

Introdução a Machine Learning

Profa. Dra. Roberta Wichmann

roberta.wichmann@idp.edu.br

Aula 5 – Classificação no Aprendizado Supervisionado: Métodos e Avaliação

Recapitulação e conceitos iniciais em classificação no Machine Learning.

Métricas de avaliação:
Precision, Recall,
Matriz de confusão, MCC e curva ROC.

Conceitos iniciais sobre árvores de decisão, KNN, regressão logística e SVM para classificação.

Recapitulando a Jornada do Machine Learning

O que nós vimos na aula passada?

- Vimos como um **modelo paramétrico como a Regressão Linear "aprende"**: ele usa o método dos Mínimos Quadrados.
- Também aprendemos que nem toda informação é útil. **Com as técnicas de Seleção de Variáveis (Forward e Backward)**, agimos como um editor que corta palavras desnecessárias de um texto para deixá-lo mais claro e poderoso.
- Entendemos que **não basta criar um modelo**; precisamos saber dar uma "nota" para ele.
- Métricas como **MAE e RMSE nos dizem, em média, "o quão longe" nossas previsões estão do valor real**. O MAE é como medir a distância média a pé, enquanto o RMSE é mais rigoroso e penaliza mais os erros muito grandes.

Recapitulando a Jornada do Machine Learning

O que nós vimos na aula passada?

- Já o R^2 é diferente: **ele nos dá uma nota percentual (de 0% a 100%)** de quanto da variabilidade dos dados o nosso modelo conseguiu explicar.
- Demos uma **primeira olhada nos modelos não paramétricos**, que são mais adaptáveis.
- O KNN Regressor **funciona baseado nos vizinhos mais próximos**: para saber o preço de uma casa, ele olha para as 'K' casas mais parecidas na vizinhança e tira a média dos preços delas.
- As Árvores de Regressão **funcionam como um fluxograma que divide os dados em grupos** cada vez menores e mais puros para, no final, dar uma previsão baseada no grupo.

Mas o que acontece quando nossa variável target é uma categoria?

Conceitos iniciais sobre Classificação

A Grande Mudança de Paradigma: De "Quanto?" para "O Quê?"

- Quando a **nossa variável alvo (target) deixa de ser um número** (como o preço de uma casa) e se torna uma categoria (como "Spam" ou "Não Spam"), saímos do mundo da Regressão e entramos no universo da Classificação.
- O objetivo agora **não é mais prever um valor numérico em uma escala contínua, mas sim ensinar o computador a rotular**, a organizar os dados em "caixas" ou grupos pré-definidos e distintos entre si.
- Pense na **diferença entre um meteorologista que prevê a temperatura exata de amanhã**, como "23.5 graus" (Regressão), e um que prevê se o dia será "Ensolarado", "Nublado" ou "Chuvoso" (Classificação). A natureza da resposta muda completamente o jogo.

Conceitos iniciais sobre Classificação

Exemplos Práticos

- Você **já usa modelos de classificação** todos os dias sem perceber!
- Seu provedor de e-mail, por exemplo, usa um classificador para ler uma nova mensagem e decidir se **ela pertence à categoria "Lixo Eletrônico" ou à categoria "Caixa de Entrada"**. O sistema não mede "o quão spam" é o e-mail; ele apenas o coloca em uma das duas caixas.
- Da mesma forma, um sistema de banco analisa as características de uma transação e a classifica como **"Fraudulenta" ou "Legítima"**, ou um aplicativo de fotos que reconhece rostos e os rotula com os nomes dos seus amigos, escolhendo o nome correto de uma lista de possibilidades.

E como medimos o quão bom é o modelo?

Conceitos iniciais sobre Classificação

Uma Nova Régua para Medir o Sucesso

- Se a tarefa mudou, **nossas métricas de avaliação também precisam mudar drasticamente.**
- Métricas como **RMSE e MAE**, que medem a distância entre a previsão e o real, **perdem totalmente o sentido aqui**, pois não podemos calcular a "distância" entre a previsão "Cachorro" e o valor real "Gato".
- Em vez de **medir o tamanho do erro numérico**, agora vamos nos concentrar na **quantidade de acertos.**

Conceitos iniciais sobre Classificação

Uma Nova Régua para Medir o Sucesso

- A pergunta principal se torna: "**Quantas vezes o modelo acertou a categoria correta?**".
- É como a **diferença fundamental entre uma prova de história e um teste de múltipla escolha**. Na prova de matemática, um erro pode ser pequeno ou grande (medido pelo RMSE).
- No teste de múltipla escolha, ou **você marca a alternativa correta (acerto) ou você marca a errada (erro)**. A partir de agora, nosso "boletim" será baseado em métricas que mede exatamente essa porcentagem de acertos.

Vamos analisar um exemplo?

Conceitos iniciais sobre Classificação

Nosso Desafio Fictício

- Criamos um modelo de Machine Learning chamado "Gato-ou-Não". **A missão dele é analisar 100 imagens e classificá-las em duas categorias: "Contém Gato" ou "Não Contém Gato".**
- Temos as seguintes informações sobre a base de dados:
 - **Total de Imagens:** 100
 - **Imagens que realmente têm gatos:** 30
 - **Imagens que realmente NÃO têm gatos:** 70

Conceitos iniciais sobre Classificação

A Missão do Nosso Modelo

- Vamos colocar **nosso recém-criado modelo de Machine Learning**, o "Gato-ou-Não", à prova.
- Sua única e importante missão é **analisar um lote de 100 imagens e tomar uma decisão binária** para cada uma delas.
- Para cada imagem que ele "vê", ele **deve responder a uma simples pergunta**: esta imagem pertence à categoria "Contém Gato" ou à categoria "Não Contém Gato"?
- Pense no modelo como um porteiro. **Para cada convidado que chega (uma imagem), ele precisa tomar a decisão final**: "Pode entrar" (Contém Gato) ou "Entrada negada" (Não Contém Gato).

Conceitos iniciais sobre Classificação

Nosso gabarito

- Antes de julgarmos o desempenho do nosso porteiro-robô, **precisamos saber a verdade sobre os nossos dados**. Este é o nosso gabarito oficial, a lista de convidados correta.
- No nosso conjunto de 100 imagens, **nós sabemos com 100% de certeza a resposta correta para cada uma**: 30 imagens realmente possuem gatos e as outras 70 imagens não possuem gatos (podem ser paisagens, cachorros, objetos, etc.).
- Essa é a realidade imutável na qual **vamos comparar cada uma das 100 decisões tomadas pelo nosso modelo** para ver o quão bom ele é em seu trabalho.

Conceitos iniciais sobre Classificação

Definindo o Foco - A Classe Positiva

- Em problemas de classificação, é **fundamental definir qual categoria é o nosso "alvo", o evento de interesse que estamos tentando detectar**. A essa categoria damos o nome de classe positiva.
- Em nosso desafio, **o objetivo é encontrar gatos**. Portanto, definimos que a classe "Contém Gato" é a nossa classe positiva. "Positivo" aqui não significa algo "bom", mas sim a confirmação do que estamos procurando.

Vamos ver o desempenho do modelo?

Conceitos iniciais sobre Classificação

O Veredito Inicial

- Após rodar o "Gato-ou-Não" em todas as 100 imagens, obtivemos o resultado geral: **o modelo acertou a classificação de 85 imagens e errou em 15 delas.**
- **Como posso mensurar se o modelo é bom ou não?**
- **Resposta:** Podemos usar a abordagem mais intuitiva de todas! A **acurácia**.
- A Acurácia é uma **métrica de classificação que divide a quantidade de acertos do modelo pelo total de observações**, isto é, a quantidade de imagens que ele acertou dividido pela quantidade total de imagens.

$$\text{Acuracia} = \frac{\text{Quantidade de acertos}}{\text{Quantidade total de casos}} = \frac{85}{100} * 100 = 85\%$$

Conceitos iniciais sobre Classificação

O Veredito Inicial

- À primeira vista, **uma taxa de acerto de 85% (chamada de Acurácia) parece um ótimo resultado!** Mas essa única nota esconde detalhes cruciais sobre o tipo de erro que ele comete.

O Teste do "Modelo Preguiçoso"

- Imagine um modelo extremamente preguiçoso. **Ele percebe que 70% das imagens não têm gatos. Então, ele adota uma estratégia simples:** "Vou dizer 'Não Contém Gato' para TODAS as imagens, sem nem olhar".

Adivinhe a acurácia dele?

Conceitos iniciais sobre Classificação

O Teste do "Modelo Preguiçoso"

- Ele **acertaria as 70 imagens que não têm gato e erraria as 30 que têm, resultando em uma Acurácia de 70%!** Um modelo completamente inútil, que não encontra um único gato, ainda assim consegue uma nota aparentemente razoável.
- Isso evidencia que **a Acurácia, por si só, não revela toda a realidade do desempenho do modelo.** Os 15 erros cometidos podem ser divididos em dois tipos distintos. Retomando a analogia do porteiro da festa dos gatos, há duas formas de falhas sérias. cada uma com implicações diferentes.

Quais são elas?

Conceitos iniciais sobre Classificação

Entendendo os Erros Tipo I e Tipo II no nosso modelo "Gato-ou-Não"

- Um **Falso Positivo** ocorre quando nosso modelo prevê "Contém Gato", mas, na realidade, a imagem não tinha um gato.
- Em termos estatísticos, o modelo **rejeitou a Hipótese Nula (H_0)** de que "não há gato", quando essa hipótese era, na verdade, verdadeira.
- A **Hipótese Nula (H_0)** é a afirmação inicial que assumimos como verdadeira antes de olhar para as evidências; geralmente, é uma declaração de "nenhum efeito" ou "nada aconteceu".
- Essa falha específica é chamada de **Erro Tipo I**.

Conceitos iniciais sobre Classificação

Entendendo os Erros Tipo I e Tipo II no nosso modelo "Gato-ou-Não"

- Um **Falso Negativo** ocorre quando nosso modelo prevê "Não Contém Gato", mas, na verdade, a imagem continha um gato.
- Estatisticamente, **o modelo não rejeitou a Hipótese Nula (H_0)**, mantendo a ideia de que "não há gato", quando essa hipótese era falsa.
- Essa falha é conhecida como **Erro Tipo II**.

Qual é o mais importante?

Conceitos iniciais sobre Classificação

Resposta

- A resposta é: **depende inteiramente do problema que você está tentando resolver.** O custo de cada tipo de erro varia drasticamente com o contexto da aplicação. Vamos ver dois exemplos claros.

Exemplo 1: O Custo do Falso Negativo (Diagnóstico de Doença)

- Imagine um **modelo que classifica pacientes como "Doente" ou "Saudável"**. Um **Falso Negativo** aqui significa dizer a uma pessoa que tem uma doença grave que ela está perfeitamente saudável.
- As consequências são catastróficas: **a pessoa não buscará tratamento, a doença progredirá e a vida dela pode ser colocada em risco.** Neste cenário, um Falso Negativo é o pior erro possível e deve ser minimizado a todo custo, mesmo que isso signifique ter mais Falsos Positivos.

Conceitos iniciais sobre Classificação

Exemplo 2: O Custo do Falso Positivo (Filtro de Spam)

- Agora, **pense em um modelo que classifica e-mails como "Spam" ou "Não Spam"**. Um **Falso Positivo** significa que um e-mail importante (como uma oferta de emprego ou um e-mail de um familiar) foi classificado incorretamente como spam e enviado para a lixeira.
- O usuário **pode perder uma oportunidade crucial ou uma comunicação importante**. Neste caso, um Falso Positivo é extremamente prejudicial. Preferimos que o modelo seja mais cauteloso e até deixe alguns spams passarem (Falsos Negativos) do que arriscar perder um e-mail legítimo.
- Portanto, **não podemos nos contentar com uma única medida**. Precisamos de uma ferramenta que nos mostre o detalhe de cada tipo de acerto e erro, permitindo-nos julgar o modelo com base no que realmente importa para o nosso problema.

Métricas em Classificação

A Matriz de Confusão

- A Matriz de Confusão é a ferramenta visual e fundamental para diagnosticar a performance de um modelo de classificação. Ela é uma tabela simples que cruza os valores reais (o gabarito) com os valores previstos pelo modelo.
- Seu nome é perfeito porque ela nos mostra exatamente onde o modelo está se "confundindo". Em vez de uma métrica única, ela nos dá um relatório completo, detalhando cada tipo de acerto e cada tipo de erro que ocorreu.

Vamos ver como é a matriz de confusão?

Métricas em Classificação

	Previsto: Gato (Positivo)	Previsto: Não Gato (Negativo)	Total Real
Real Gato(Positivo)	VP = 25	FN = 5	30
Real: Não Gato (Negativo)	FP = 10	VN = 60	70
Total Previsto	35	65	100

- A matriz 2x2 nos apresenta quatro cenários possíveis para cada uma das 100 imagens que nosso modelo avaliou.
- **Verdadeiros Positivos (VP ou TP):** O acerto perfeito! A imagem realmente tinha um gato (Verdadeiro) e o nosso modelo corretamente previu "Contém Gato" (Positivo).
- Resultado no nosso teste: **25 imagens.**

Métricas em Classificação

	Previsto: Gato (Positivo)	Previsto: Não Gato (Negativo)	Total Real
Real Gato(Positivo)	VP = 25	FN = 5	30
Real: Não Gato (Negativo)	FP = 10	VN = 60	70
Total Previsto	35	65	100

- **Verdadeiros Negativos (VN ou TN):** O outro acerto perfeito! A imagem realmente não tinha um gato (Verdadeiro) e o nosso modelo corretamente previu "Não Contém Gato" (Negativo).
- Resultado no nosso teste: **60 imagens.**

Métricas em Classificação

	Previsto: Gato (Positivo)	Previsto: Não Gato (Negativo)	Total Real
Real Gato(Positivo)	VP = 25	FN = 5	30
Real: Não Gato (Negativo)	FP = 10	VN = 60	70
Total Previsto	35	65	100

- **Falsos Positivos (FP):** O "alarme falso"! A imagem não tinha um gato (Falso), mas o modelo se confundiu e previu "Contém Gato" (Positivo). É o nosso "cachorro impostor" que entrou.
- Resultado no nosso teste: **10 imagens.**

Métricas em Classificação

	Previsto: Gato (Positivo)	Previsto: Não Gato (Negativo)	Total Real
Real Gato(Positivo)	VP = 25	FN = 5	30
Real: Não Gato (Negativo)	FP = 10	VN = 60	70
Total Previsto	35	65	100

- **Falsos Negativos (FN):** A "falha na detecção"! A imagem realmente tinha um gato (Falso), mas o modelo não o viu e previu "Não Contém Gato" (Negativo). É o nosso "gato legítimo" que foi barrado na porta.
- Resultado no nosso teste: **5 imagens.**

Métricas em Classificação

O que concluímos?

- Com esta matriz, a história fica clara! **A acurácia de 85% não é apenas um número. Ela é composta por 25 acertos em gatos e 60 acertos em não-gatos.**
- **Mais importante:** agora podemos quantificar os erros que mais nos preocupam. Sabemos que nosso modelo produziu 10 alarmes falsos (FP) e deixou 5 gatos passarem (FN).
- Esta matriz é a nossa base. **A partir destes quatro números fundamentais, podemos agora calcular métricas muito mais inteligentes e específicas**, como Precision e Recall, para medir o desempenho do modelo em relação aos erros que mais importam para nós.

Métricas em Classificação

Precision

- A Precision responde a uma pergunta muito específica e focada na qualidade das previsões positivas: "De todas as vezes que o modelo previu a categoria positiva (disse 'Contém Gato'), quantas vezes ele estava realmente certo?"
- Essa métrica é crucial quando o custo de um Falso Positivo (FP) é alto. Ela mede a "pureza" das previsões positivas do modelo. Uma alta precision significa que o modelo é muito confiável quando afirma ter encontrado algo.

Como se calcula?

Métricas em Classificação

Fórmula

$$\text{Precision} = \frac{VP}{VP + FP} = \frac{25}{25 + 10} = \frac{25}{35} \approx 71.4\%$$

Interpretação

- Das vezes que o modelo previu 'Contém Gato', ele acertou em 71,4% delas, isto é a Precision utiliza dos VP (Verdadeiros Positivos) e dos FP (Falsos Positivos) para gerar uma taxa que varia de 0 até 100%, de forma que quanto mais próximo dos 100% melhor é o modelo em acertar a classe positiva (contém gato).

E o recall?

Métricas em Classificação

Recall

- O Recall (também chamado de Sensibilidade ou Taxa de Verdadeiros Positivos) responde a uma pergunta complementar, focada na capacidade de detecção: **"De todas as imagens que realmente continham um gato, qual a porcentagem que o nosso modelo conseguiu identificar?"**
- Essa métrica é **fundamental quando o custo de um Falso Negativo (FN) é muito alto**. Ela mede a "abrangência" ou a "completude" do modelo em encontrar todas as instâncias positivas que existem.

