

**UNIVERSIDADE DE SÃO PAULO**

Instituto de Ciências Matemáticas e de Computação

## **MBA em Inteligência Artificial e Big Data**

**Antonio Lopes**

Resumo das Aulas do Curso de MBA em Inteligência Artificial e Big Data

## SUMÁRIO

<b>I</b>	<b>CURSO 2 - CIÊNCIA DE DADOS, APRENDIZADO DE MÁQUINA E MINERAÇÃO DE DADOS</b>	<b>5</b>
<b>1</b>	<b>NAIVE BAYES</b>	<b>7</b>
1.1	Para que é utilizado?	7
1.2	A Base Teórica: O Teorema de Bayes	7
1.3	O "Naive"(Ingênuo) e a Fórmula para Múltiplas Características	8
1.4	Tipos de Naive Bayes	8
1.5	Exemplo Prático: Filtro de Spam	9
1.6	Exemplo em Python	9
1.7	Pontos Positivos (Vantagens)	11
1.8	Pontos Negativos (Desvantagens e Cuidados)	11
1.9	Conclusão	12
<b>2</b>	<b>ÁRVORE DE DECISÃO</b>	<b>13</b>
2.1	Para que é utilizado?	13
2.2	O Conceito Central: Como a Árvore "Aprende"?	13
2.2.1	Entropia	14
2.2.2	Ganho de Informação	14
2.2.3	Índice Gini	14
2.3	Exemplo Prático: Jogar Tênis?	15
2.4	Exemplo em Python	15
2.5	Pontos Positivos (Vantagens)	16
2.6	Pontos Negativos (Desvantagens)	17
2.7	Como Mitigar as Desvantagens no Contexto de Big Data?	17
2.8	Conclusão	17
<b>3</b>	<b>AVALIAÇÃO DE CLASSIFICADORES</b>	<b>19</b>
3.1	Para que é utilizado?	19
3.2	Conceitos e Métricas Fundamentais (Com Fórmulas)	19
3.2.1	Matriz de Confusão (Confusion Matrix)	20
3.2.2	Acurácia (Accuracy)	20
3.2.3	Precisão (Precision)	20
3.2.4	Revocação (Recall ou Sensibilidade)	21
3.2.5	Pontuação F1 (F1-Score)	21
3.2.6	Curva ROC e AUC (Área Sob a Curva)	21
<b>3.3</b>	<b>Exemplo Prático</b>	<b>22</b>

3.4	Exemplo em Python . . . . .	22
3.5	Pontos Positivos (Por que é crucial?) . . . . .	24
3.6	Pontos Negativos (Cuidados) . . . . .	24
3.7	Conclusão . . . . .	25
4	RANDOM FOREST . . . . .	27
4.1	Para que é utilizado? . . . . .	27
4.2	O Algoritmo: Como Funciona? (Conceito e Pseudocódigo) . . . . .	28
4.3	Exemplo Prático: Prever se uma pessoa gosta de filme de ação . . . . .	29
4.4	Exmplo em Python . . . . .	29



## **Parte I**

**Curso 2 - Ciência de Dados, Aprendizado de Máquina e Mineração de Dados**



## 1 NAIVE BAYES

O Naive Bayes é um algoritmo de aprendizado de máquina supervisionado usado principalmente para problemas de classificação. Ele é baseado no Teorema de Bayes da teoria da probabilidade, com uma suposição "ingênua"(naive): a de que as características (features) usadas para prever uma classe são independentes umas das outras, dado o valor da classe.

Embora essa suposição de independência raramente seja verdadeira no mundo real (daí o termo "ingênuo"), o algoritmo surpreendentemente funciona muito bem em uma vasta gama de problemas, especialmente em processamento de linguagem natural (NLP) e mineração de texto.

### 1.1 Para que é utilizado?

O Naive Bayes é incrivelmente versátil e rápido, sendo aplicado em:

1. Filtro de Spam: Seu uso mais famoso. Classifica e-mails como "spam" ou "não spam" com base nas palavras que contêm.
2. Análise de Sentimento: Classifica textos (como reviews, tweets) como positivos, negativos ou neutros.
3. Classificação de Documentos: Categoriza artigos de notícias em temas como "esportes", "política", "tecnologia", etc.
4. Sistemas de Recomendação: Pode ser usado para recomendar produtos com base em características simples.
5. Diagnóstico Médico: Auxilia no diagnóstico de doenças com base em sintomas (embora com cautela, devido à suposição de independência).

### 1.2 A Base Teórica: O Teorema de Bayes

A fórmula central do algoritmo é uma aplicação direta do Teorema de Bayes:

$$P(A|B) = \frac{P(B|A) * P(A)}{P(B)}$$

Onde:

- $P(A|B)$  é a probabilidade posterior. É o que queremos descobrir: a probabilidade da classe A (ex: "spam") dado que as características B (ex: palavras "oferta", "grátis") ocorreram.

- $P(B|A)$  é a verossimilhança (likelihood). A probabilidade de observar as características  $B$  dado que a classe é  $A$ .
- $P(A)$  é a probabilidade anterior (prior). A probabilidade inicial da classe  $A$  (ex: a proporção de e-mails que são spam na sua base de treino).
- $P(B)$  é a probabilidade marginal (evidence). A probabilidade geral de observar as características  $B$ . Age como uma constante de normalização.

Na prática, para classificação, calculamos a probabilidade posterior para cada classe possível e escolhemos a classe com a maior probabilidade.

