvo é "No-show", que indica se o paciente faltou à consulta ("Yes") ou compareceu ("No").

Existem **14 variáveis**, incluindo:

- **Data da consulta**
- **Local do atendimento**
- **Características do paciente** (idade, gênero, presença de doenças crônicas)
- **Se recebeu lembrete via SMS**
- **Se estava em um programa de assistência social (Bolsa Família - Scholarship)**
- **Quantos dias de antecedência a consulta foi marcada**

CONTEXTO

Essa base permite analisar fatores que influenciam a ausência dos pacientes, ajudando unidades de saúde a reduzir faltas e melhorar o atendimento.

Possíveis hipóteses para investigação:

- Pacientes mais jovens ou mais velhos faltam mais?
- O tempo entre o agendamento e a consulta influencia na presença?
- Receber um lembrete por SMS reduz faltas?
- Pacientes com certas condições médicas faltam mais?

ROTEIRO

1. Realize a **análise exploratória univariada** de todas as variáveis no conjunto de dados, interpretando suas distribuições.
2. Faça a **análise bidimensional** entre cada variável explicativa e a variável resposta. Quais variáveis parecem ter maior influência sobre a variável de interesse?
3. Para as **variáveis categóricas**, converta-as para o tipo `category` do pandas.
4. Para este estudo, **exclua** as variáveis **ID**, **Neighbourhood**, e **data** da análise.
5. Construa o melhor **modelo** usando as técnicas de aprendizado de máquinas para prever a variável resposta.
6. Obtenha as **classificação** para cada observação no conjunto de dados.
7. Gere a **tabela de classificação** para avaliar o desempenho do modelo e encontre o melhor ponto de corte para classificar as probabilidades. Qual é o percentual de classificações corretas? E quais são a **precision** e **recall** do modelo?

CIÊNCIA DE DADOS E INTELIGÊNCIA ARTIFICIAL

REFERÊNCIAS



MINISTÉRIO DA
SAÚDE



REFERÊNCIAS

- Anderson, R. A., Sweeney, J. D. e Williams, T. A. Estatística Aplicada à Administração e Economia. Editora Cengage. 4ª edição, 2019.
- VanderPlas, J. [Python Data Science Handbook](#) - O'Reilly Media, Inc.
- McKinney, W. Python for Data Analysis - Data Wrangling with Pandas, NumPy & Jupyter. 3rd Edition - Open Access: <https://wesmckinney.com/book/>
- Google Colab - <https://colab.research.google.com/>
- Scipy - <https://scipy.org/>
- Statsmodels - <https://www.statsmodels.org/>
- Scikit-learn - <https://scikit-learn.org/>



CIÊNCIA DE DADOS E INTELIGÊNCIA ARTIFICIAL



MINISTÉRIO DA
SAÚDE