Como se calcula?

Métricas em Classificação

Fórmula

$$\text{Recall} = \frac{VP}{VP + FN} = \frac{25}{25 + 5} = \frac{25}{30} \approx 83.3\%$$

Interpretação

- O modelo conseguiu capturar 83,3% das imagens que realmente “contém o gato”, isto é a Recall utiliza dos VP (Verdadeiros Positivos) e dos FN (Falsos Negativos) para gerar uma taxa que varia de 0 até 100%, de forma que quanto mais próximo dos 100% mais robusto é o modelo contra os FN.

Mas a interpretação parece a mesma! O que muda?

Métricas em Classificação

Note o seguinte:

$$\text{Precision} = \frac{VP}{VP + FP} = \frac{25}{25 + 10} = \frac{25}{35} \approx 71.4\%$$

- O precision tem no seu denominador o componente FP (quantidade falsos positivos).

O que acontece quando FP é muito alto?

- Como o FP está no denominador, caso o FP tenha um valor muito alto, então o Precision vai diminuir muito seu valor! Imagine que agora temos os valores de VP = 15 e FP = 20, o Precision novo será de:

$$\text{Precision} = \frac{VP}{VP + FP} = \frac{15}{15 + 20} = \frac{15}{35} \approx 42.86\%$$

- Portanto o Precision é bastante sensível aos Falsos Positivos!

Métricas em Classificação

No Recall, temos:

$$\text{Recall} = \frac{VP}{VP + FN} = \frac{25}{25 + 5} = \frac{25}{30} \approx 83.3\%$$

- O Recall tem no seu denominador o componente FN (quantidade falsos negativos).

O que acontece quando FN é muito alto?

- Como o FN está no denominador, caso o FN tenha um valor muito alto, então o Recall vai diminuir muito seu valor! Imagine que agora temos os valores de $VP = 5$ e $FN = 15$, o Recall novo será de:

$$\text{Recall} = \frac{VP}{VP + FN} = \frac{5}{5 + 15} = \frac{5}{20} = 25\%$$

- Portanto o Recall é bastante sensível aos Falsos Negativos!

Métricas em Classificação

Trade-off entre precision e recall

- Frequentemente, **o precision e o recall entram em conflito**. Se tornarmos nosso modelo mais rigoroso para aumentar a Precision, ele provavelmente deixará de encontrar alguns casos menos óbvios, diminuindo o Recall (o modelo deixa de encontrar alguns gatos, mesmo que estejam visíveis).
- Se o **tornarmos mais "sensível" para aumentar o Recall** (encontrar todos os gatos, não importa o quão escondidos), ele provavelmente cometerá mais alarmes falsos, diminuindo a Precision.

Tem alguma métrica que resolve isso?

Métricas em Classificação

F1-score

- O F1-Score entra em cena para resolver esse dilema. **Ele é a média harmônica de Precision e Recall.**
- Isso significa que **o F1-Score só será alto se AMBAS, Precision e Recall, forem altas.** É a métrica ideal quando você precisa de um bom equilíbrio entre não cometer alarmes falsos e, ao mesmo tempo, não deixar passar nenhuma instância positiva.

Como se calcula?

Métricas em Classificação

Fórmula

$$F1-Score = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} = 2 \cdot \frac{0.714 \cdot 0.833}{0.714 + 0.833} = 2 \cdot \frac{0.595}{1.547} = 2 \cdot 0.384 = 0.769 \approx 76.9\%$$

Conclusão

- Como tanto o precision quanto o recall variam de 0 a 1 e quanto mais próximo de 1, melhor, nosso modelo possui um valor equilibrado de F1-Score de 76,7%, o que indica um bom balanceamento entre o precision e o recall.

Existem mais métricas?

Métricas em Classificação

MCC (Matthews Correlation Coefficient)

- Pense no MCC como a métrica mais rigorosa e justa. Diferente de outras métricas que podem focar em um aspecto específico, o MCC é a **única métrica popular que, por design, leva em consideração todos os quatro quadrantes da Matriz de Confusão** (VP, VN, FP, FN) em seu cálculo de forma equilibrada.
- Ele gera um único número, um coeficiente de correlação que varia de -1 a +1, para resumir a qualidade geral da classificação. **Essencialmente, ele mede o quão bem as previsões do modelo se correlacionam com a realidade e quanto mais próximo de +1, melhor é o modelo.**

Como se calcula?

Métricas em Classificação

Fórmula

$$MCC = \frac{(VP \times VN) - (FP \times FN)}{\sqrt{(VP + FP)(VP + FN)(VN + FP)(VN + FN)}} = \frac{(25 \times 60) - (10 \times 5)}{\sqrt{(25 + 10)(25 + 5)(60 + 10)(60 + 5)}} \approx \frac{1450}{2185.8} \approx 0.664$$

Conclusão

- Esta é uma nota boa. Ela nos diz que existe uma correlação positiva entre as previsões do nosso modelo e a realidade.
- O valor está bem longe de 0 (aleatório) e se aproxima de +1 (perfeito), confirmando, de forma robusta e balanceada, que nosso modelo é genuinamente útil e informativo.

Existe abordagens visuais para avaliação?

Métricas em Classificação

Curva ROC

- Até agora, tratamos as previsões do modelo como uma decisão final de "Sim" ou "Não". Na realidade, **a maioria dos modelos calcula uma probabilidade** (ex: "tenho 75% de certeza que isto é um gato"). A decisão final é tomada comparando essa probabilidade a um limite de corte (threshold), que geralmente é 50%.
- A Curva ROC (Receiver Operating Characteristic) é um gráfico que mostra como o desempenho do modelo muda quando ajustamos o ponto de corte (threshold) que decide se uma imagem contém um gato ou não.

O que é esse ponto de corte?

Métricas em Classificação

Ponto de Corte

- O threshold é um valor limite usado para converter probabilidades previstas por um modelo em classes binárias.
- Por exemplo, se um modelo prevê 0,7 de probabilidade para um paciente ter doença, e o threshold é 0,5, ele será classificado como doente.
- Alterar o threshold muda o número de positivos e negativos previstos pelo modelo, impactando métricas como sensibilidade e precisão.

E como funciona na prática?

Métricas em Classificação

Ponto de Corte

- Suponha um modelo que prevê a probabilidade de um email ser spam:
 - Email 1: 0,9 e a **classe real é Spam**
 - Email 2: 0,2 e a **classe real é Não Spam**
 - Email 3: 0,6 e a **classe real é Spam**
 - Email 4: 0,4 e a **classe real é Não Spam**
- **Com threshold 0,3, quais dos e mails serão classificados como spam?**
 - Email 1: Spam - **VP**
 - Email 2: Não Spam - **VN**
 - Email 3: Spam - **VP**
 - Email 4: Spam - **FP**

Métricas em Classificação

Ponto de Corte

- Com threshold 0,3, quais dos e mails serão classificados como spam?
 - Email 1: Spam - **VP**
 - Email 2: Não Spam - **VN**
 - Email 3: Spam - **VP**
 - Email 4: Spam - **FP**
- Portanto o Precision será de:

$$\text{Precision} = \frac{VP}{VP + FP} = \frac{2}{2 + 1} = \frac{2}{3} \approx 0.67$$

Métricas em Classificação

Ponto de Corte

- Com threshold 0,3, quais dos e mails serão classificados como spam?

- Email 1: Spam - **VP**
- Email 2: Não Spam - **VN**
- Email 3: Spam - **VP**
- Email 4: Spam - **FP**

- E o Recall será:

$$\text{Recall} = \frac{VP}{VP + FN} = \frac{2}{2 + 0} = \frac{2}{2} = 1$$

Métricas em Classificação

Ponto de Corte

- Suponha um modelo que prevê a probabilidade de um email ser spam:
 - Email 1: 0,9 e a **classe real é Spam**
 - Email 2: 0,2 e a **classe real é Não Spam**
 - Email 3: 0,6 e a **classe real é Spam**
 - Email 4: 0,4 e a **classe real é Não Spam**
- **Com threshold 0,5, quais dos e mails serão classificados como spam?**
 - Email 1: Spam - **VP**
 - Email 2: Não Spam - **VN**
 - Email 3: Spam - **VP**
 - Email 4: Não Spam - **VN**

Métricas em Classificação

Ponto de Corte

- Com threshold 0,5, quais dos e mails serão classificados como spam?

- Email 1: Spam - VP
- Email 2: Não Spam - VN
- Email 3: Spam - VP
- Email 4: Não Spam - VN

- O Precision será:

$$\text{Precision} = \frac{VP}{VP + FP} = \frac{2}{2 + 0} = \frac{2}{2} = 1$$

Métricas em Classificação

Ponto de Corte

- Com threshold 0,5, quais dos e mails serão classificados como spam?

- Email 1: Spam - VP
- Email 2: Não Spam - VN
- Email 3: Spam - VP
- Email 4: Não Spam - VN

- E o Recall será:

$$\text{Recall} = \frac{VP}{VP + FN} = \frac{2}{2 + 0} = \frac{2}{2} = 1$$

Métricas em Classificação

Ponto de Corte

- Suponha um modelo que prevê a probabilidade de um email ser spam:
 - Email 1: 0,9 e a **classe real é Spam**
 - Email 2: 0,2 e a **classe real é Não Spam**
 - Email 3: 0,6 e a **classe real é Spam**
 - Email 4: 0,4 e a **classe real é Não Spam**
- **Com threshold 0,7, quais dos e mails serão classificados como spam?**
 - Email 1: Spam - **VP**
 - Email 2: Não Spam - **VN**
 - Email 3: Não Spam - **FN**
 - Email 4: Não Spam - **VN**

Métricas em Classificação

Ponto de Corte

- Com threshold 0,7, quais dos e mails serão classificados como spam?
 - Email 1: Spam - **VP**
 - Email 2: Não Spam - **VN**
 - Email 3: Não Spam - **FN**
 - Email 4: Não Spam - **VN**
- O Precision será:

$$\text{Precision} = \frac{VP}{VP + FP} = \frac{1}{1 + 0} = \frac{1}{1} = 1$$

Métricas em Classificação

Ponto de Corte

- Com threshold 0,7, quais dos e mails serão classificados como spam?

- Email 1: Spam - VP
- Email 2: Não Spam - VN
- Email 3: Não Spam - FN
- Email 4: Não Spam - VN

- O Recall será:

$$\text{Recall} = \frac{VP}{VP + FN} = \frac{1}{1 + 1} = \frac{1}{2} = 0.50$$

Métricas em Classificação

Ponto de Corte

Pontos De Corte	Precision	Recall
0,3	0,67	1
0,5	1	1
0,7	1	0,5

- Note que conforme aumentamos o ponto de corte, o Recall diminui ou seja começamos a observar um aumento de Falsos Negativos.
- Agora quando diminuimos o ponto de corte, o precision diminui.
- Essa dinâmica é chamado Trade-Off, assim o melhor ponto de corte que maximize tantos o precision e o recall é o ponto 0,5.

Métricas em Classificação

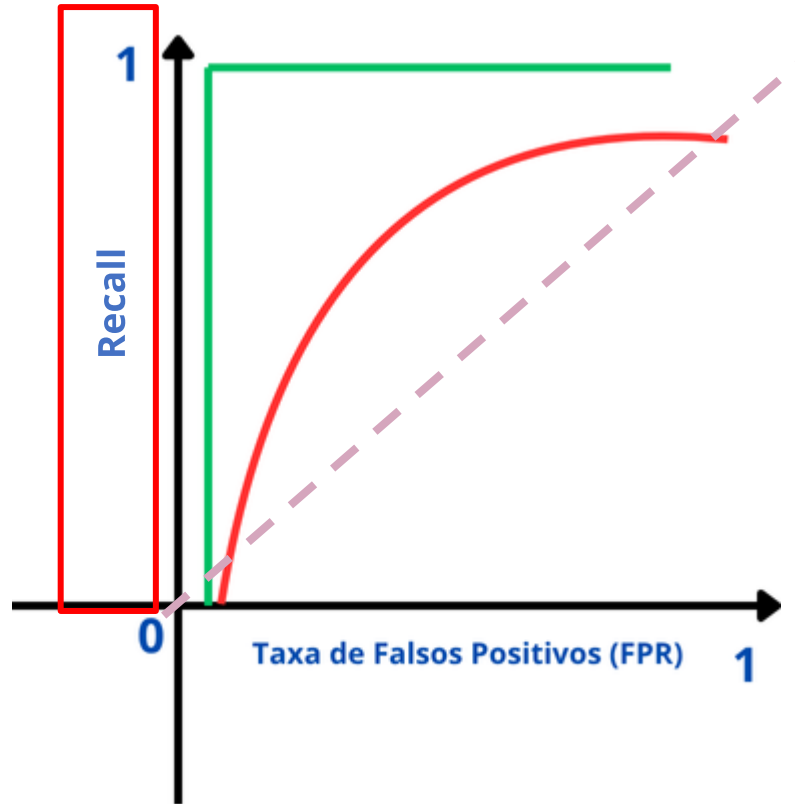
Ponto de Corte

- Perceba que conforme mudamos o ponto de corte, a classificação muda.
- Assim para a curva ROC conseguiremos calcular os valores de Recall e da métrica de Taxa de Falsos Positivos para cada um dos pontos de corte.
- Se variarmos mais ainda o ponto de corte conseguimos ter vários valores de Recall e de Taxa de Falsos Positivos e se plotarmos em um gráfico, conseguiremos um resumo do desempenho geral do nosso modelo.

E como funciona na prática?

Métricas em Classificação

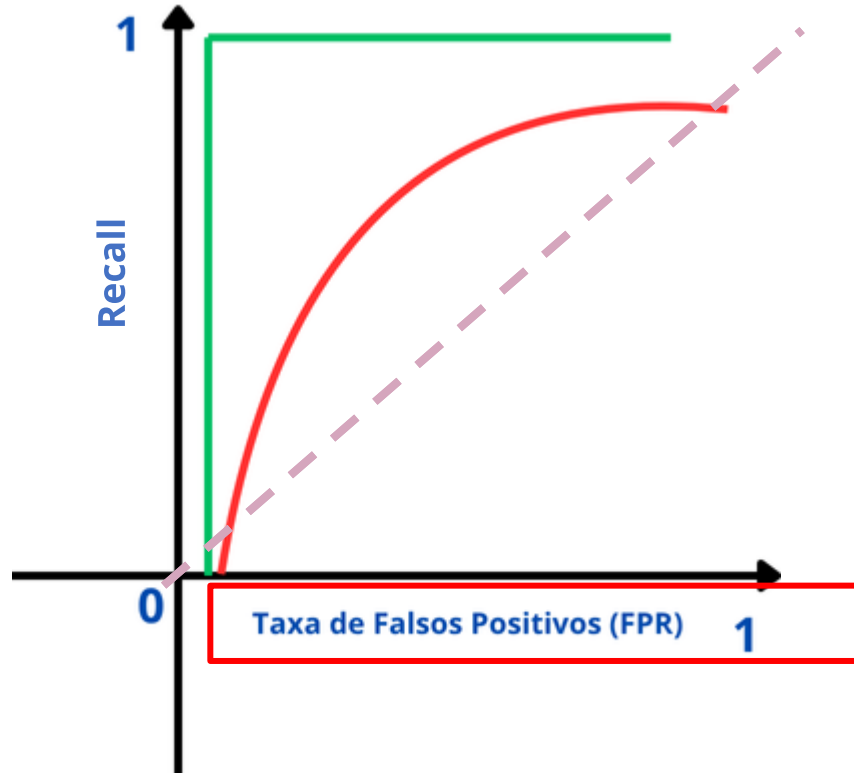
Curva ROC



- A Curva ROC plota **duas métricas importantes**, uma contra a outra.
- **Eixo Y (Vertical):** Recall: "De todos os gatos que realmente existem, qual a porcentagem que o modelo conseguiu encontrar?".
- Nosso objetivo aqui é **MAXIMIZAR** este valor. Queremos que a curva suba o mais rápido possível, pois significa que estamos encontrando os positivos.