### 1.3 O "Naive"(Ingênuo) e a Fórmula para Múltiplas Características

A suposição "ingênua" de independência permite que simplifiquemos drasticamente o cálculo. Se temos várias características  $(B_1, B_2, B_3, \dots, B_n)$ , a probabilidade  $P(B|A)$  se torna o produto das probabilidades de cada característica individual:

$$P(B_1, B_2, \dots, B_n|A) \simeq P(B_1|A) * P(B_2|A) * \dots * P(B_n|A)$$

Portanto, a fórmula do Naive Bayes para uma classe  $A$  fica:

$$P(A|B_1, B_2, \dots, B_n) \simeq \frac{P(A)*P(B_1|A)*P(B_2|A)*\dots*P(B_n|A)}{P(B)}$$

Como  $P(B)$  é igual para todas as classes, podemos ignorá-lo para comparação e simplesmente encontrar a classe  $A$  que maximiza o numerador:

$$ClassePreditada = \underset{A}{\operatorname{argmax}} [P(A) * \prod_{i=1}^n P(B_i|A)] \text{ (para } i \text{ de } 1 \text{ até } n \text{ características)}$$

### 1.4 Tipos de Naive Bayes

A maneira de calcular  $P(B_i|A)$  (a probabilidade de uma característica dada uma classe) pode variar, dando origem a diferentes variantes:

1. **Gaussian Naive Bayes:** Assume que os valores das características contínuas seguem uma distribuição normal (gaussiana). Usado quando os dados são numéricos.
2. **Multinomial Naive Bayes:** Usa uma distribuição multinomial para modelar a contagem de frequência de palavras. É a variante mais comum para classificação de texto.
3. **Bernoulli Naive Bayes:** Projetado para características binárias (ex: 0 ou 1, verdadeiro ou falso). Assume que as características são geradas por uma distribuição de Bernoulli (lançamento de moeda).



## 1.5 Exemplo Prático: Filtro de Spam

Vamos imaginar um dataset minúsculo de 10 e-mails para treinar nosso modelo.

### Passo 1: Calcular os Priors $P(A)$

- 6 e-mails são spam  $\rightarrow P(\text{Spam}) = 6/10 = 0.6$
- 4 e-mails não são spam (ham)  $\rightarrow P(\text{Ham}) = 4/10 = 0.4$

### Passo 2: Calcular as Likelihoods $P(\text{Palavra}|\text{Classe})$ . Vamos analisar a palavra "oferta".

- A palavra "oferta" aparece em 4 e-mails spam.
- A palavra "oferta" aparece em 1 e-mail ham.
- $P(\text{oferta}|\text{Spam}) = 4/6 \simeq 0.666$
- $P(\text{oferta}|\text{Ham}) = 1/4 = 0.25$

### Passo 3: Fazer uma Previsão (Classificação)

Suponha que recebemos um NOVO e-mail que contém a palavra "oferta".

É spam ou ham?

Calculamos a probabilidade para cada classe (ignorando o denominador  $P(\text{oferta})$ ):

1.  $P(\text{Spam}|\text{oferta}) \propto P(\text{Spam}) * P(\text{oferta}|\text{Spam}) = 0.6 * 0.666 \simeq 0.4$
2.  $P(\text{Ham}|\text{oferta}) \propto P(\text{Ham}) * P(\text{oferta}|\text{Ham}) = 0.4 * 0.25 = 0.1$

Como  $0.4 (\text{Spam}) > 0.1 (\text{Ham})$ , o modelo classifica este novo e-mail como SPAM.

Na realidade, usamos dezenas de milhares de palavras, e o cálculo é o produto de todas as probabilidades.

## 1.6 Exemplo em Python

```

1 # Exemplo de Naive Bayes para classificar mensagens (spam ou nao spam
  )
2
3 from sklearn.model_selection import train_test_split
4 from sklearn.feature_extraction.text import CountVectorizer
5 from sklearn.naive_bayes import MultinomialNB
6 from sklearn.metrics import accuracy_score, classification_report
7

```

```
8 # Dataset simples de mensagens
9 mensagens = [
10     "Promocao imperdível, compre ja!",
11     "Oferta exclusiva para voce",
12     "Nosso encontro esta confirmado amanha",
13     "Lembrete da reuniao as 10h",
14     "Ganhe dinheiro facil e rapido",
15     "Voce foi selecionado para um premio"
16 ]
17
18 # Classes (Spam = 1, Nao Spam = 0)
19 rotulos = [1, 1, 0, 0, 1, 1]
20
21 # 1. Transformar texto em vetores de contagem (Bag of Words)
22 vectorizer = CountVectorizer()
23 X = vectorizer.fit_transform(mensagens)
24
25 # 2. Dividir em treino e teste
26 X_train, X_test, y_train, y_test = train_test_split(X, rotulos,
27     test_size=0.3, random_state=42)
28
29 # 3. Criar e treinar o classificador
30 modelo = MultinomialNB()
31 modelo.fit(X_train, y_train)
32
33 # 4. Prever no conjunto de teste
34 y_pred = modelo.predict(X_test)
35
36 # 5. Avaliacao
37 print("Acuracia:", accuracy_score(y_test, y_pred))
38 print("Relatorio de classificacao:\n", classification_report(y_test,
39     y_pred))
40
41 # 6. Testar uma nova mensagem
42 nova_msg = ["Promocao exclusiva so hoje"]
43 nova_msg_vec = vectorizer.transform(nova_msg)
44 print("Nova mensagem classificada como:", modelo.predict(nova_msg_vec
45     ))
```

### **O que acontece nesse código:**

1. Transformação do texto → usamos CountVectorizer para converter palavras em

vetores de frequência (Bag of Words).