Métricas em Classificação

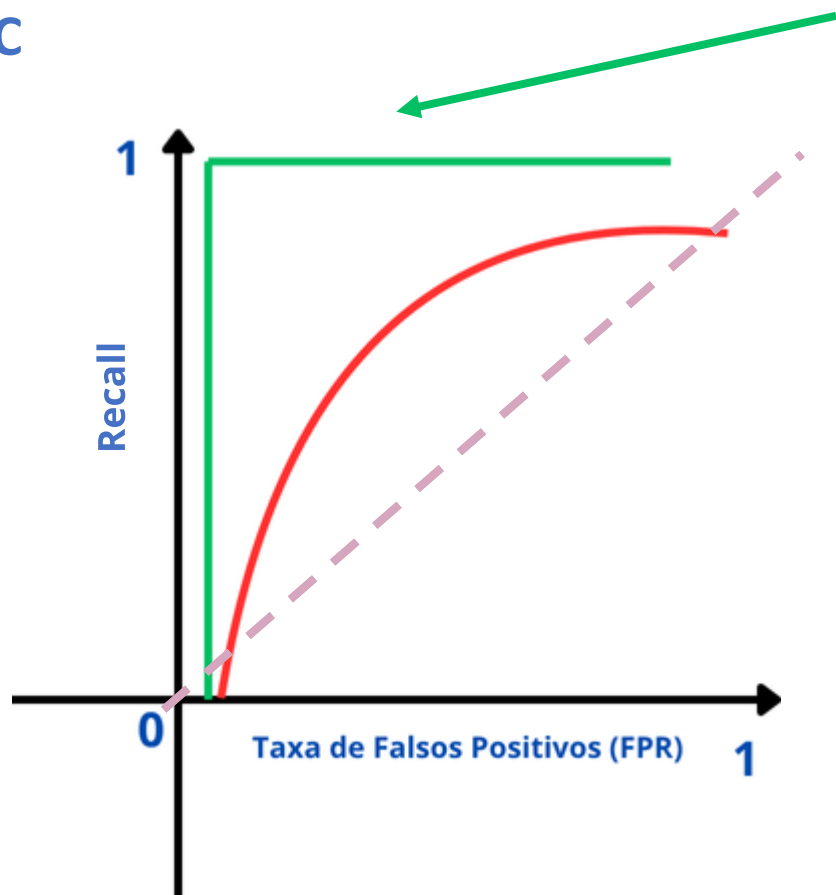
Curva ROC



- **Eixo X (Horizontal):** Esta métrica responde: "De todos os não-gatos, qual a porcentagem que o modelo incorretamente classificou como gatos?".
- Nosso objetivo aqui é **MINIMIZAR** este valor. Queremos que a curva avance o mais lentamente possível para a direita, pois significa que estamos evitando alarmes falsos.

Métricas em Classificação

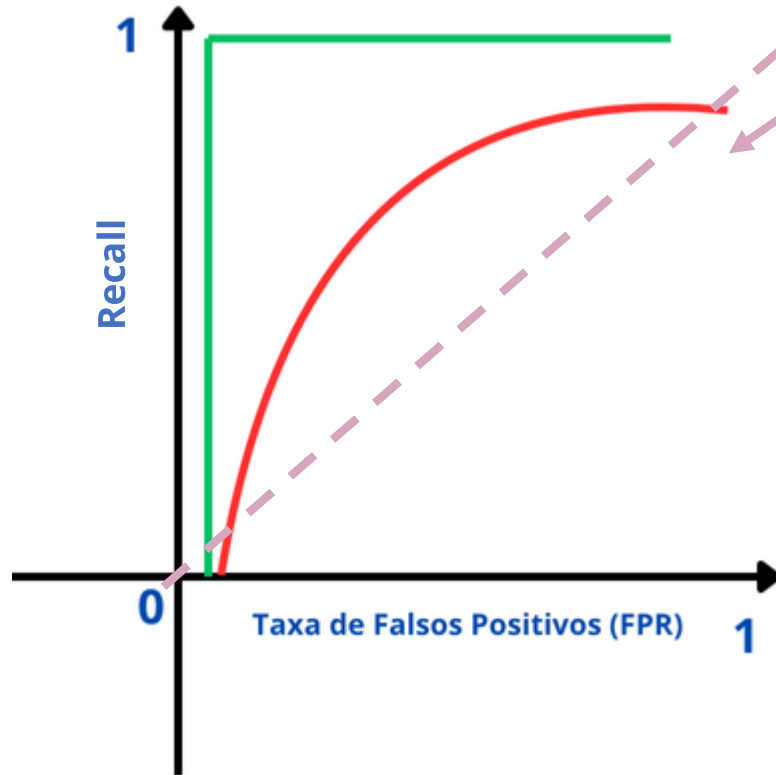
Curva ROC



- **Curva em verde** seria a curva ROC perfeita, **onde o modelo acerta tudo** já que a Taxa de Verdadeiros Positivos é ALTA e a taxa de Falsos Positivos é BAIXA.

Métricas em Classificação

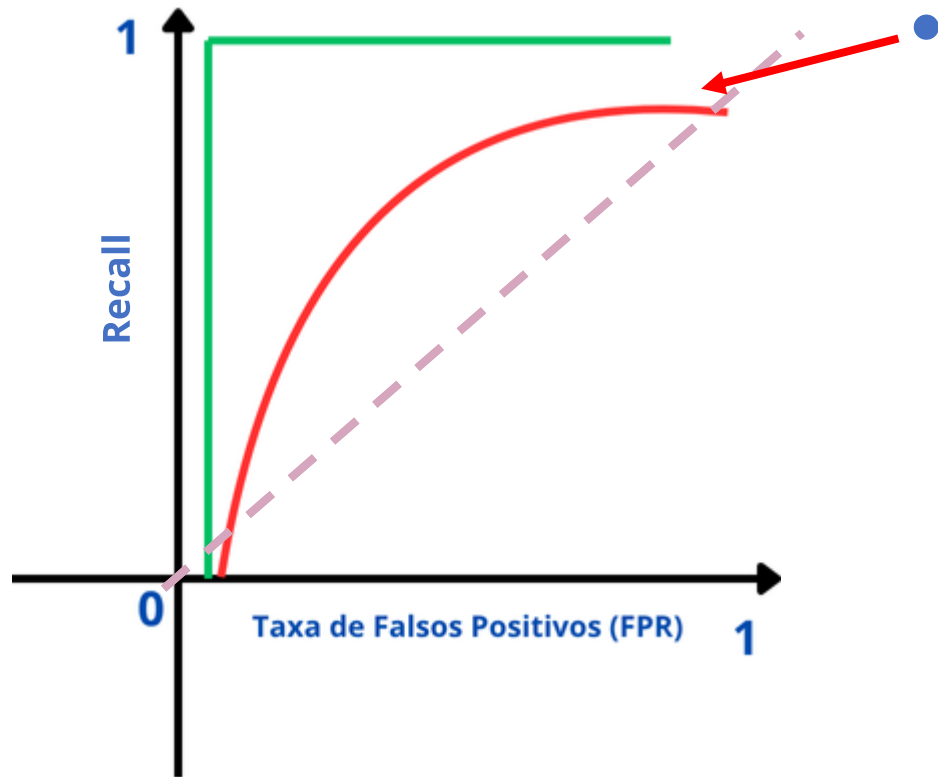
Curva ROC



- A linha Pontilhada é um **modelo aleatório**. Imagine que você vai classificar se tem ou não um gato na imagem jogando a moeda.
- Você teria 50% de probabilidade de acertar ou errar (é um modelo ruim mas definitivamente é um modelo).

Métricas em Classificação

Curva ROC

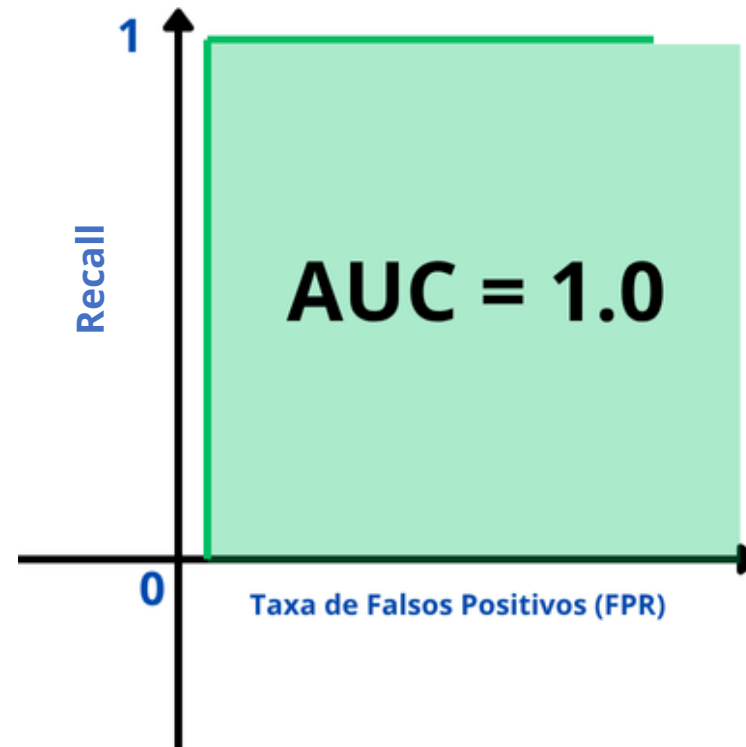


Com o **modelo aleatório**, podemos utilizar como modelo comparação para definir o que é um “bom” modelo. A **Curva vermelha** acima mostra um exemplo claro.

Métricas em Classificação

Curva ROC

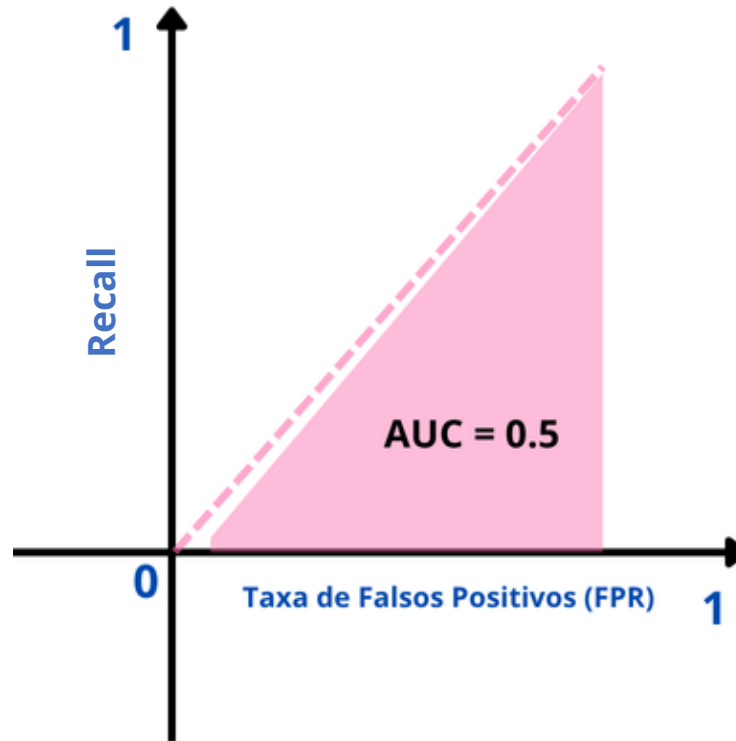
- Como um gráfico inteiro pode ser difícil de comparar, calculamos a **Área Sob a Curva (AUC)**, que nos dá uma única nota de 0 a 1 para resumir o desempenho geral.
- **AUC = 1.0**: Modelo perfeito (a área de um quadrado perfeito).



Métricas em Classificação

Curva ROC

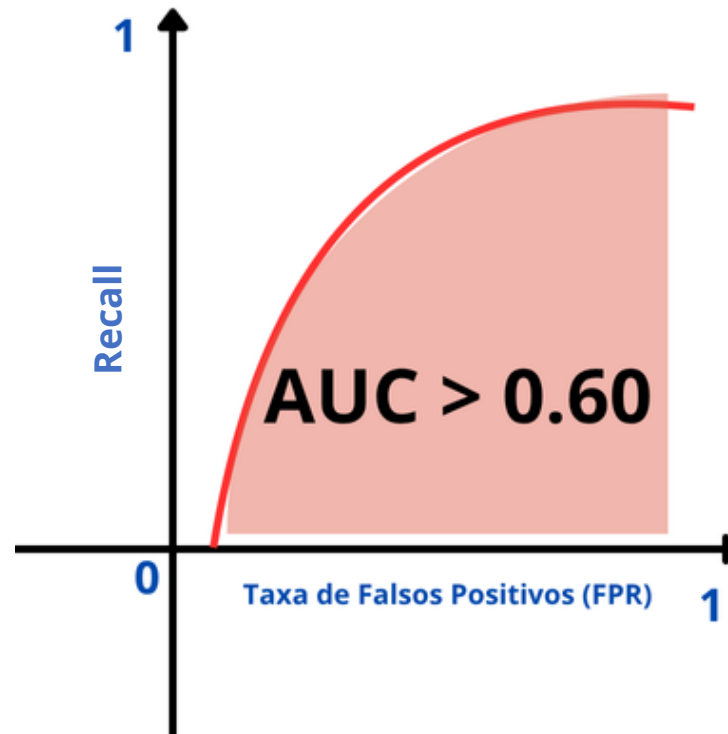
- Como um gráfico inteiro pode ser difícil de comparar, calculamos a **Área Sob a Curva (AUC)**, que nos dá uma única nota de 0 a 1 para resumir o desempenho geral.
- **AUC = 0.5**: Modelo inútil/aleatório (a área sob a linha diagonal).



Métricas em Classificação

Curva ROC

- Como um gráfico inteiro pode ser difícil de comparar, calculamos **a Área Sob a Curva (AUC)**, que nos dá uma única nota de 0 a 1 para resumir o desempenho geral.
- **AUC > 0.6:** Geralmente considerado um modelo útil. AUC > 0.9: Considerado um modelo excelente.



Modelos Para Classificação

Sobre Nossos Modelos

- Na aula passada utilizamos 2 modelos para fazer **modelagem quando a variável target de custos médicos é contínua** (regressão).
- E se invés de uma variável contínua, fosse algo categórico como por exemplo, um indivíduo possuir **“Custos Altos”** ou **“Custos Baixos”**.
- Agora **saímos da perspectiva da regressão e entramos** na modelagem de dados categóricos, isto é a classificação.
- A **boa notícia é que os modelos vistos conseguem ser utilizados para tarefas de classificação.**

Mas como os modelos vão se comportar para esse cenário?

Modelos Para Classificação

Exemplo:

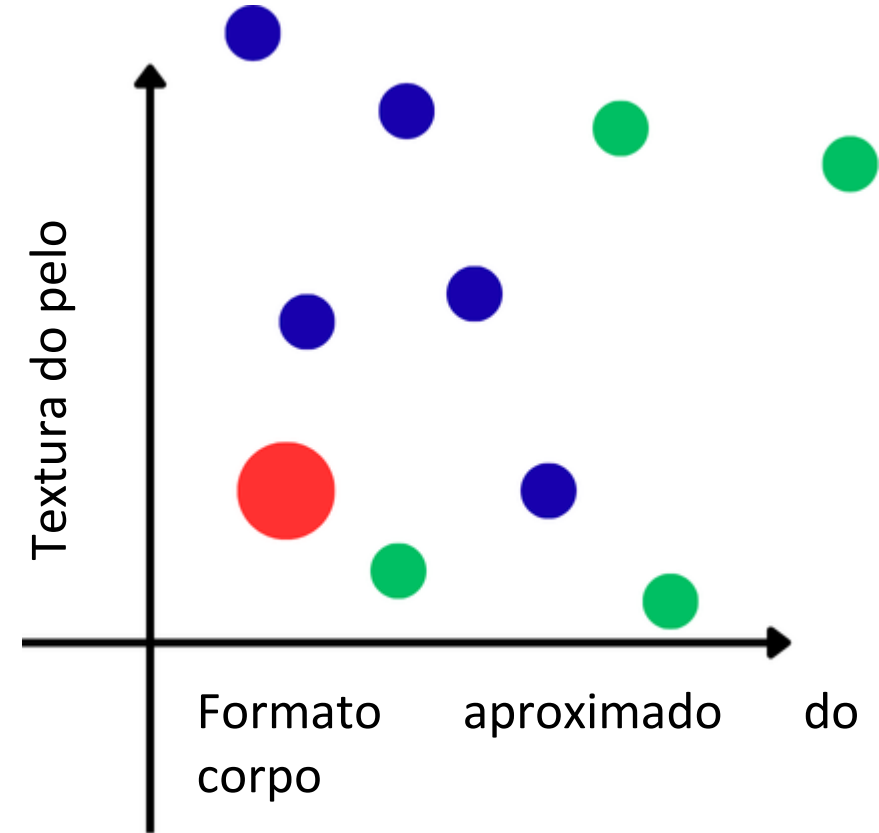
- Imagine que somos biólogos e **queremos criar um sistema simples para classificar animais que encontramos.**
- Nossa tarefa é ensinar **o computador a diferenciar Gatos de outros animais com base em duas características visuais que podemos medir facilmente por meio de imagens:** O formato aproximado do corpo e a textura do pelo.
- Coletando as informações e agora podemos **utilizar os algoritmos de Machine Learning** para a modelagem.