2. Treinamento → MultinomialNB aprende as probabilidades a partir dos dados.
3. Avaliação → usamos `accuracy_score` e `classification_report` para medir a performance.
4. Previsão → o modelo classifica uma nova mensagem como Spam (1) ou Não Spam (0).

### 1.7 Pontos Positivos (Vantagens)

- **Extremamente Rápido:** Tanto para treinar quanto para prever, pois são apenas cálculos de probabilidade. É ideal para Big Data onde o volume de dados é enorme.
- **Simple e Intuitivo:** Fácil de entender e implementar.
- **Funciona Bem com Poucos Dados:** Dá bons resultados mesmo com conjuntos de treinamento relativamente pequenos.
- **Lida Bem com Alta Dimensionalidade:** Desempenho muito bom mesmo quando existem milhares de características (como em processamento de texto).
- **Robusto a Ruídos:** Dados irrelevantes ou ruidosos tendem a ser "cancelados" na multiplicação das probabilidades.

### 1.8 Pontos Negativos (Desvantagens e Cuidados)

- **Suposição de Independência "Ingênua":** Esta é sua maior fraqueza. No mundo real, as características costumam ser correlacionadas (ex: a palavra "futebol" e a palavra "gol" não são independentes). Isso pode levar a estimativas de probabilidade imprecisas.
- **Problema de Frequência Zero (Zero-Frequency):** Se uma categoria de característica não foi observada no conjunto de treinamento, sua probabilidade se torna zero ( $P(B_i|A) = 0$ ). Como estamos multiplicando probabilidades, um único zero "zera" toda a previsão. Isso é resolvido usando Suavização de Laplace (adicionar 1 a todas as contagens).
- **Previsões Probabilísticas Não Confiáveis:** Embora seja ótimo para classificação (escolher uma categoria), as probabilidades brutas que ele gera ( $P(Classe|Características)$ ) costumam ser mal calibradas e não devem ser tomadas como valores absolutos de confiança.

## 1.9 Conclusão

O Naive Bayes é um algoritmo poderoso, eficiente e surpreendentemente eficaz, apesar de sua suposição simplificadora. Ele é uma ferramenta excelente para ser usada como baseline (linha de base) em qualquer projeto de classificação, especialmente com dados textuais. Sua velocidade o torna uma escolha primordial para aplicações em tempo real e para lidar com os vastos volumes de dados do universo do Big Data.

Se você tem um problema de classificação, comece com o Naive Bayes. Ele lhe dará rapidamente uma boa ideia do desempenho que você pode esperar e, a partir daí, você pode experimentar algoritmos mais complexos como Random Forest ou XGBoost para tentar superá-lo.

## 2 ÁRVORE DE DECISÃO

Uma Árvore de Decisão é um algoritmo de aprendizado supervisionado usado para classificação e regressão. Sua estrutura mimics a forma como os humanos tomam decisões naturalmente, através de uma sequência de perguntas sim/não (ou condições) que levam a uma conclusão.

A estrutura do algoritmo é uma árvore invertida, composta por:

- **Nó Raiz (Root Node):** Representa toda a população ou amostra de dados. É o ponto de partida, onde é feita a primeira pergunta/divisão.
- **Nós de Decisão (Internal Nodes):** Representam uma pergunta sobre uma característica (feature) e a divisão (ramificação) dos dados.
- **Folhas (Leaf Nodes):** São os nós finais que contêm a resposta ou a decisão (a classe predita em classificação, ou um valor contínuo em regressão).

### 2.1 Para que é utilizado?

As Árvores de Decisão são incrivelmente versáteis:

#### 1. **Classificação:** Prever uma categoria.

- Exemplo: Um banco decidir se concede ou não um empréstimo com base em idade, salário e histórico de crédito.

#### 2. **Regressão:** Prever um valor numérico.

- Exemplo: Prever o preço de uma casa com base em seu tamanho, número de quartos e localização.

#### 3. **Engenharia de Features:** Identificar as características mais importantes para um problema.

#### 4. **Base para Algoritmos Ensemble:** Árvores simples são os "tijolos" que constroem algoritmos campeões como Random Forest e Gradient Boosting (XGBoost, LightGBM), amplamente usados em Big Data.

### 2.2 O Conceito Central: Como a Árvore "Aprende"?

O algoritmo aprende dividindo o conjunto de treinamento de forma recursiva, escolhendo a melhor pergunta (a melhor característica e o melhor ponto de corte) a

ser feita em cada nó. O objetivo é criar subconjuntos (ramos) que sejam cada vez mais puros – ou seja, que contenham predominantemente exemplos de uma única classe.

Para medir a "impureza" de um nó e encontrar a melhor divisão, são usadas métricas. As principais fórmulas são Entropia, Ganho de Informação (Information Gain) e Índice de Gini.

### 2.2.1 Entropia

Mede o grau de desordem ou impureza de um conjunto de dados. A entropia é máxima (1.0) quando as classes estão perfeitamente misturadas (50% de cada) e zero (0.0) quando o nó é perfeitamente puro.

#### **Fórmula da Entropia:**

$$Entropia(S) = \sum_{i=1}^c p_i \log_2(p_i)$$

- $S$ : O conjunto de dados no nó.
- $p_i$ : A proporção da classe  $i$  no nó.

### 2.2.2 Ganho de Informação

Mede a redução da entropia que uma divisão proporciona. O algoritmo escolhe a característica e o ponto de corte que maximizam o Ganho de Informação.

#### **Fórmula do Ganho de Informação:**

$$Ganho(S, A) = Entropia(S) - \sum_{v \in Valores(A)} \frac{|S_v|}{|S|} Entropia(S_v)$$

- $A$ : A característica pela qual estamos dividindo.
- $S_v$ : O subconjunto de dados onde a característica  $A$  tem o valor  $v$ .