Comecemos com o KNN!

Modelos Para Classificação

KNN Classifier

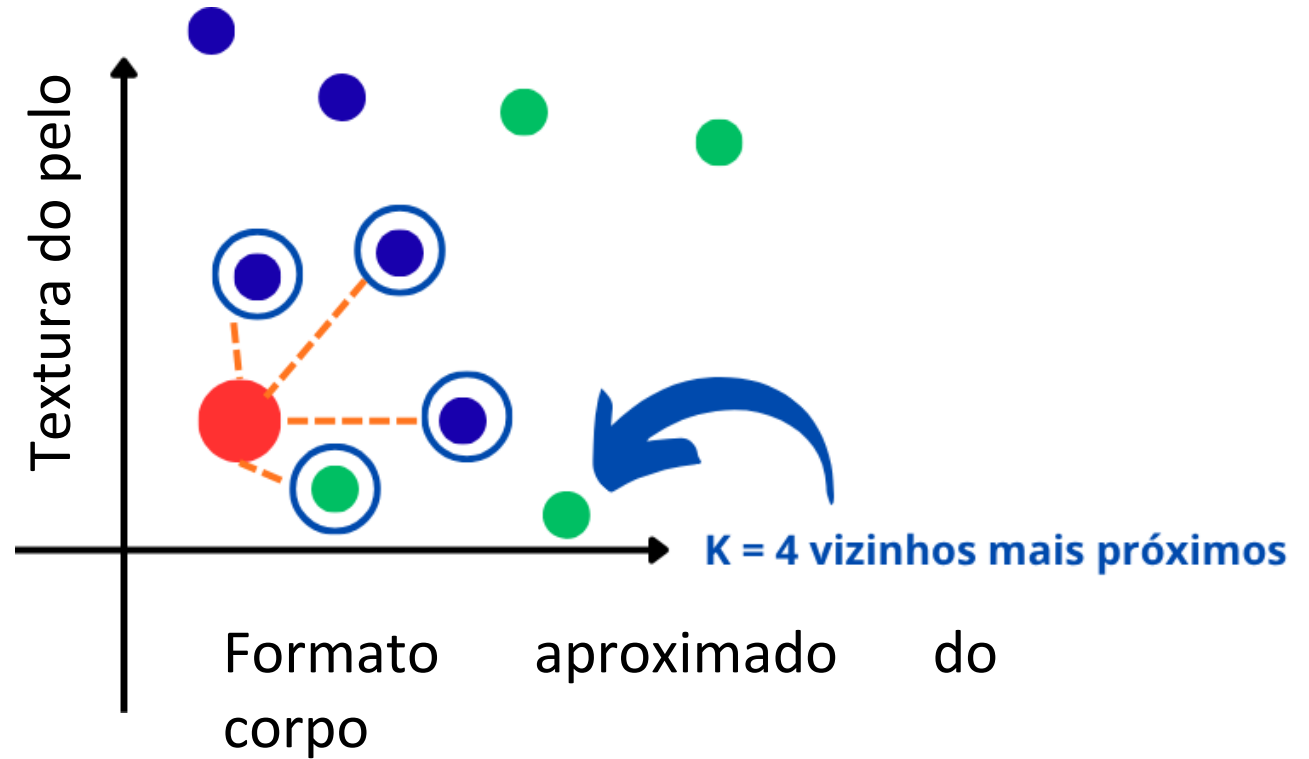
- Observe a imagem ao lado.
- Cada bolinha é um animal:
 - A bolinha **Azul** são os gatos.
 - A bolinha **Verde** são outros animais.
 - A bolinha **Vermelha** é o novo animal a ser classificado.
- **Quais os $k=4$ vizinhos mais próximos?**



Modelos Para Classificação

KNN Classifier

- Observe a imagem abaixo:
- Cada bolinha é um animal:
 - A bolinha **Azul** são os gatos.
 - A bolinha **verde** são outros animais.
 - A bolinha **vermelha** é o novo animal a ser classificado.
- **Quais os $k=4$ vizinhos mais próximos?**



- Portanto a nova observação será classificada como **gato**.

Modelos Para Classificação

KNN Classifier

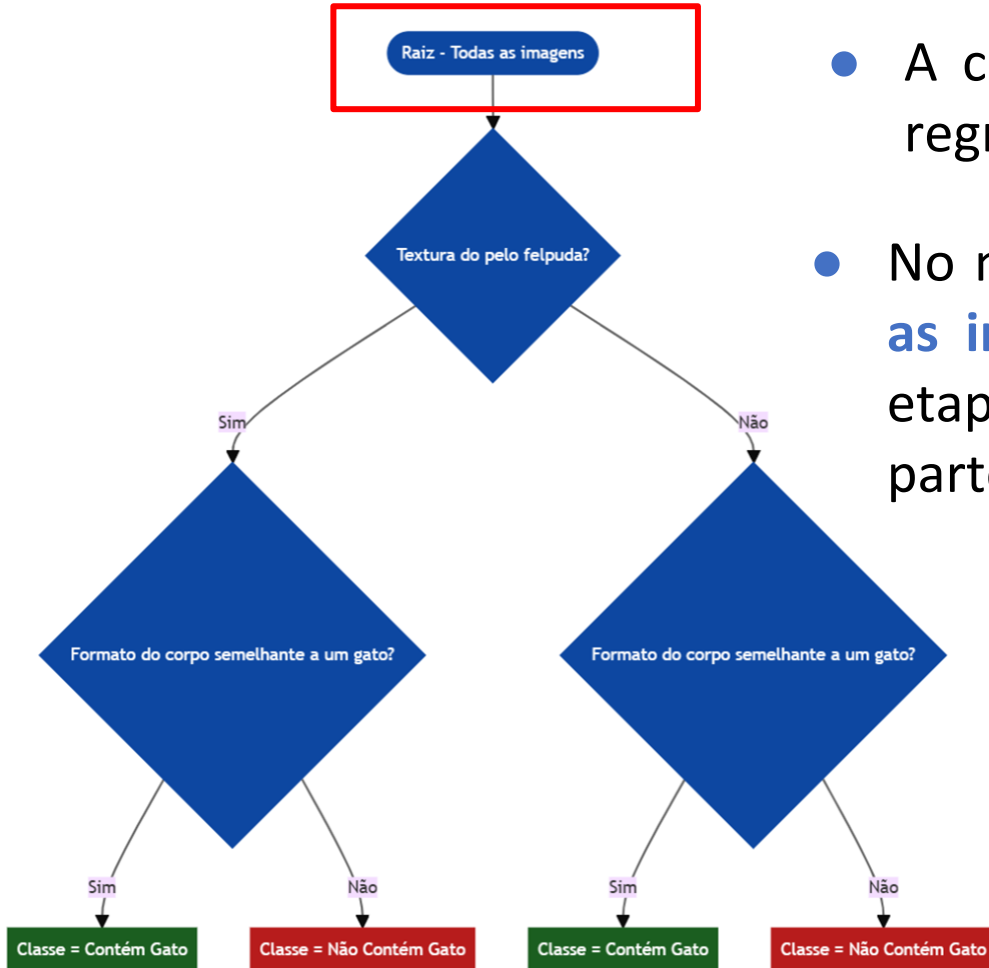
- Da mesma forma que na regressão, para prever o valor de um novo ponto, olhamos para os K vizinhos mais próximos (nos dados de treinamento) com base na categoria da maioria dos k vizinhos.

Exemplos de Hiperparâmetros

- **K:** Número de vizinhos a serem considerados.
 - **K pequeno:** Modelo mais flexível, mas mais sensível a ruído.
 - **K grande:** Modelo mais estável, mas pode ignorar detalhes importantes.
- **Métrica de Distância:** Como medir a "proximidade" entre os pontos.
 - **Ex: Distância Euclidiana, Distância de Manhattan.**
 - Para mais informações acesse a documentação [clcando aqui.](#)

Modelos Para Classificação

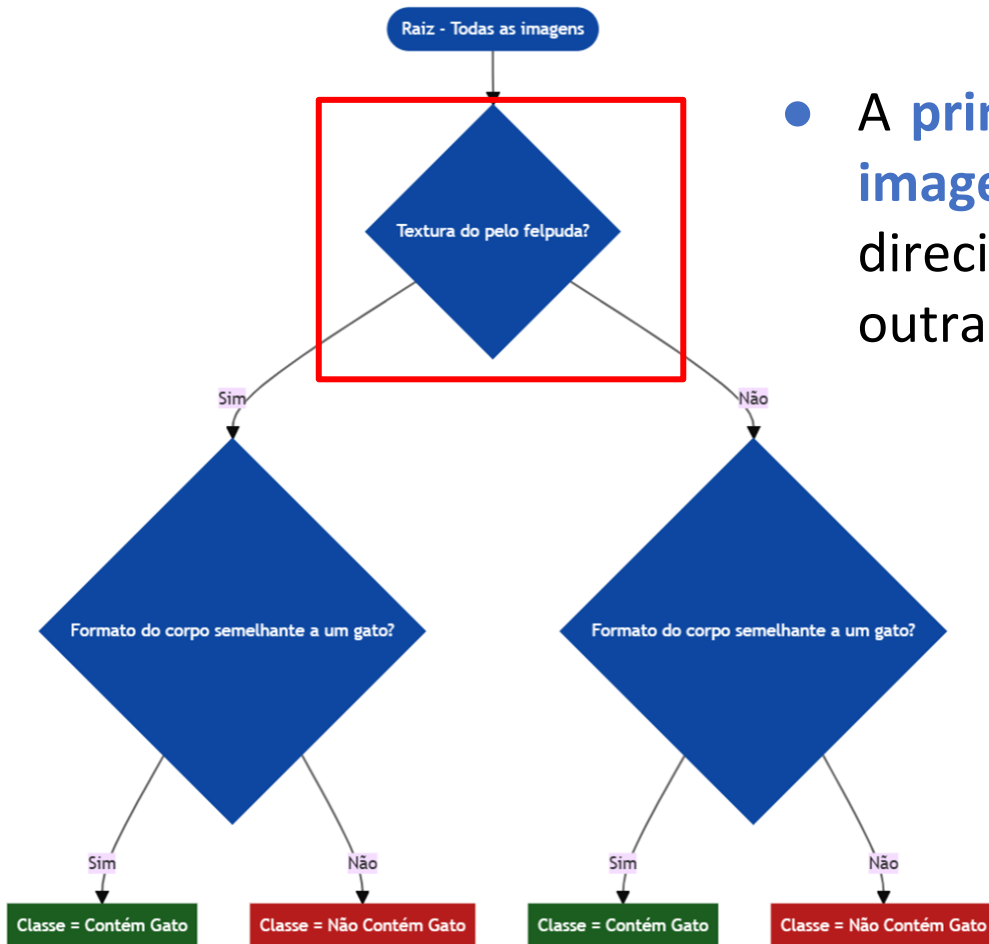
Árvore de Decisão



- A classificação se mantém na mesma ideia que vimos na regressão.
- No modelo “Gato-ou-Não”, **começamos considerando todas as imagens disponíveis**, formando a raiz da árvore. Nessa etapa inicial, não há distinção entre as imagens; todas fazem parte de um mesmo conjunto.

Modelos Para Classificação

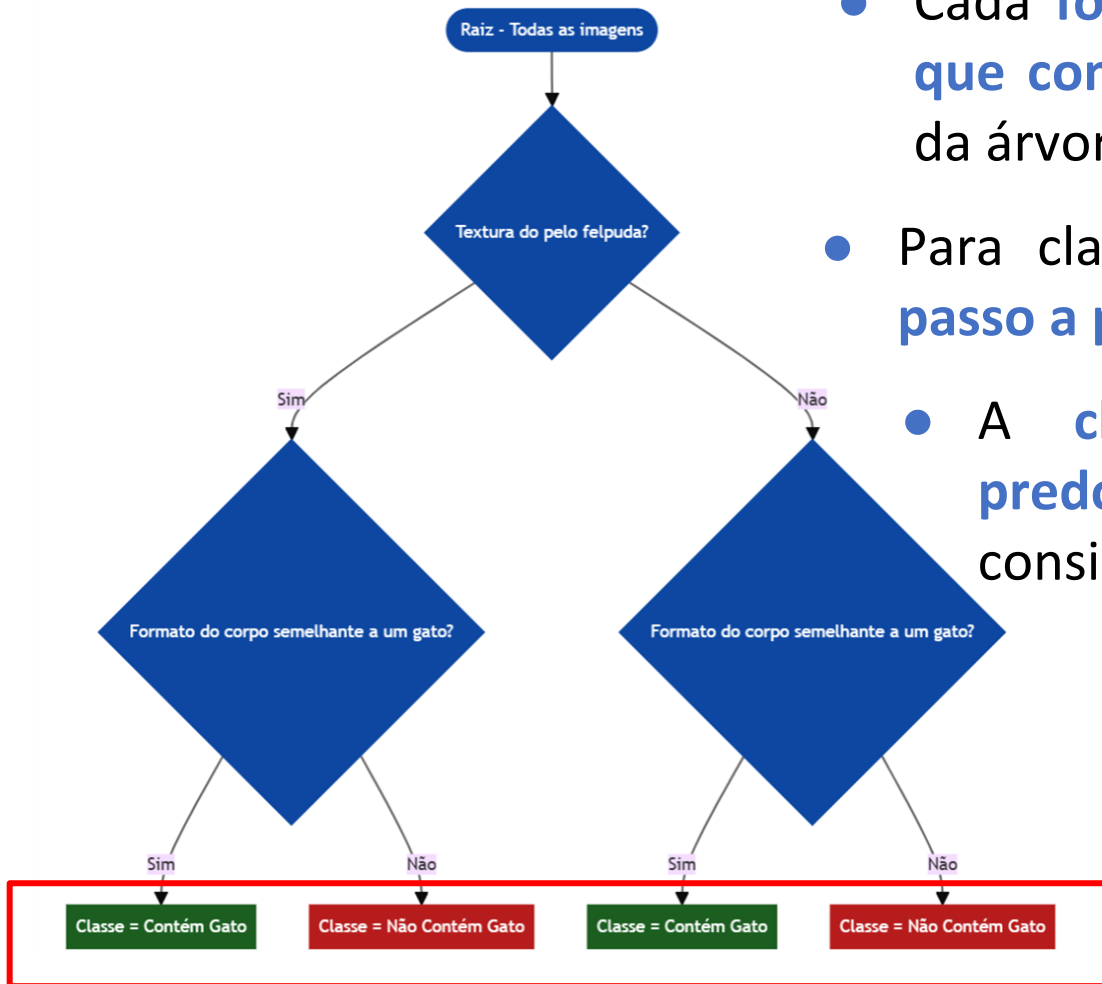
Árvore de Decisão



- A **primeira divisão ocorre com base na textura do pelo das imagens**. Se a textura do pelo for felpuda, a imagem é direcionada para uma sub-região específica da árvore, onde outras características serão analisadas.

Modelos Para Classificação

Árvore de Decisão



- Cada **folha da árvore representa um conjunto de imagens que compartilham padrões semelhantes** segundo as regras da árvore.
- Para classificar uma nova imagem, **seguimos essas regras passo a passo, desde a raiz até chegar à folha adequada.**
- A **classificação final é determinada pela classe predominante na folha em que a imagem caiu**, garantindo consistência com os dados observados.

Modelos Para Classificação

Exemplos de Hiperparâmetros:

Profundidade Máxima da Árvore: Controla a complexidade da árvore.

- **Árvores mais profundas:** Podem se ajustar melhor aos dados de treinamento, mas correm o risco de overfitting, pois podem se ajustar demais aos dados.
- **Árvores mais rasas:** São mais simples e fáceis de interpretar, mas podem ter baixo desempenho.

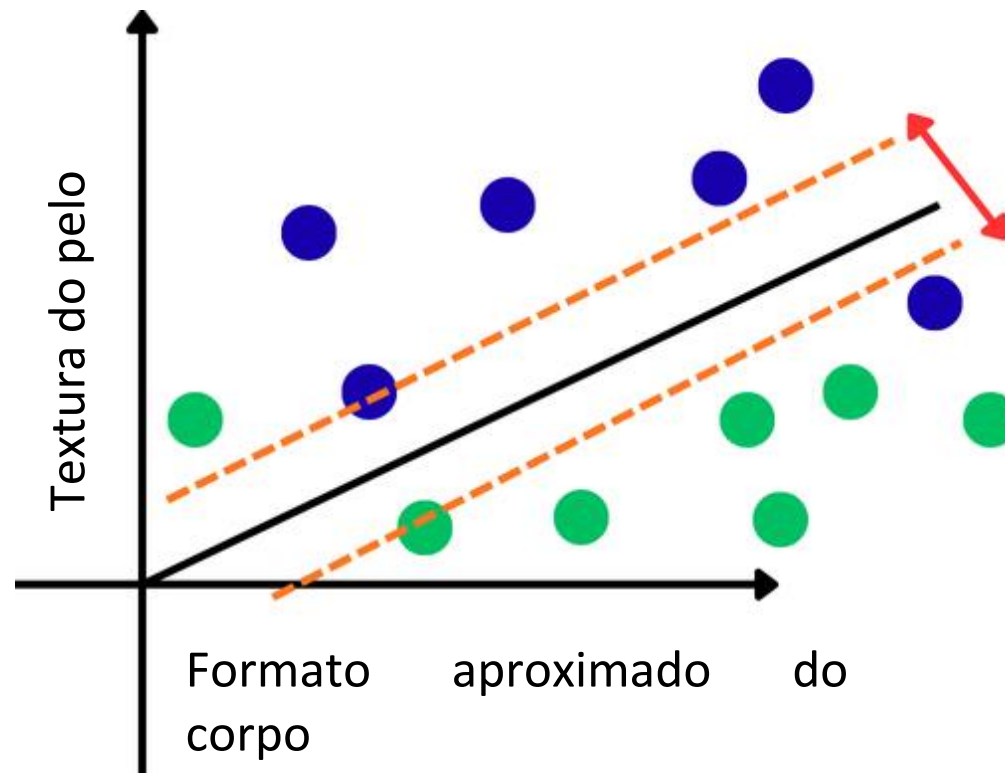
Número mínimo de amostras que um nó precisa ter para ser dividido.

- **Número pequeno:** Corre o risco de overfitting, mas pode se ajustar melhor aos dados.
- **Número grande:** Árvores mais simples, regiões grandes, risco de underfitting.

Modelos Para Classificação

Suport Vector Classifier -SVC

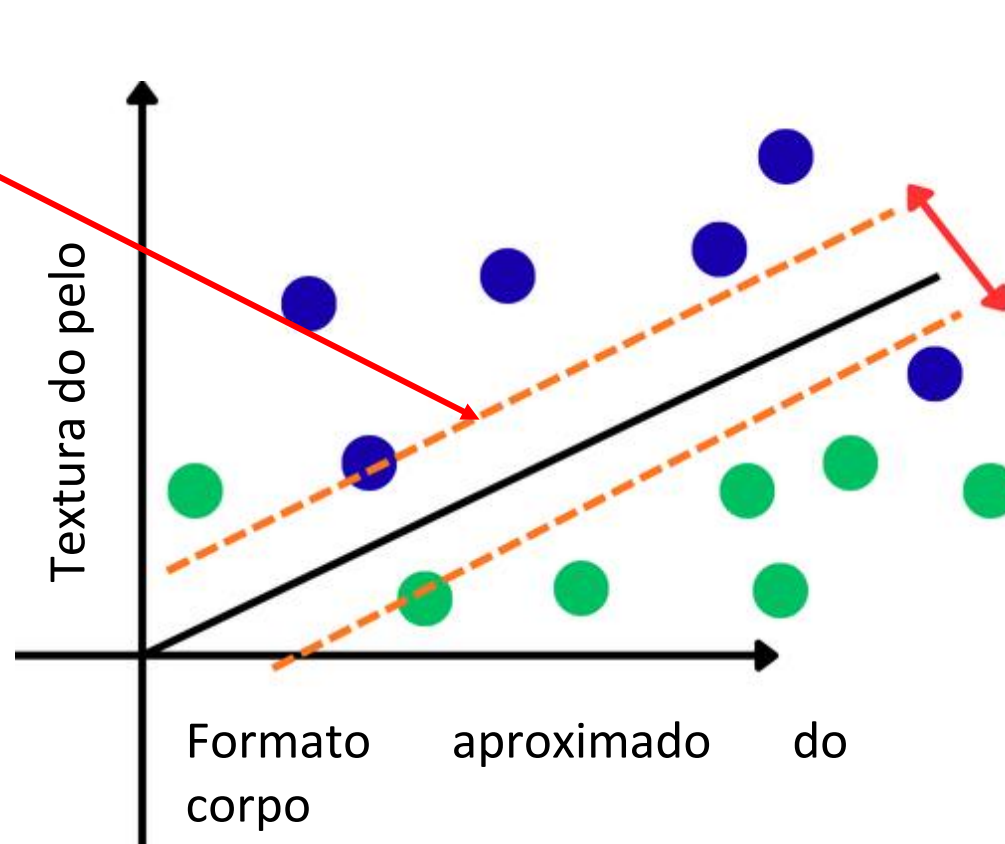
- O SVC é um **modelo de classificação cujo objetivo principal é encontrar a melhor fronteira possível para separar duas classes de dados**. Ele não está satisfeito em apenas desenhar qualquer linha que divida os grupo.



Modelos Para Classificação

Suport Vector Classifier -SVC

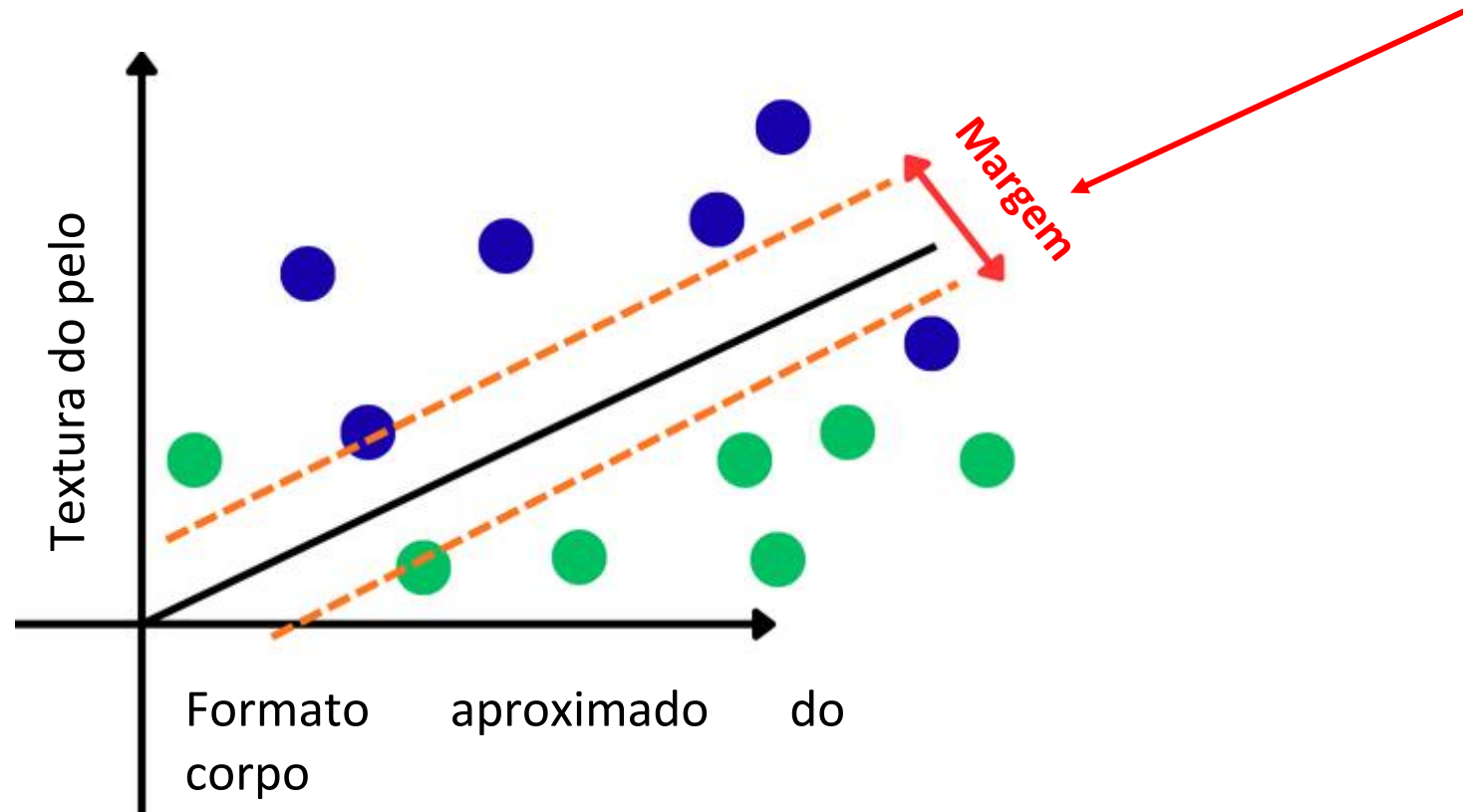
- A missão do SVM **é encontrar a divisão que seja a mais segura, robusta e confiante possível.** Ele busca a fronteira que cria a maior "zona de segurança" entre as classes.



Modelos Para Classificação

Suport Vector Classifier -SVC

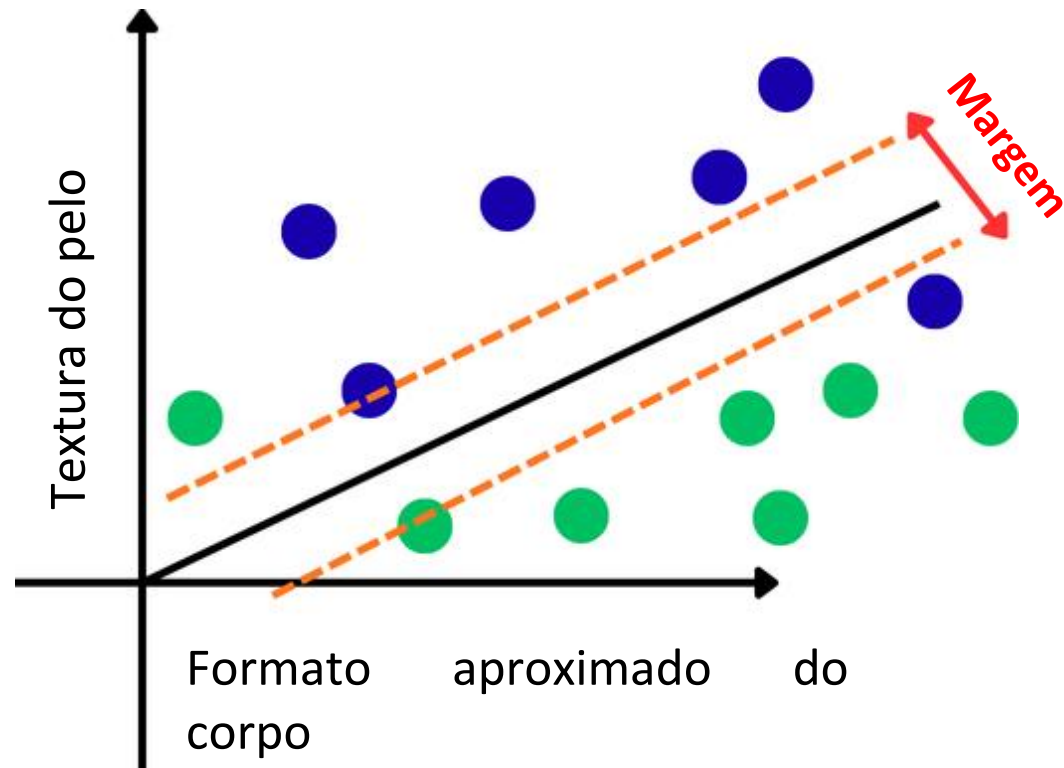
- Note que abaixo temos um exemplo de dados que são linearmente separáveis, mas o SVC não se limita só a esse problema.



Modelos Para Classificação

Suport Vector Classifier -SVC

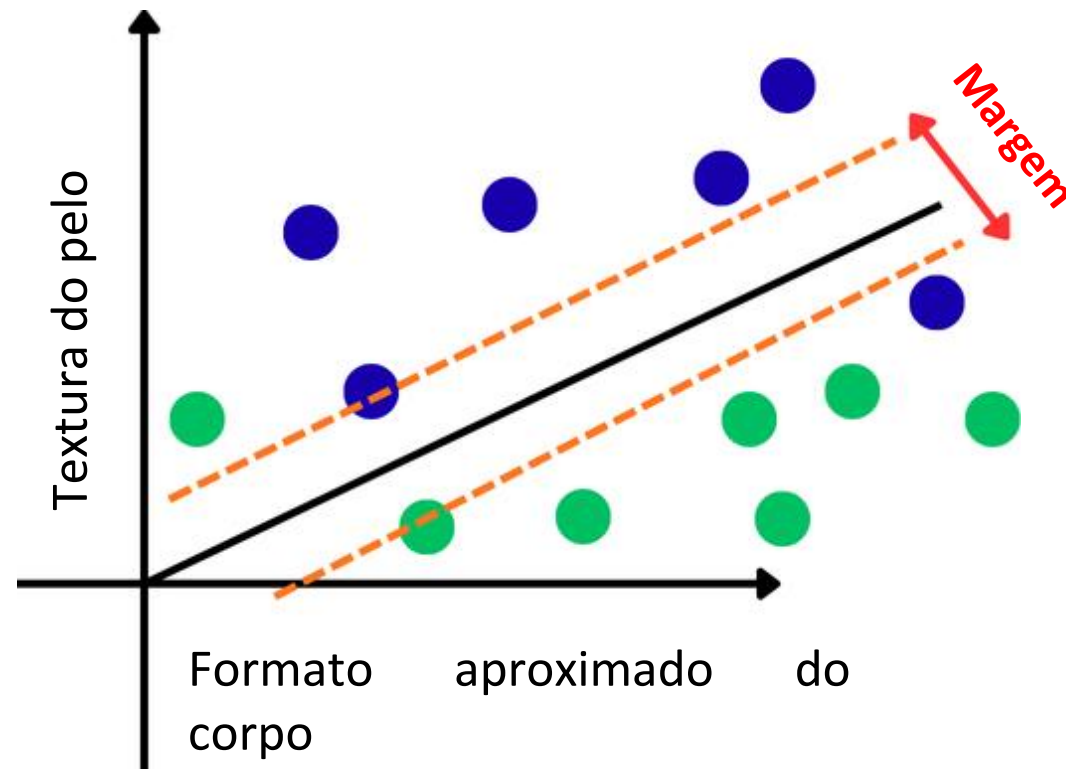
- Quando os dados não são linearmente separáveis (consegue dividir bem em duas fronteiras). O Kernel (Hiperparâmetro do modelo) vem para sanar essa problemática deixando linearmente separável.



Modelos Para Classificação

Support Vector Classifier -SVC

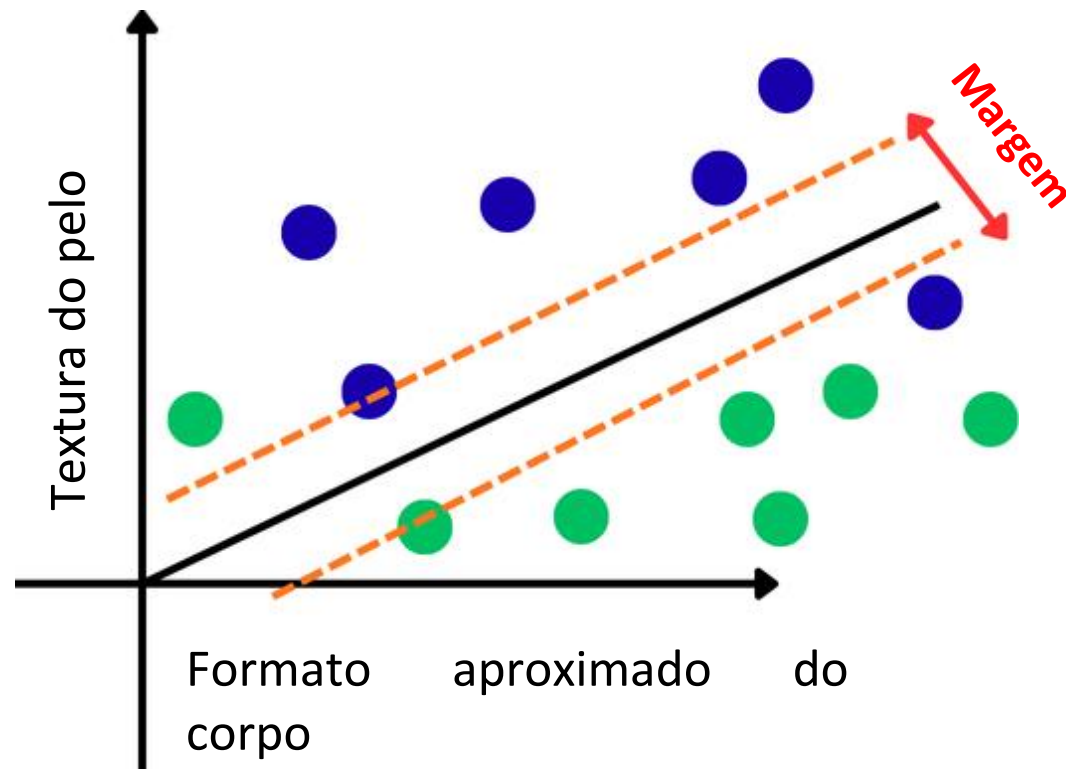
- A "mágica" do SVM está no conceito de margem. **A margem é a distância entre a linha de decisão central e os pontos de dados mais próximos de cada classe.** É a largura da nossa "avenida".