### 2.2.3 Índice Gini

Uma alternativa muito comum à Entropia. Mede a impureza de forma semelhante, mas é computacionalmente mais eficiente.

#### **Fórmula do Índice Gini:**

$$Gini(S) = 1 - \sum_{i=1}^c p_i^2$$

- Um índice Gini de 0 significa pureza perfeita.

O algoritmo escolhe a divisão que minimiza o índice Gini médio dos nós filhos. Na prática, o resultado final (Gini vs. Entropia) é frequentemente muito similar.

## 2.3 Exemplo Prático: Jogar Tênis?

Vamos usar um exemplo clássico. Queremos prever se uma pessoa vai jogar tênis com base nas condições do tempo.

Tempo	Temperatura	Umidade	Vento	Jogar?
Sunny	Hot	High	Weak	No
Sunny	Hot	High	Strong	No
Overcast	Hot	High	Weak	Yes
Rainy	Mild	High	Weak	Yes
Rainy	Cool	Normal	Weak	Yes
Rainy	Cool	Normal	Strong	No
Overcast	Cool	Normal	Strong	Yes
...	...	...	...	...

O algoritmo começa no nó raiz com todos os exemplos. Ele calcula a impureza (ex: Entropia) para o estado inicial.

Em seguida, ele testa cada característica (Outlook, Temperature, etc.) para ver qual delas, ao dividir os dados, proporciona o maior Ganho de Informação.

1. Suponha que Outlook seja a característica com maior ganho. Ela se torna o nó raiz, com ramos para Sunny, Overcast, e Rainy.
2. O ramo Overcast já é puro (todos os exemplos são "Yes"). Ele se torna uma folha com a decisão "Yes".
3. O ramo Sunny ainda é impuro. O algoritmo repete o processo apenas para os exemplos onde Outlook = Sunny, buscando a próxima melhor característica para dividi-los (ex: Humidity). Esse processo continua até que todos os ramos terminem em folhas puras ou até que um critério de parada seja atingido.

## 2.4 Exemplo em Python

```

1 # Importando bibliotecas
2 from sklearn.datasets import load_iris
3 from sklearn.model_selection import train_test_split
4 from sklearn.tree import DecisionTreeClassifier, plot_tree
5 import matplotlib.pyplot as plt
6
7 # 1. Carregar dataset Iris
8 iris = load_iris()
9 X = iris.data # atributos (comprimento e largura de sepalas e
    petalas)
10 y = iris.target # classes (Setosa, Versicolor, Virginica)

```

```
11
12 # 2. Dividir em treino e teste
13 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size
    =0.3, random_state=42)
14
15 # 3. Criar e treinar Arvore de Decisao
16 clf = DecisionTreeClassifier(criterion="entropy", max_depth=3,
    random_state=42)
17 clf.fit(X_train, y_train)
18
19 # 4. Avaliar acuracia
20 accuracy = clf.score(X_test, y_test)
21 print(f"Acuracia no teste: {accuracy:.2f}")
22
23 # 5. Visualizar a Arvore
24 plt.figure(figsize=(12,8))
25 plot_tree(clf, feature_names=iris.feature_names,
26 class_names=iris.target_names,
27 filled=True, rounded=True)
28 plt.show()
```

### O que acontece nesse código:

1. Carrega o dataset Iris.
2. Divide em treino (70%) e teste (30%).
3. Cria uma árvore de decisão com critério de entropia e profundidade máxima 3.
4. Calcula a acurácia no conjunto de teste.
5. Desenha a árvore de decisão.

O gráfico gerado vai mostrar as divisões baseadas nas features (ex.: comprimento da pétala  $\leq 2.45 \rightarrow$  Setosa).

## 2.5 Pontos Positivos (Vantagens)

- **Extremamente Intuitivo e Interpretável:** A lógica da árvore pode ser facilmente explicada e visualizada, mesmo para não-especialistas. Isso é crucial em áreas como medicina e finança ("caixa preta transparente").
- **Não Requer Pré-processamento Complexo:** Lida bem com dados numéricos e categóricos e não exige normalização ou padronização dos dados.



- **Lida Bem com Características Irrelevantes:** O processo de seleção de divisões naturalmente identifica e usa as features mais importantes, ignorando as menos relevantes.
- **Versátil:** Pode resolver problemas de classificação e regressão.

## 2.6 Pontos Negativos (Desvantagens)

- **Propenso a Overfitting (Super-ajuste):** Árvores muito profundas e complexas memorizam o ruído e os detalhes do conjunto de treinamento, perdendo a capacidade de generalizar para dados novos. Isso é um grande problema.
- **Instabilidade:** Pequenas mudanças nos dados de treinamento podem resultar em uma árvore de estrutura completamente diferente.
- **Viés para Classes Dominantes:** Em conjuntos desbalanceados, a árvore pode ficar enviesada para a classe com mais exemplos.
- **Dificuldade com Relações Não-Lineares Complexas:** Embora capturem não-linearidades, podem não ser a melhor escolha para problemas extremamente complexos se usadas sozinhas.

## 2.7 Como Mitigar as Desvantagens no Contexto de Big Data?

As desvantagens principais são resolvidas usando Técnicas de Ensemble (Conjunto):

- **Floresta Aleatória (Random Forest):** Constrói centenas de árvores, cada uma treinada em um subconjunto aleatório dos dados e features. O resultado final é uma "votação" entre todas as árvores. Isso reduz drasticamente o overfitting e a instabilidade.
- **Gradient Boosting (XGBoost, LightGBM, CatBoost):** Constrói árvores sequencialmente, onde cada nova árvore tenta corrigir os erros da árvore anterior. São algoritmos extremamente poderosos e dominantes em competições de machine learning.