Modelos Para Classificação

Support Vector Classifier -SVC

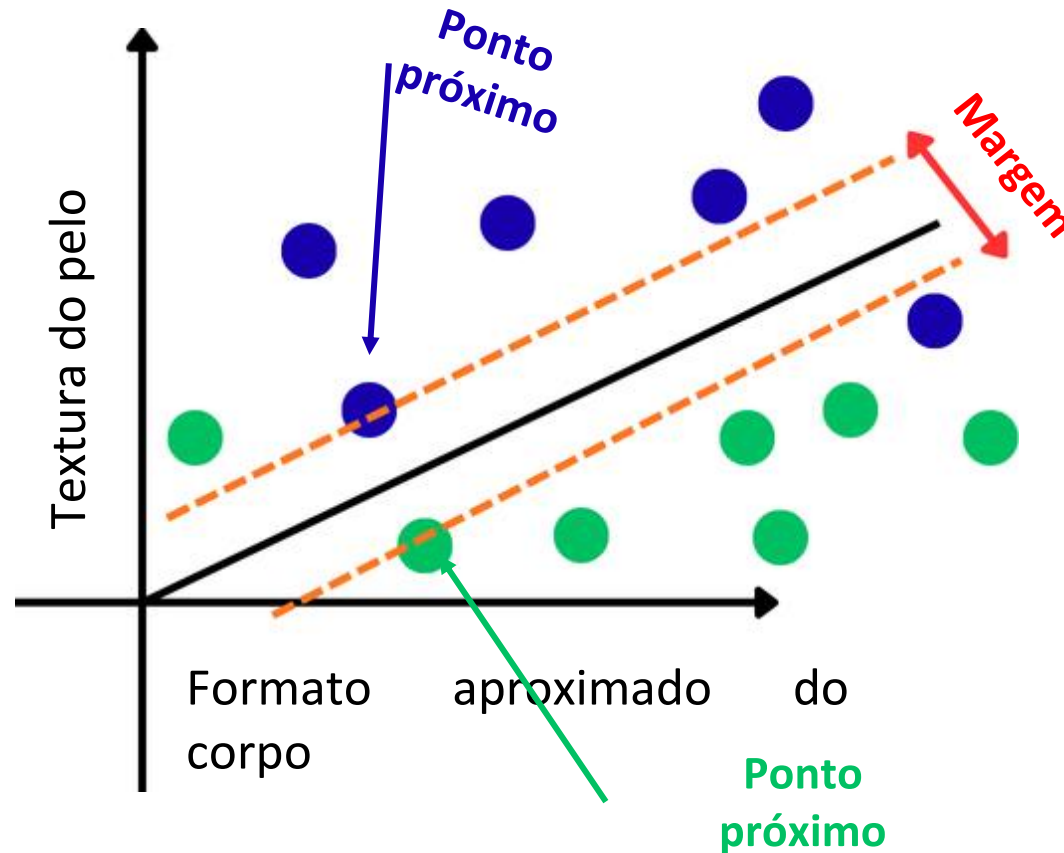
- O objetivo principal do SVM é **maximizar essa margem**. Ao criar a avenida mais larga possível, o modelo se torna mais generalizável e tem menos probabilidade de classificar erroneamente novos pontos de dados que possam aparecer perto da fronteira.



Modelos Para Classificação

Support Vector Classifier -SVC

- Os Vetores de Suporte **são os pontos de dados de cada classe que estão sendo utilizados como referência para o cálculo da margem.**



Modelos Para Classificação

Exemplos de Hiperparâmetros:

C (Parâmetro de Regularização): Controla a tolerância do modelo a erros de classificação nos dados de treino:

- **C baixo:** O modelo prioriza uma margem mais larga e uma fronteira mais simples, mesmo que isso signifique classificar incorretamente alguns pontos de treino. Reduz o risco de overfitting.
- **C alto:** O modelo tenta classificar corretamente o máximo de pontos de treino possível, resultando em uma fronteira mais complexa e uma margem mais estreita. Aumenta o risco de overfitting.

Modelos Para Classificação

Exemplos de Hiperparâmetros:

Kernel: Define a forma da fronteira de decisão, ou seja, a "ferramenta" usada para separar os dados.

- **Exemplos:** linear e rbf (padrão).

Gamma: Controla o alcance da influência de um único ponto de treino (usado principalmente com o kernel rbf).

- **gamma baixo:** A influência de cada ponto é ampla e de longo alcance, resultando em uma fronteira de decisão mais suave e generalizada.
- **gamma alto:** A influência de cada ponto é muito localizada e de curto alcance, fazendo com que a fronteira se ajuste de perto aos pontos individuais, o que pode levar ao overfitting.

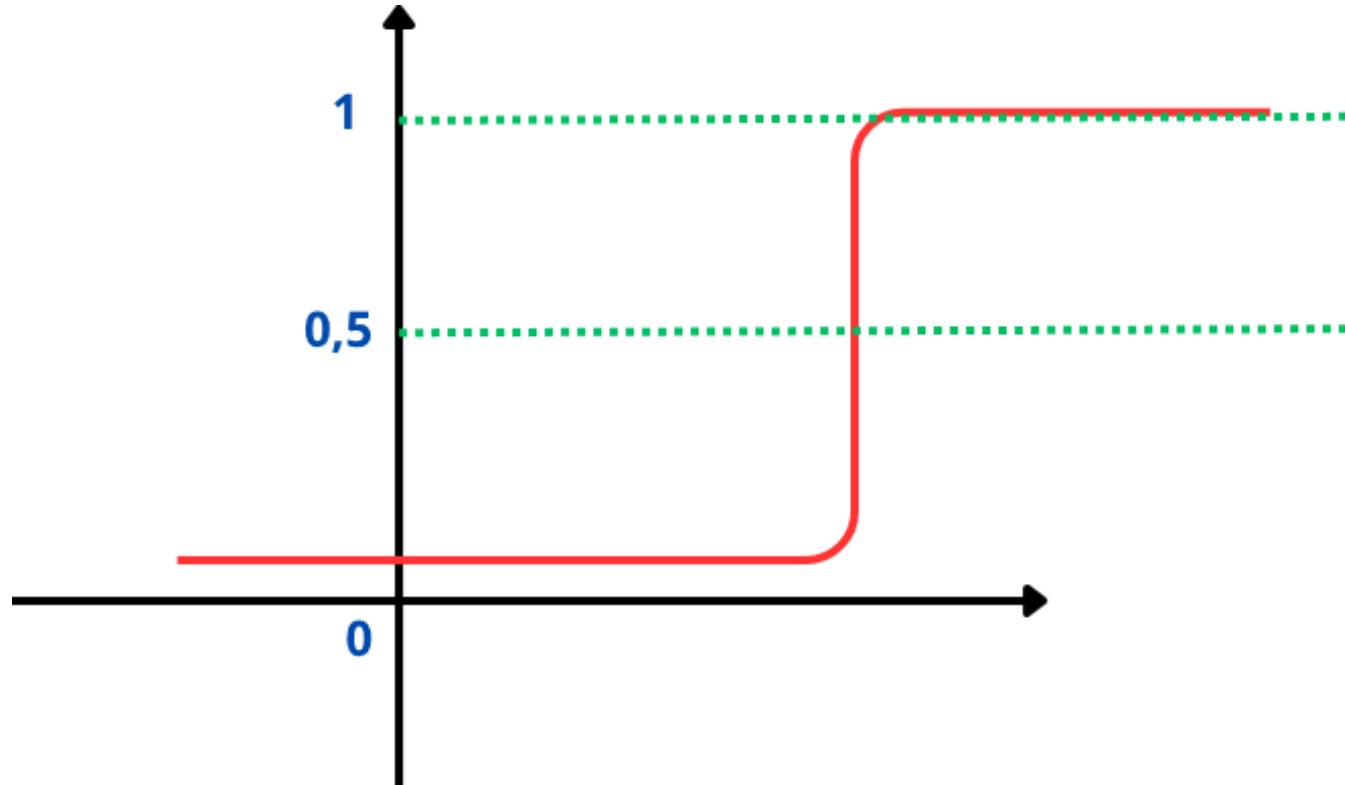
Modelos Para Classificação

Regressão Logística

- Apesar de ter "Regressão" no nome, a **Regressão Logística é um dos algoritmos mais populares e eficientes para classificação**, especialmente para problemas com duas categorias (classificação binária).
- Seu **objetivo é responder a perguntas diretas** de "Sim" ou "Não", como: "Este cliente vai comprar o produto?", "Este e-mail é spam?", ou "Esta transação é fraudulenta?".
- O nome "Regressão" **vem do fato de que, por baixo dos panos, ela começa calculando um score contínuo (muito parecido com a Regressão Linear)**, mas seu grande truque é transformar esse score em uma probabilidade, que é então usada para a classificação.

Modelos Para Classificação

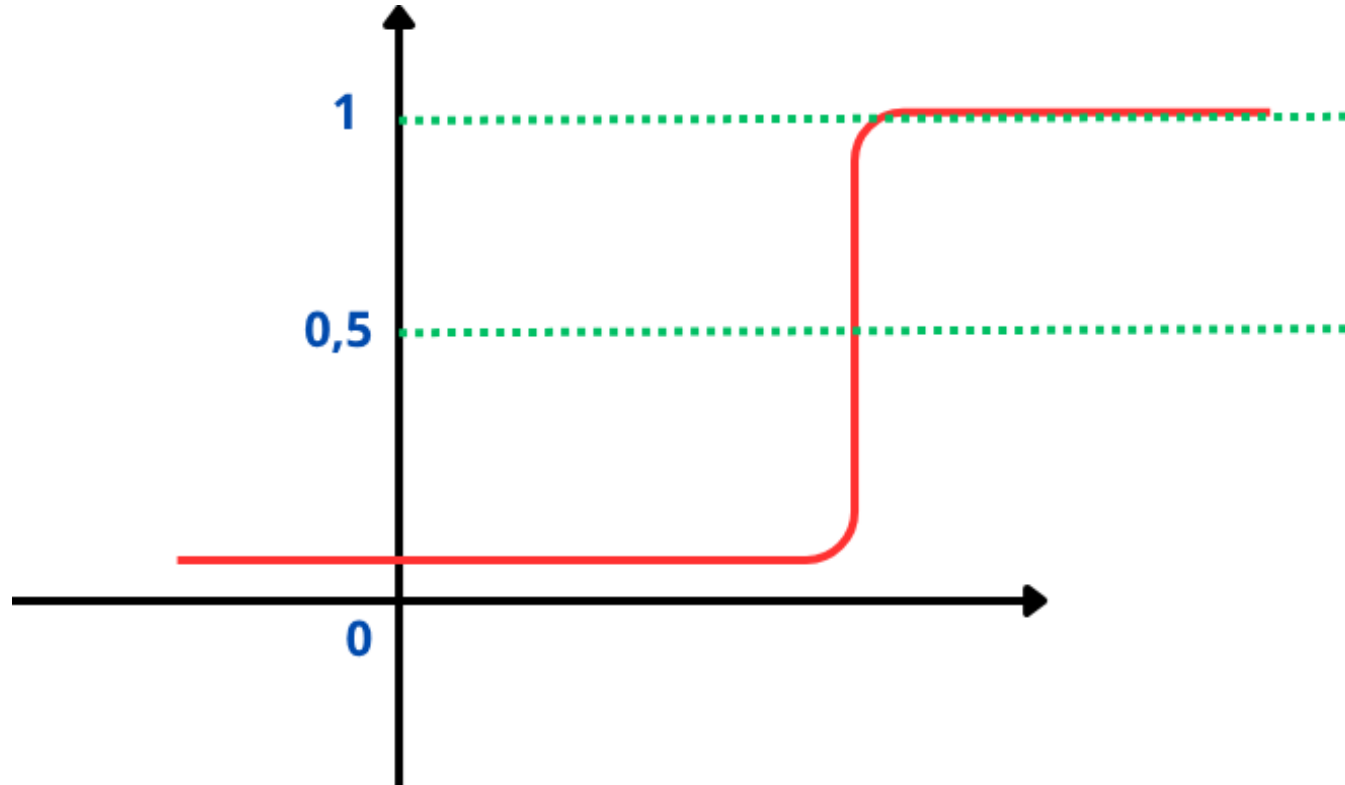
A Função Sigmoid



- O coração da Regressão Logística é uma função matemática chamada Função Sigmoid. Ela tem um formato característico de uma letra "S" e é a responsável pela "mágica" do modelo.

Modelos Para Classificação

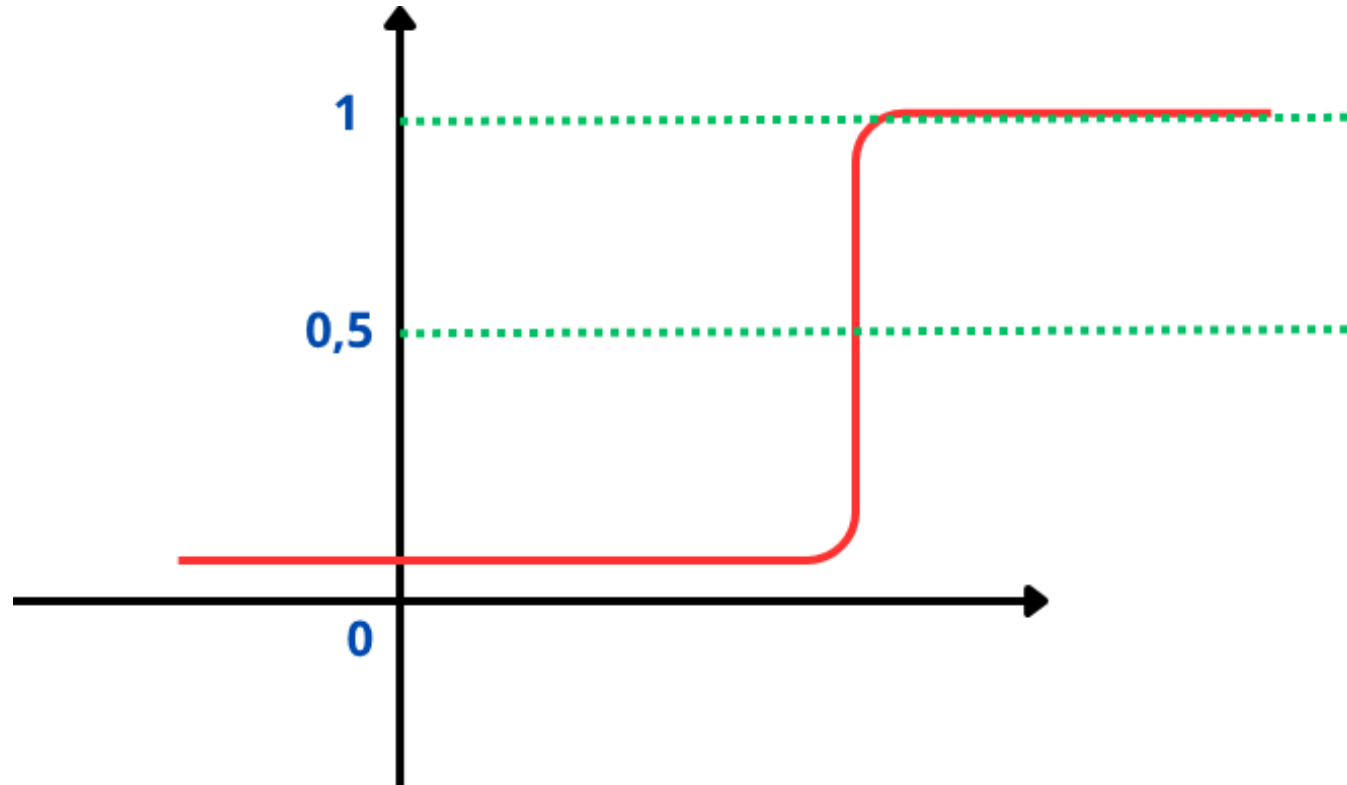
A Função Sigmoid



- A função sigmoide converte qualquer número real em um valor entre 0 e 1. Ela cresce suavemente: números muito negativos se aproximam de 0, números muito positivos se aproximam de 1, e valores perto de zero ficam próximos de 0,5.

Modelos Para Classificação

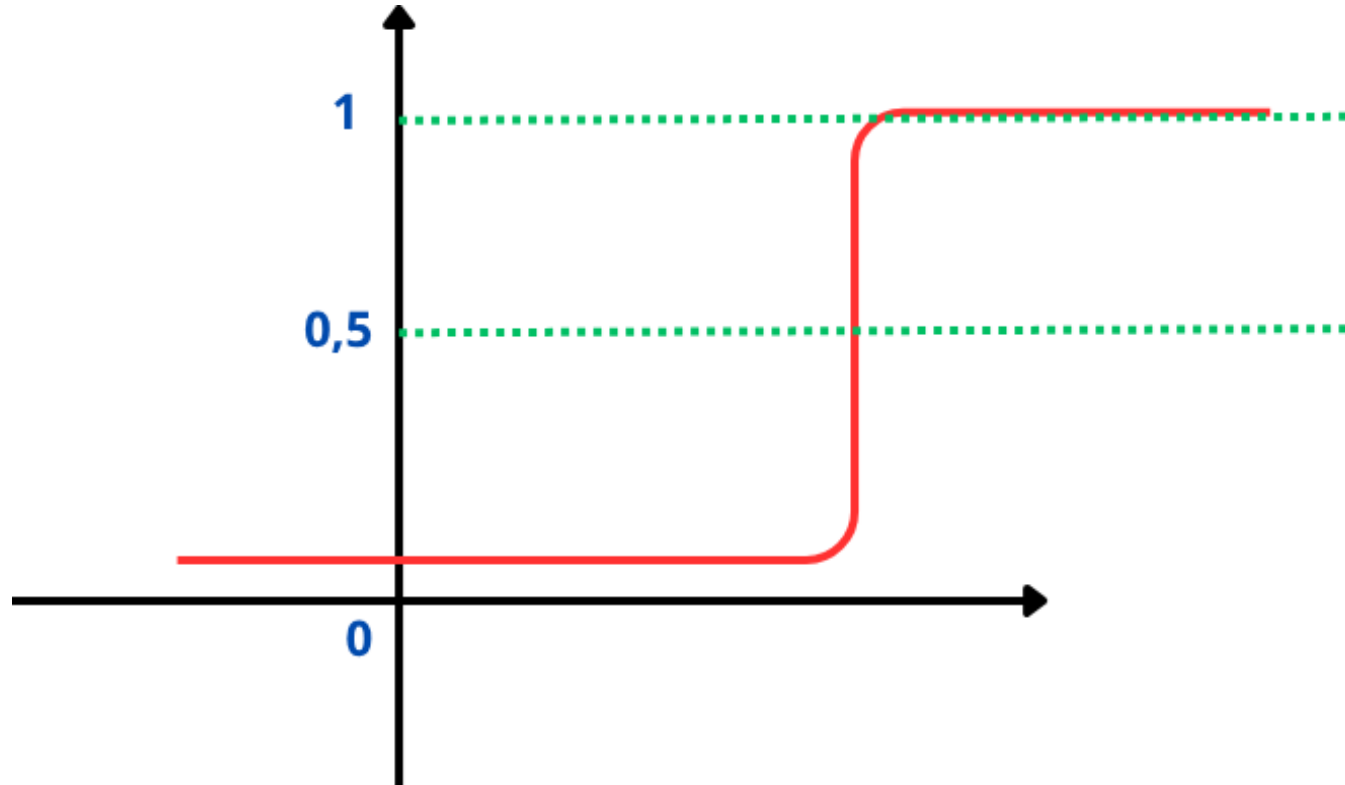
A Função Sigmoid



- O modelo nos entrega uma probabilidade (ex: "Existe 82% de chance de este e-mail ser spam") mas, para classificar, precisamos de uma resposta final ("É Spam" ou "Não é Spam"). Para isso, usamos um limite de corte (threshold).

Modelos Para Classificação

A Função Sigmoid



- O valor padrão para esse limite é 50% (ou 0.5). Se a probabilidade calculada pelo modelo for maior que o limite, ele classifica como a classe positiva ("Sim", "Spam", etc.). Se for menor, classifica como a classe negativa.

Modelos Para Classificação

Exemplos de Hiperparâmetros:

C (Regularização inversa): Controla a força da penalização aplicada aos coeficientes do modelo.

- **C baixo:** Mais regularização, os coeficientes tendem a ser menores, o que pode reduzir overfitting.
- **C alto:** Menos regularização, o modelo se ajusta mais aos dados de treino, podendo capturar mais padrões, mas também aumentando o risco de overfitting.

Solver (Algoritmo de otimização): Determina o método usado para encontrar os coeficientes que minimizam a função de custo.

- Ex: lbfgs, liblinear etc.

Modelos Para Classificação


Exemplos de Hiperparâmetros:

Penalty (Penalização): Define o tipo de regularização usada para evitar que os coeficientes fiquem muito grandes.

- **L2 (Ridge):** Penaliza o quadrado dos coeficientes, mantendo todos os coeficientes pequenos e distribuídos.
- **L1 (Lasso):** Penaliza o valor absoluto dos coeficientes, podendo zerar alguns deles, o que ajuda na seleção de variáveis.
- Para mais informações acesse a documentação [clikando aqui.](#)

Recapitulando: Dividindo os Dados da aula_01_exemplo_01

- Vamos criar um novo script para realizar todo o passo a passo de modelagem, mas primeiro vamos carregar nossos dados.



Código

```
[1]: import pandas as pd # manipulação de tabelas (DataFrames)
from sklearn.model_selection import train_test_split, GridSearchCV # divisão treino/teste + busca de hiperparâmetros
from sklearn.preprocessing import OneHotEncoder, StandardScaler # codificação categórica + padronização numérica
from sklearn.impute import SimpleImputer # preenchimento de valores faltantes
from sklearn.compose import ColumnTransformer # aplicar transformações diferentes em colunas
from sklearn.pipeline import Pipeline # encadear pré-processamento + modelo
from sklearn.linear_model import LogisticRegression # modelo de classificação binária
from sklearn.svm import SVC # modelo de classificação SVM
from sklearn.tree import DecisionTreeClassifier # classificador categórico do Decision Tree
from sklearn.neighbors import KNeighborsClassifier # classificador categórico do KNN
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report, matthews_corrcoef # métricas de avaliação do modelo
import numpy as np # operações numéricas e matrizes
import matplotlib.pyplot as plt # criação de gráficos
from sklearn.metrics import ConfusionMatrixDisplay, RocCurveDisplay # visualizar a curva roc e a matriz de confusão

# Carregando dataset
df = pd.read_csv("aula_01_exemplo_01.csv") # lê o arquivo aula_01_exemplo_01.csv em um DataFrame


df['tem_filhos'] = (df['children'] > 0).astype(int) # Cria uma nova coluna 'tem_filhos' que indica se a pessoa tem filhos (1) ou não (0)
# (df['children'] > 0) cria uma Series booleana (True/False)
# .astype(int) converte True para 1 e False para 0
```



Aqui, adicionaremos os comandos para conseguir manipular os modelos de classificação e as novas métricas de avaliações que aprendemos nessa aula!

Recapitulando: Dividindo os Dados da aula_01_exemplo_01

- Vamos criar um novo script para realizar todo o passo a passo de modelagem, mas primeiro vamos carregar nossos dados.



Código

```
[1]: import pandas as pd # manipulação de tabelas (DataFrames)
from sklearn.model_selection import train_test_split, GridSearchCV # divisão treino/teste + busca de hiperparâmetros
from sklearn.preprocessing import OneHotEncoder, StandardScaler # codificação categórica + padronização numérica
from sklearn.impute import SimpleImputer # preenchimento de valores faltantes
from sklearn.compose import ColumnTransformer # aplicar transformações diferentes em colunas
from sklearn.pipeline import Pipeline # encadear pré-processamento + modelo
from sklearn.linear_model import LogisticRegression # modelo de classificação binária
from sklearn.svm import SVC # modelo de classificação SVM
from sklearn.tree import DecisionTreeClassifier # classificador categórico do Decision Tree
from sklearn.neighbors import KNeighborsClassifier # classificador categórico do KNN
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report, matthews_corrcoef # métricas de avaliação do modelo
import numpy as np # operações numéricas e matrizes
import matplotlib.pyplot as plt # criação de gráficos
from sklearn.metrics import ConfusionMatrixDisplay, RocCurveDisplay # visualizar a curva roc e a matriz de confusão

# Carregando dataset
df = pd.read_csv("aula_01_exemplo_01.csv") # lê o arquivo aula_01_exemplo_01.csv em um DataFrame

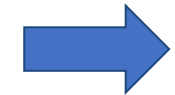
df['tem_filhos'] = (df['children'] > 0).astype(int) # Cria uma nova coluna 'tem_filhos' que indica se a pessoa tem filhos (1) ou não (0)
# (df['children'] > 0) cria uma Series booleana (True/False)
# .astype(int) converte True para 1 e False para 0
```


Recapitulando: Dividindo os Dados da aula_01_exemplo_01

- Agora faremos o seguinte, vamos categorizar nossa variável target, isto é transformar a variável que é contínua, para categórica!
- **Por que fazer isso?**
- **Facilidade de interpretação:** Categorizar transforma números contínuos em grupos simples, como "custos altos" e "custos baixos". Isso facilita a leitura e comunicação para gestores ou públicos não técnicos, que podem não estar familiarizados com a análise de valores numéricos.
- **Comparação entre grupos :** Com categorias, é mais fácil comparar perfis, como verificar a idade média dos que têm custos altos versus baixos.
- **Desvantagem:** Ao transformar valores contínuos em categorias, você acaba simplificando o conjunto de dados e muitos detalhes que poderiam virar algum insight valioso pro modelo acaba se perdendo.

Recapitulando: Dividindo os Dados da aula_01_exemplo_01

- Vamos categorizar nossa variável target pela mediana, assim metade das nossas observações estarão em um grupo e metade no outro.



Código

```
[2]: # Calcula a mediana de 'charges'
      mediana_charges = df['charges'].median()

      print("Valor da Mediana:", mediana_charges)

      # Cria coluna binária: 1 se charges >= mediana (custos altos), 0 caso contrário (custos baixos)
      df['custos_categoricos'] = (df['charges'] >= mediana_charges).astype(int)

      # Exibe as primeiras linhas para conferir
      df[['charges', 'custos_categoricos']].head()
```



Saída

```
Valor da Mediana: 9382.033
[2]:
```

	charges	custos_categoricos
0	16884.92400	1
1	1725.55230	0
2	4449.46200	0
3	21984.47061	1
4	3866.85520	0

- O comando `.median()` vai calcular a mediana da nossa variável target **charges**.
- Como **para calcular a mediana é necessário ordenar nossos dados de custos do menor até o maior**, conseguiremos fazer com que metade dos indivíduos tenha o valor da categoria 1 (custos altos) e a outra metade 0 (custos baixos).

Recapitulando: Dividindo os Dados da aula_01_exemplo_01

- Vamos categorizar nossa variável target pela mediana, assim metade das nossas observações estarão em um grupo e metade no outro.



Código

```
[2]: # Calcula a mediana de 'charges'
      mediana_charges = df['charges'].median()

      print("Valor da Mediana:", mediana_charges)

      # Cria coluna binária: 1 se charges >= mediana (custos altos), 0 caso contrário (custos baixos)
      df['custos_categoricos'] = (df['charges'] >= mediana_charges).astype(int)

      # Exibe as primeiras linhas para conferir
      df[['charges', 'custos_categoricos']].head()
```



Saída

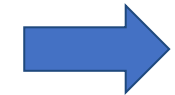
```
Valor da Mediana: 9382.033
[2]:
```

	charges	custos_categoricos
0	16884.92400	1
1	1725.55230	0
2	4449.46200	0
3	21984.47061	1
4	3866.85520	0

- O comando `(df['charges'] >= mediana_charges)` cria uma nova coluna cujo os valores são booleanos, ou seja True e False.

Recapitulando: Dividindo os Dados da aula_01_exemplo_01

- Vamos categorizar nossa variável target pela mediana, assim metade das nossas observações estarão em um grupo e metade no outro.



Código

```
[2]: # Calcula a mediana de 'charges'
      mediana_charges = df['charges'].median()

      print("Valor da Mediana:", mediana_charges)

      # Cria coluna binária: 1 se charges >= mediana (custos altos), 0 caso contrário (custos baixos)
      df['custos_categoricos'] = (df['charges'] >= mediana_charges).astype(int)

      # Exibe as primeiras linhas para conferir
      df[['charges', 'custos_categoricos']].head()
```



Saída

```
Valor da Mediana: 9382.033
[2]:
```

	charges	custos_categoricos
0	16884.92400	1
1	1725.55230	0
2	4449.46200	0
3	21984.47061	1
4	3866.85520	0

- O comando `.astype(int)` vai converter esses valores para inteiro, ou seja 1 para True (custos altos) e 0 para False (custos baixos).
- **ATENÇÃO:** Sempre converta sua variável target categórica em 0 e 1. O python tem problemas com variáveis que estão no formato de String!

Recapitulando: Dividindo os Dados da aula_01_exemplo_01

- Como estão nossas categorias? vamos visualizar?



Código

```
[3]: df["custos_categoricos"].value_counts()
```



Saída

```
[3]: custos_categoricos  
1    669  
0    669  
Name: count, dtype: int64
```

- Note que temos classes bem equilibradas graças a mediana.
- Geralmente **não teremos essa característica** nos nossos dados, o que pode prejudicar na hora de dividir os dados em treino e teste.

Recapitulando: Dividindo os Dados da aula_01_exemplo_01

- Como estão nossas categorias? vamos visualizar?



Código

```
[3]: df["custos_categoricos"].value_counts()
```



Saída

```
[3]: custos_categoricos  
1    669  
0    669  
Name: count, dtype: int64
```

- Imagine que invés de classes equilibradas, tivéssemos 800 para 1 e 200 para 0.
- Esse tipo **de disparidade prejudica a capacidade do modelo de generalizar** e reconhecer corretamente novos exemplos na base de teste.
- Em situações extremas, **o modelo pode até ignorar uma classe inteira, simplesmente porque não teve representatividade suficiente para aprender.**

Recapitulando: Dividindo os Dados da aula_01_exemplo_01

- Como estão nossas categorias? vamos visualizar?



Código

```
[3]: df["custos_categoricos"].value_counts()
```



Saída

```
[3]: custos_categoricos  
1    669  
0    669  
Name: count, dtype: int64
```

- Para resolver esse problema, é utilizado o conceito de **estratificação**.
- A estratificação vai **preservar a proporção de classes para cada uma das bases de dados**.
- No exemplo **anterior**, vimos que se tivéssemos a **disparidade de 800 observações com classe 1 e 200 para classe 0**. Como temos 1000 observações, para a classe 1 teríamos um percentual de 80% e para classe 2 teríamos 20%.

Recapitulando: Dividindo os Dados da aula_01_exemplo_01

- Como estão nossas categorias? vamos visualizar?



Código

```
[3]: df["custos_categoricos"].value_counts()
```



Saída

```
[3]: custos_categoricos  
     1      669  
     0      669  
     Name: count, dtype: int64
```

- Assim **quando dividirmos os dados em treino e teste, esse percentual seria alocado para a base de treino**, isto é 80% das observações na base de treino seria da classe 1 e 20% seria da classe 0.
- Da mesma forma que no treino, o teste teria as mesmas proporções de 80% para classe 1 e 20% para a classe 0.

Recapitulando: Dividindo os Dados da aula_01_exemplo_01

- Como estão nossas categorias? vamos visualizar?



Código

```
[3]: df["custos_categoricos"].value_counts()
```



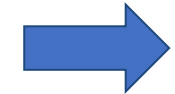
Saída

```
[3]: custos_categoricos  
1    669  
0    669  
Name: count, dtype: int64
```

- Portanto é de suma importância que o processo de estratificação seja feito nos dados quando há classes desbalanceadas.
- A implementação é simples, na etapa de divisão dos dados, basta especificar um novo parâmetro chamado **stratify** e o próprio python lida com a divisão estratificada.

Recapitulando: Dividindo os Dados da aula_01_exemplo_01

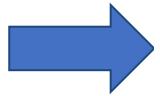
- Vamos deixar claro qual a variável target e quais são as preditoras.



Código

```
[4]: # Separando as covariáveis (X) e target (y)
x = df.drop(["custos_categoricos", "charges"], # drop(["custos_categoricos", "charges"], axis=1) remove a coluna 'custos_categoricos' e 'charges';
            axis=1) # retorna um novo DataFrame sem 'custos_categoricos'
y = df["custos_categoricos"] # seleciona a coluna 'custos_categoricos' como Series – esta é a variável alvo que queremos prever

x.head()
```



Saída

```
[4]:
```

	age	sex	bmi	children	smoker	region	tem_filhos
0	19	female	27.900	0	yes	southwest	0
1	18	male	33.770	1	no	southeast	1
2	28	male	33.000	3	no	southeast	1
3	33	male	22.705	0	no	northwest	0
4	32	male	28.880	0	no	northwest	0

- Note que dessa vez, retiramos a variável continua “charges” da modelagem. Assim evitamos vazamento de dados.