## 2.8 Conclusão

A Árvore de Decisão é um algoritmo fundamental, servindo tanto como uma ferramenta poderosa por si só quanto como a base para os métodos de ensemble mais sofisticados que dominam o cenário atual de Big Data e Aprendizado de Máquina. Sua simplicidade e interpretabilidade a tornam uma escolha valiosa para explorar dados,

criar baselines e para aplicações onde entender a decisão do modelo é tão importante quanto a sua precisão.

### 3 AVALIAÇÃO DE CLASSIFICADORES

É o processo de medir o desempenho e a qualidade de um modelo de Machine Learning treinado para tarefas de classificação. Não basta apenas treinar o modelo, é fundamental quantificar o quão bem ele generaliza para dados não vistos durante o treinamento. Isso nos permite:

- Comparar diferentes modelos (ex: Naive Bayes vs. Árvore de Decisão vs. Random Forest) de forma objetiva.
- Ajustar hiperparâmetros (tuning) para melhorar o modelo.
- Garantir que o modelo atenda aos requisitos do negócio antes de colocá-lo em produção.
- Identificar vieses e problemas específicos, como desempenho ruim em uma classe particular.

No contexto de Big Data, onde os modelos são complexos e os impactos das decisões automatizadas são grandes, a avaliação rigorosa é não apenas uma boa prática, mas uma necessidade.

#### 3.1 Para que é utilizado?

A avaliação é utilizada em absolutamente todo projeto de classificação, como:

- **Sistemas de Recomendações:** Avaliar se as recomendações são relevantes.
- **Diagnóstico Médico:** Medir a precisão de um modelo em detectar uma doença (é crucial aqui evitar falsos negativos).
- **Deteção de Fraude:** Avaliar a capacidade de identificar transações fraudulentas sem incomodar clientes legítimos com alarmes falsos (falsos positivos).
- **Análise de Sentimento:** Verificar se a classificação de textos como positivo/negativo/neutro está correta.

#### 3.2 Conceitos e Métricas Fundamentais (Com Fórmulas)

A avaliação começa com a **Matriz de Confusão**, uma tabela que resume as previsões do modelo versus os valores reais.

### 3.2.1 Matriz de Confusão (Confusion Matrix)

Imagine um problema binário (classe positiva: 1, classe negativa: 0).

	<b>Previsão Negativo (0)</b>	<b>Previsão Positivo (1)</b>
<b>Real Negativo (0)</b>	Verdadeiro Negativo (TN)	Falso Positivo (FP)
<b>Real Positivo(1)</b>	Falso Negativo (FN)	Verdadeiro Positivo (TP)

- **Verdadeiro Positivo (TP):** O modelo previu positivo e acerto.
- **Falso Negativo (FP):** O modelo previu positivo, mas errou (o real era negativo). Também chamado de **Erro Tipo I**.
- **Falso Negativo (FN):** O modelo previu negativo, mas errou (o real era positivo). Também chamado de **Erro Tipo II**.
- **Verdadeiro Negativo (TN):** O modelo previu negativo e acerto.

Todas as métricas principais derivam desses quatro valores.

### 3.2.2 Acurácia (Accuracy)

É a métrica mais intuitiva: a proporção de previsões totais que o modelo acertou.

**Fórmula:**

$$Accuracy = \frac{TP+TN}{TP+TN+FP+FN}$$

**Ponto Negativo:** Pode ser enganosa em conjuntos de dados desbalanceados. Por exemplo, se 99% dos dados são da classe 0, um modelo que sempre prevê 0 terá 99% de acurácia, mas é inútil.

### 3.2.3 Precisão (Precision)

Dentre todas as previsões positivas que o modelo fez, quantas eram realmente positivas? Responde: "Quando o modelo diz que é positiva, qual a chance de estar certo?".

**Fórmula:**

$$Precision = \frac{TP}{TP+FP}$$

É crucial quando o custo dos Falso Positivos (FP) é alto.

- Exemplo (Spam): Classificar um e-mail importante como spam (FP) é muito ruim. Queremos alta precisão.

### 3.2.4 Revocação (Recall ou Sensibilidade)

Dentre todos os casos realmente positivos, quantos o modelo conseguiu identificar? Responde: "O modelo está conseguindo encontrar todos os positivos existentes?".

**Fórmula:**

$$Recall = \frac{TP}{TP+FN}$$

É crucial quando o custo dos Falso Negativos (FN) é alto.

- **Exemplo (Diagnóstico de Câncer):** Deixar de identificar um paciente doente (FN) é um erro gravíssimo. Queremos alta Revocação.

### 3.2.5 Pontuação F1 (F1-Score)

É a média harmônica entre a Precisão e Revocação. Ela busca um equilíbrio entre as duas métricas, sendo muito útil quando precisamos de um único número para avaliar o modelo, especialmente em cenários desbalanceados.

**Fórmula:**

$$F_1 = 2 \times \frac{Precision \times Recall}{Precision + Recall}$$

Um F1-Score alto indica que o modelo tem boa Precisão e boa Revocação. Um F1-Score baixo sinaliza que uma das duas métricas está muito ruim.

### 3.2.6 Curva ROC e AUC (Área Sob a Curva)

Essa é uma métrica mais avançada e extremamente importante.

- **ROC Curve:** Gráfico que mostra o desempenho de um modelo classificador em todos os limiares de classificação. Ele plota a Taxa de Verdadeiros Positivos (Recall) vs. a Taxa de Falsos Positivos ( $FPR = \frac{FP}{FP+TN}$ ).
- **AUC (Area Under the Curve):** A área sob a curva ROC. Resume a curva em um único valor.

**Interpretações da AUC:**

- **AUC = 1.0:** Classificador perfeito.
- **AUC = 0.5):** Classificador que não é melhor que um chute aleatório. Uma linha diagonal.
- **AUC entre 0.5 e 1.0:** Quanto maior o valor, melhor o modelo é em distinguir entre as classes positivas e negativa.

A AUC é excelente porque é insensível a desbalanceamento de classe e ao limiar de classificação escolhido.

### 3.3 Exemplo Prático

Vamos avaliar um modelo que detecta fraudes em cartão de crédito em um conjunto de dados altamente desbalanceado (99% legítimas, 1% fraudulentas).

#### Matriz de Confusão (Em uma amostra de 10.000 transações)

Real / Previsto	Não Fraude	Fraude
Não Fraude	9.850	50
Fraude	20	80

- **TN = 9.850, FP = 50, FN = 20, TP = 80**

#### Cálculo das Métricas:

- **Acurácia:**  $(9.850 + 80) / 10.000 = 99.3\%$  (Parece ótimo, mas é enganoso!).
- **Precisão:**  $80 / (80 + 50) = 61.5\%$  (Dos alertas de fraude, apenas  $\simeq 62\%$  eram verdadeiros. Muitos falsos alarmes).
- **Revocação:**  $80 / (80 + 20) = 80.0\%$  (O modelo capturou 80% de todas as fraudes existentes).
- **F1-Score:**  $2 * (0.615 * 0.80) / (0.615 + 0.80) = \sim 0.695$

#### Análise:

A Acurácia esconde o problema. A Revocação é aceitável (encontramos a maioria das fraudes), mas a Precisão é baixa (muitos clientes legítimos estão sendo incomodados). O F1-Score de 0.695 mostra que há um trade-off a ser gerido. Talvez seja necessário ajustar o limiar do modelo para aumentar a Precisão, mesmo que sacrifique um pouco da Revocação, dependendo da estratégia de negócio.

### 3.4 Exemplo em Python

```

1 # Importando bibliotecas
2 from sklearn.datasets import load_breast_cancer
3 from sklearn.model_selection import train_test_split
4 from sklearn.tree import DecisionTreeClassifier
5 from sklearn.metrics import (
6 confusion_matrix, classification_report, accuracy_score,
```

```
7 roc_curve, roc_auc_score
8 )
9 import matplotlib.pyplot as plt
10 import seaborn as sns
11
12 # 1. Carregar dataset
13 data = load_breast_cancer()
14 X, y = data.data, data.target
15
16 # 2. Dividir em treino e teste
17 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size
    =0.3, random_state=42)
18
19 # 3. Treinar um classificador simples (Arvore de Decisao)
20 clf = DecisionTreeClassifier(max_depth=4, random_state=42)
21 clf.fit(X_train, y_train)
22
23 # 4. Fazer previsoes
24 y_pred = clf.predict(X_test)
25 y_prob = clf.predict_proba(X_test)[:,1] # Probabilidade para
    cÃ;lculo de AUC-ROC
26
27 # 5. Matriz de Confusao
28 cm = confusion_matrix(y_test, y_pred)
29
30 plt.figure(figsize=(6,5))
31 sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", xticklabels=data.
    target_names, yticklabels=data.target_names)
32 plt.xlabel("Previsto")
33 plt.ylabel("Real")
34 plt.title("Matriz de Confusao")
35 plt.show()
36
37 # 6. Metricas principais
38 print("Acuracia:", accuracy_score(y_test, y_pred))
39 print("\nRelatorio de Classificacao:")
40 print(classification_report(y_test, y_pred, target_names=data.
    target_names))
41
42 # 7. Curva ROC e AUC
43 fpr, tpr, thresholds = roc_curve(y_test, y_prob)
44 auc = roc_auc_score(y_test, y_prob)
```

```
45
46 plt.figure(figsize=(6,5))
47 plt.plot(fpr, tpr, label=f"AUC = {auc:.2f}")
48 plt.plot([0,1], [0,1], linestyle="--", color="gray")
49 plt.xlabel("Falso Positivo (FPR)")
50 plt.ylabel("Verdadeiro Positivo (TPR)")
51 plt.title("Curva ROC")
52 plt.legend()
53 plt.show()
```

### O que o código faz:

1. Carrega o dataset de câncer de mama (classes: maligno vs benigno).
2. Divide em treino (70%) e teste (30%).
3. Treina a Árvore de Decisão.
4. Faz previsões.
5. Gera Matriz de Confusão com visualização.
6. Calcula Acurácia, Precisão, Recall e F1-Score.
7. Plota a Curva ROC e calcula a AUC.

### 3.5 Pontos Positivos (Por que é crucial?)

- **Fornece uma Visão Holística:** Vai muito além da acurácia, revelando os trade-offs reais do modelo (Precisão vs Revocação).
- **Permite Tomada de Decisão Informada:** A escolha da métrica ideal é guiada pelo problema de negócio (e.g. "prefiro falsos alarmes ou deixar fraude passar?").
- **Fundamenta para Comparação:** É a única maneira objetiva de dizer se o modelo A é melhor que o modelo B.
- **Identifica Vieses:** Métricas calculadas por classes (em problemas multiclasse) mostram se o modelo performa mal em um grupo específico.

### 3.6 Pontos Negativos (Cuidados)

- **Métrica Inadequada:** Escolher a métrica errada pode levar a decisões ruins. Focar apenas na acurácia é o erro mais comum.



- **Vazamento de Dados (Data Leakage):** Se a métrica for calculada em dados que foram usados no treinamento ou que contêm informações do "futuro", ela será otimista e irremediavelmente enviesada. A avaliação deve ser sempre feita em conjunto de test holdout totalmente isolado ou com validação cruzada.
- **Não Captura Custos de Negócio:** Uma métrica como F1-Score é estatística. Ela não captura diretamente que um Falso Negativo pode custar \$1.000 e um Falso Positivo \$10. Em alguns casos, é necessário construir uma função de custo customizada.
- **Despesa Computacional:** Em Big Data, calcular métricas como AUC e validação cruzada em grandes volumes de dados pode ser computacionalmente intensivo.

### 3.7 Conclusão

A Avaliação de Classificadores é a bússula que guia o desenvolvimento de modelos de ML. Sem ela, estamos navegando às cegas. Dominar conceitos como Matriz de Confusão, Precisão, Revocação, F1-Score e AUC é essencial para qualquer profissional da área, permitindo a construção de modelos não apenas precisos, mas robustos, justos e alinhados com os objetivos de negócio. Em Big Data, onde os stakes são altos, essa prática é não negociável.



## 4 RANDOM FOREST

O Random Forest é um algoritmo de aprendizado supervisionado usado para classificação e regressão. Ele pertence à categoria de método ensemble (conjunto), especificamente ao tipo bagging.

A ideia central por trás do Random Forest é simples, mas brilhante:

- Em vez de confiar em uma única Árvore de Decisão, treine centenas ou milhares delas e combine suas previsões para obter um resultado mais preciso e estável.

O "Aleatório" no nome vem de duas fontes de aleatoriedade introduzidas durante o treinamento de cada árvore:

1. **Bagging (Bootstrap Aggregating):** Cada árvore é treinada em um subconjunto diferente dos dados de treinamento, criado através de amostragem com reposição.
2. **Seleção Aleatória de Features:** Ao buscar a melhor feature para fazer uma divisão em um nó, o algoritmo só considera um subconjunto aleatório de todas as features disponíveis.

Essas duas técnicas combatem diretamente o principal problema das Árvores de Decisão: o **overfitting**.

### 4.1 Para que é utilizado?

O Random Forest é um algoritmo extremamente versátil, aplicado em uma vasta gama de problemas:

#### 1. Problemas de Classificação:

- **Diagnóstico Médico:** Classificar se um paciente tem ou não uma doença com base em sintomas e exames.
- **Deteção de Fraude:** Identificar transações financeiras fraudulentas.
- **Marketing:** Prever se um cliente irá ou não cancelar um serviço (churn).
- **Classificação de Imagens:** Identificar objetos em imagens (embora redes neurais convolucionais sejam mais comuns para isso hoje).

#### 2. Problemas de Regressão:

- **Previsão de Demanda:** Prever as vendas de um produto.
  - **Precificação de Ativos:** Estimar o valor de imóveis ou outros bens.
3. **Seleção de Features:** Por sua natureza, o Random Forest pode ranquear a importância de cada variável para a previsão, sendo um ferramenta valiosa para feature engineering.
  4. **Big Data:** Sua natureza paralelizável o torna excelente para ser executado em clusters de computadores (usando frameworks como Spark), lidando eficientemente com grandes volumes de dados.

#### 4.2 O Algoritmo: Como Funciona? (Conceito e Pseudocódigo)

O treinamento de um Random Forest segue os seguintes passos:

##### 1. Criação de Múltiplos Conjuntos de Dados (Bootstrapping):

- Para cada árvore  $t$  na floresta (de  $T$  árvores), crie um novo conjunto de treinamento  $D_t$  amostrando  $N$  exemplos do conjunto original de treinamento com reposição. Isso significa que alguns exemplos serão repetidos e outros não serão relacionados (estes são chamados de exemplos "out-of-bag- OOB).

##### 2. Treinamento de Árvores com Aleatoriedade:

- Para cada árvore  $t$ , treine uma Árvore de Decisão completa no conjunto  $D_t$ .
- **Critério de Aleatoriedade:** Em cada nó de cada árvore, ao buscar a melhor feature para dividir os dados, considere apenas um subconjunto de  $m$  features de um total de  $M$  features. Um valor comum é  $m = \sqrt{M}$  para classificação.
- A árvore é crescida até seu tamanho máximo, sem poda.

##### 3. Combinação das Previsões (Aggregation):

- **Classificação:** A previsão final da floresta é a classe que recebeu a maioria dos votos (moda) de todas as árvores.
- **Regressão:** A previsão final é a média das previsões de todas as árvores.

##### Fórmula para Regressão:

$$\hat{y} = \frac{1}{T} \sum_{t=1}^T \hat{y}_t \text{ onde } \hat{y}_t \text{ é a previsão da árvore } t.$$

### 4.3 Exemplo Prático: Prever se uma pessoa gosta de filme de ação

Suponha que teos dados sobre pessoas (idade, gênero, profissão) e se elas gostam de filmes de ação.

1. **Bootstrapping:** O algoritmo cria 500 conjuntos de dados diferentes (cada um com  $\sim 63\%$  dos dados originais).
2. **Treinamento Aleatório:** Para a Árvore 1, o nó raiz pode ser forçado a escolher apenas entre as features "idade" e "profissão" (em vez de todas as features). Ela escolhe "idade < 25" e se divide.
3. **Continua:** A próxima divisão na Árvore pode considerar "gênero" e "profissão", e assim por diante. A Árvore 2 será treinada em um conjunto de dados ligeiramente diferente e, em seus nós, considerará subconjuntos aleatórios diferentes de features.
4. **Previsão:** Uma nova pessoa (idade=30, profissão=Engenheiro, gênero=Masculino) é mostrada para todas as 500 árvores:
  - 300 árvores votam "Gosta".
  - 200 árvores votam "Não Gosta".
  - **Previsão Final:** "Gosta" (voto majoritário).

### 4.4 Exmplo em Python

```
1 # Importando bibliotecas
2 from sklearn.datasets import load_breast_cancer
3 from sklearn.model_selection import train_test_split
4 from sklearn.ensemble import RandomForestClassifier
5 from sklearn.metrics import classification_report, confusion_matrix,
   roc_auc_score
6 import matplotlib.pyplot as plt
7 import seaborn as sns
8 import pandas as pd
9
10 # 1. Carregar dataset
11 data = load_breast_cancer()
12 X, y = data.data, data.target
13 feature_names = data.feature_names
14
15 # 2. Dividir em treino e teste
```

```
16 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size
    =0.3, random_state=42)
17
18 # 3. Treinar Random Forest
19 clf = RandomForestClassifier(
20     n_estimators=100,          # número de árvores
21     max_depth=None,           # profundidade ilimitada
22     random_state=42,
23     n_jobs=-1                  # usar todos os núcleos do processador
24 )
25 clf.fit(X_train, y_train)
26
27 # 4. Avaliacao
28 y_pred = clf.predict(X_test)
29 y_prob = clf.predict_proba(X_test)[:,1]
30
31 print("Relatorio de Classificacao:\n", classification_report(y_test,
    y_pred, target_names=data.target_names))
32 print("AUC-ROC:", roc_auc_score(y_test, y_prob))
33
34 # 5. Matriz de Confusao
35 cm = confusion_matrix(y_test, y_pred)
36 plt.figure(figsize=(6,5))
37 sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", xticklabels=data.
    target_names, yticklabels=data.target_names)
38 plt.xlabel("Previsto")
39 plt.ylabel("Real")
40 plt.title("Matriz de Confusao - Random Forest")
41 plt.show()
42
43 # 6. Importancia das Variaveis
44 importances = clf.feature_importances_
45 df_importances = pd.DataFrame({"feature": feature_names, "importance"
    : importances})
46 df_importances = df_importances.sort_values("importance", ascending=
    False).head(10)
47
48 plt.figure(figsize=(8,6))
49 sns.barplot(x="importance", y="feature", data=df_importances, palette
    ="viridis")
50 plt.title("Top 10 Variáveis Mais Importantes - Random Forest")
51 plt.show()
```

### O que esse código faz:

1. Carrega o dataset de câncer de mama.
2. Divide em treino (70%) e teste (30%).
3. Treina um Random Forest com 100 árvores.
4. Avalia o modelo com precisão, recal, F1-Score e AUC-ROC.
5. Mostra a matriz de confusão.
6. Exibe as 10 variáveis mais importantes para a classificação.

#### 4.5 Pontos Positivos (Vantagens)

- **Alta Precisão:** Geralmente oferece performance de ponta ("state-of-the-art") em uma ampla variedade de problemas, competindo com algoritmos muito mais complexos.
- **Robusto contra Overfitting:** A aleatoriedade e a média de muitas árvores previnem que o modelo memorize o ruído dos dados de treinamento. Isso é sua maior vantagem sobre uma Árvore de Decisão única.
- **Lida Bem com Grandes Conjuntos de Dados:** O algoritmo é altamente paralelizável. Cada árvore pode ser treinada independentemente em um núcleo de CPU diferente, tornando-o ideal para Big Data.
- **Fornecer Importância de Features:** Calcula automaticamente quais features são mais preditivas, o que é valioso para entendimento do problema.
- **Versátil:** Funciona para classificação e regressão.
- **Lida com Dados Desbalanceados:** Oferece opções para balanceamento de classes (ex: `class_weight='balanced'` no `Scikit-learn`).
- **Não Requer Normalização de Dados:** Como é baseado em árvores, não é afetado pela escala das features.

#### 4.6 Pontos Negativos (Desvantagens)

item **Pouco Interpretável ("Black Box"):** Enquanto uma única árvore é fácil de explicar, uma floresta com 500 árvores não é. É difícil extrair regras simples de negócio.

- **Computacionalmente Intensivo e Lento para Prever:** Treinar e fazer previsões com centenas de árvores consome mais recursos computacionais e tempo do que um modelo simples como Regressão Logística. Não é ideal para aplicações que exigem previsões em tempo real muito rápido (low-latency).
- **Pode ser Propenso a Overfitting em Dados Muito Ruidosos:** Embora muito mais robusto que uma árvore única, se os dados forem extremamente ruidosos, a floresta acaba sobreajustando.
- **Tendência de Interpolar Mal:** Em tarefas de regressão, as previsões do Random Forest tendem a ser "suavizadas" e não extrapolam bem para fora do intervalo dos dados de treinamento. Ele não produz previsões fora do range visto durante o treinamento.

#### 4.7 Conclusão

O Random Forest é um algoritmo poderoso, robusto e incrivelmente popular que resolve as principais fraquezas das Árvores de Decisão. Sua capacidade de lidar com dados complexos, sua alta precisão e sua natureza paralelizável o tornam uma ferramenta indispensável no toolkit de qualquer Cientista de Dados ou Engenheiro de Machine Learning, especialmente no contexto de Big Data.

Ele é frequentemente a segunda melhor opção para quase qualquer problema: não é sempre o absoluto melhor, mas é quase sempre excepcionalmente bom e confiável. É excelente ponto de partida após criar uma baseline com um modelo mais simples, antes de partir para algoritmos de boosting mais complexos como XGBoost ou LightGBM.