Recapitulando: Dividindo os Dados da aula_01_exemplo_01

- Agora vamos deixar claro quais são as variáveis categóricas e quais são as numéricas.


Código

```
[5]: # Definição explícita das colunas por tipo
variaveis_categoricas = ["sex", "smoker", "region"] # listas com nomes das colunas categóricas (devem bater exatamente com os nomes do DataFrame)
variaveis_numericas = ["age", "bmi", "children", "tem_filhos"] # lista das colunas numéricas que serão escalonadas
```

- Perceba que não incluímos a variável target, já que fazemos manipulações de padronizar e transformar apenas nas covariáveis.

O código acima não retorna nada pra gente!

Recapitulando: Dividindo os Dados da aula_01_exemplo_01

- Vamos dividir nossos dados em dados de treinamento e dados de teste. Usaremos o comando `train_test_split()`.


Código

```
[6]: # Divisão treino/teste

x_treino, x_teste, y_treino, y_teste = train_test_split( # função para divisão dos dados
    X,                                                  # covariáveis
    y,                                                  # variável target
    stratify = y,                                     # 20% para teste
    test_size=0.2,                                     # Reprodutibilidade, como essa função aleatoriza os dados, vamos fixar essa aleatorização
    random_state=42
)

print("número de linhas e colunas da base de treino:",x_treino.shape)
print("número de linhas e colunas da base de teste:",x_teste.shape)
```


Saída

```
número de linhas e colunas da base de treino: (1070, 7)
número de linhas e colunas da base de teste: (268, 7)
```

- Temos 1070 linhas (ou indivíduos) na base de treino e 268 linhas na base de teste.
- Note que agora adicionamos o parâmetro `stratify = y` para especificar que eu quero que ele estratifique. Para modelagens futuras, é uma ótima opção já deixar implementado.

Recapitulando: Preparando os Dados da aula_01_exemplo_01

- Vamos recriar cada processo de imputação e transformação de dados.



Código

```
[5]: # Pré-processamento

escalador = StandardScaler() # Escalonador, ele vai padronizar as variáveis numéricas

categorizador = OneHotEncoder(drop="first", # O categorizador vai Remover a primeira dummy para evitar redundância
                               handle_unknown="ignore") # Ignora categorias desconhecidas em dados de teste

imputador_numerico = SimpleImputer(strategy="median") # imputador numérico usando mediana

imputador_categorico = SimpleImputer(strategy="most_frequent") # imputador categórico usando a moda
```

O código acima não retorna nada pra gente!

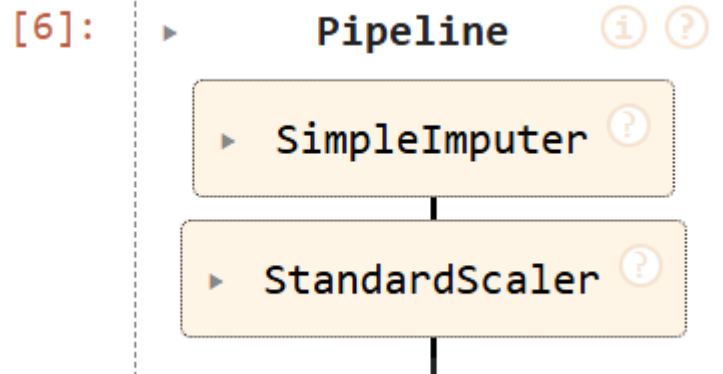
Recapitulando: Preparando os Dados da aula_01_exemplo_01

- Vamos agora recriar a criação das etapas numéricas e categóricas no python.

→
Código

```
[6]: # Etapas de transformação das variáveis numéricas
etapas_numericas = Pipeline(
    [
        ("imputer", imputador_numerico),    # "imputer" é o nome da etapa → aplica a imputação (substitui valores ausentes pela mediana)
        ("scaler", escalonador)             # "scaler" é o nome da etapa → aplica padronização
    ]
)
etapas_numericas
```

→
Saída



- Criando a etapa numérica, nesse caso não irá mudar nada para a abordagem de classificação.

Recapitulando: Preparando os Dados da aula_01_exemplo_01

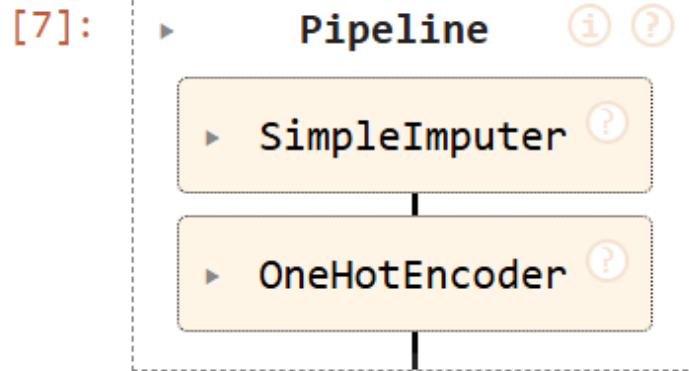
- Vamos agora recriar a criação das etapas numéricas e categóricas no python.

➔
Código

```
[7]: # Etapas de transformação das variáveis categóricas
etapas_categoricas = Pipeline(
    [
        ("imputer", imputador_categorico), # "imputer" é o nome da etapa → aplica a imputação (substitui valores ausentes pela moda)
        ("encoder", categorizador)        # "encoder" é o nome da etapa → aplica OneHotEncoder
    ]
)

etapas_categoricas
```

➔
Saída



- Criando a etapa categórica. Da mesma forma que o anterior, não irá mudar nada o processo de transformação e imputação.

Recapitulando: Preparando os Dados da aula_01_exemplo_01

- Agora vamos juntar tudo novamente.


Código

```
[8]: # Juntando todo o processamento das variáveis categóricas e numéricas
preprocessador = ColumnTransformer(
    [
        ("num", etapas_numericas, variaveis_numericas), # Nome da etapa, Escalonador e a lista de variáveis numéricas
        ("cat", etapas_categoricas, variaveis_categoricas) # Nome da etapa, categorizador e a lista de variáveis categóricas
    ]
)

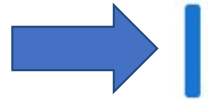
x_treino_transformado = preprocessador.fit_transform(x_treino) # aplicando a imputação e transformação nos dados de treino
x_teste_transformado = preprocessador.transform(x_teste) # aplicando a imputação e transformação nos dados de teste
```

- O **ColumnTransformer()** ainda vai servir como um processo de união entre as etapas categóricas e contínuas, mesmo no processo de classificação.

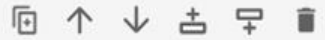
O código acima não retorna nada pra gente!

Modelando os Dados da aula_01_exemplo_01

- Agora vamos criar os novos comandos para inserir os modelos que aprendemos nessa aula. Os comandos anteriores são repetições da aula passada e já foram escritos previamente, basta rodá-los.
- Vamos criar o modelos de classificação. Começaremos pela árvore de decisão.



```
[13]: # criando o modelo  
modelo = DecisionTreeClassifier(random_state=42) # criando o modelo e fixando a aleatorização
```

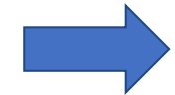


Código

- Criamos nosso modelo e armazenamos ele em uma variável chamada **modelo**.

Modelando os Dados da aula_01_exemplo_01

- Agora vamos criar nossa grade de hiperparâmetros.



Código

```
[10]: # Grade de hiperparâmetros

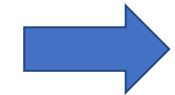
param_grid = {
    "max_depth": [2,3,5,7], # Profundidade máxima da árvore
    "min_samples_split": [2,5,10,15,20,25]# Nº mínimo de amostras para dividir um nó
}
```

- Note que são os mesmos hiperparâmetros que vimos: “**max_depth**” é a profundidade máxima da árvore e o “**min_samples_split**” é Número mínimo de amostras que um nó precisa ter para ser dividido.

O código acima não retorna nada pra gente!

Modelando os Dados da aula_01_exemplo_01

- Agora vamos usar o Grid Search para encontrar os melhores hiperparâmetros.



Código


```
[15]: # Configurando Grid Search
      grid_search = GridSearchCV(
          estimator= modelo,          # modelo a ser otimizado
          param_grid=param_grid,      # Grade de hiperparâmetros
          cv=5,                       # 5-fold cross-validation
          scoring="f1" # Métrica de comparação: F1-SCORE
      )
```



- Vamos utilizar a **métrica F1-Score** como métrica principal para escolher os melhores hiperparâmetros do modelo.
- **Escolhemos apenas uma métrica selecionar o modelo pois poderia ter conflitos de métrica!** Lembre-se que o precision e o recall são inversamente proporcionais, logo faz mais sentido escolher uma métrica que resolva esse problema.

Modelando os Dados da aula_01_exemplo_01

- Agora vamos usar o Grid Search nos nossos dados de treino usando o comando `.fit()`.


Código

```
[12]: # Treinando com Grid Search
      grid_search.fit(x_treino_transformado, y_treino) # usando as covariáveis já preprocessadas e nossa variável target

      melhor_modelo_arvore = grid_search.best_estimator_ # Melhor modelo encontrado
```

- Agora temos o melhor modelo para o caso da árvore de decisão.

O código acima não retorna nada pra gente!

Modelando os Dados da aula_01_exemplo_01

- Por fim, vamos analisar o desempenho do modelo nos dados de treinamento com as novas métricas.

Código

```
[17]: # Predição no treino
y_pred_treino_arvore = melhor_modelo_arvore.predict(x_treino_transformado) # covariáveis de treino

# Avaliação
print("=== Métricas de Treino ===")
print("Acurácia:", accuracy_score(y_treino, y_pred_treino_arvore))
print("MCC:", matthews_corrcoef(y_treino, y_pred_treino_arvore))
print("\nRelatório de Classificação:\n", classification_report(y_treino, y_pred_treino_arvore))
```

Saída

```
=== Métricas de Treino ===
Acurácia: 0.9299065420560748
MCC: 0.8662301137279158
```

Relatório de Classificação:

	precision	recall	f1-score	support
0	0.88	0.99	0.93	535
1	0.99	0.87	0.93	535
accuracy			0.93	1070
macro avg	0.94	0.93	0.93	1070
weighted avg	0.94	0.93	0.93	1070

- **Acurácia:** temos uma taxa de acerto de 93%, o que aparentemente indica que o modelo consegue prever bem.
- **Lembre-se:** A acurácia não é uma boa métrica para se analisar unicamente!
- **MCC:** temos uma correlação positiva forte de 86% entre as previsões do modelo e os valores reais, o que mostra que o modelo consegue prever bem.

Modelando os Dados da aula_01_exemplo_01

- Por fim, vamos analisar o desempenho do modelo nos dados de treinamento com as novas métricas.



Código

```
[17]: # Predição no treino
y_pred_treino_arvore = melhor_modelo_arvore.predict(x_treino_transformado) # covariáveis de treino

# Avaliação
print("=== Métricas de Treino ===")
print("Acurácia:", accuracy_score(y_treino, y_pred_treino_arvore))
print("MCC:", matthews_corrcoef(y_treino, y_pred_treino_arvore))
print("\nRelatório de Classificação:\n", classification_report(y_treino, y_pred_treino_arvore))
```

```
=== Métricas de Treino ===
Acurácia: 0.9299065420560748
MCC: 0.8662301137279158
```

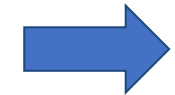
Relatório de Classificação:

	precision	recall	f1-score	support
0	0.88	0.99	0.93	535
1	0.99	0.87	0.93	535
accuracy			0.93	1070
macro avg	0.94	0.93	0.93	1070
weighted avg	0.94	0.93	0.93	1070

- a função **classification_report()** vai retornar um relatório completo do seu modelo.
- Assim ele mostra os dois possíveis cenários de predição.

Modelando os Dados da aula_01_exemplo_01

- Por fim, vamos analisar o desempenho do modelo nos dados de treinamento com as novas métricas.



Código

```
[17]: # Predição no treino
y_pred_treino_arvore = melhor_modelo_arvore.predict(x_treino_transformado) # covariáveis de treino

# Avaliação
print("=== Métricas de Treino ===")
print("Acurácia:", accuracy_score(y_treino, y_pred_treino_arvore))
print("MCC:", matthews_corrcoef(y_treino, y_pred_treino_arvore))
print("\nRelatório de Classificação:\n", classification_report(y_treino, y_pred_treino_arvore))
```

```
=== Métricas de Treino ===
Acurácia: 0.9299065420560748
MCC: 0.8662301137279158
```

Relatório de Classificação:

	precision	recall	f1-score	support
0	0.88	0.99	0.93	535
1	0.99	0.87	0.93	535
accuracy			0.93	1070
macro avg	0.94	0.93	0.93	1070
weighted avg	0.94	0.93	0.93	1070

- O primeiro é Quando consideramos a classe 0 (custo baixo) como a classe positiva.
- O segundo é quando consideramos (e é de fato) que a classe 1 (custo alto) é a classe positiva.

Modelando os Dados da aula_01_exemplo_01

- Por fim, vamos analisar o desempenho do modelo nos dados de treinamento com as novas métricas.



Código

```
[17]: # Predição no treino
y_pred_treino_arvore = melhor_modelo_arvore.predict(x_treino_transformado) # covariáveis de treino

# Avaliação
print("=== Métricas de Treino ===")
print("Acurácia:", accuracy_score(y_treino, y_pred_treino_arvore))
print("MCC:", matthews_corrcoef(y_treino, y_pred_treino_arvore))
print("\nRelatório de Classificação:\n", classification_report(y_treino, y_pred_treino_arvore))
```

```
=== Métricas de Treino ===
Acurácia: 0.9299065420560748
MCC: 0.8662301137279158
```

Relatório de Classificação:

	precision	recall	f1-score	support
0	0.88	0.99	0.93	535
1	0.99	0.87	0.93	535
accuracy			0.93	1070
macro avg	0.94	0.93	0.93	1070
weighted avg	0.94	0.93	0.93	1070

- Considerando a classe 0 (custo baixo) como a classe positiva, temos altos valores de precision, recall e f1-score.

Modelando os Dados da aula_01_exemplo_01

- Por fim, vamos analisar o desempenho do modelo nos dados de treinamento com as novas métricas.

Código

```
[17]: # Predição no treino
y_pred_treino_arvore = melhor_modelo_arvore.predict(x_treino_transformado) # covariáveis de treino

# Avaliação
print("=== Métricas de Treino ===")
print("Acurácia:", accuracy_score(y_treino, y_pred_treino_arvore))
print("MCC:", matthews_corrcoef(y_treino, y_pred_treino_arvore))
print("\nRelatório de Classificação:\n", classification_report(y_treino, y_pred_treino_arvore))
```

Saída

```
=== Métricas de Treino ===
Acurácia: 0.9299065420560748
MCC: 0.8662301137279158
```

Relatório de Classificação:

	precision	recall	f1-score	support
0	0.88	0.99	0.93	535
1	0.99	0.87	0.93	535
accuracy			0.93	1070
macro avg	0.94	0.93	0.93	1070
weighted avg	0.94	0.93	0.93	1070

- O **support** é uma coluna que diz a quantidade de observações (indivíduos) em cada categoria. Nesse caso temos 535 indivíduos da classe 0 (custo baixo) e 535 indivíduos da classe 1 (custo alto).
- Para o cálculo global das medidas que são a **macro e weighted (ponderada)**, utilizamos todos os indivíduos, por isso 1070 que é a soma dos indivíduos da classe 0 e 1

Modelando os Dados da aula_01_exemplo_01

- Por fim, vamos analisar o desempenho do modelo nos dados de treinamento com as novas métricas.



Código

```
[17]: # Predição no treino
y_pred_treino_arvore = melhor_modelo_arvore.predict(x_treino_transformado) # covariáveis de treino

# Avaliação
print("=== Métricas de Treino ===")
print("Acurácia:", accuracy_score(y_treino, y_pred_treino_arvore))
print("MCC:", matthews_corrcoef(y_treino, y_pred_treino_arvore))
print("\nRelatório de Classificação:\n", classification_report(y_treino, y_pred_treino_arvore))
```

```
=== Métricas de Treino ===
Acurácia: 0.9299065420560748
MCC: 0.8662301137279158
```

Relatório de Classificação:

	precision	recall	f1-score	support
0	0.88	0.99	0.93	535
1	0.99	0.87	0.93	535
accuracy			0.93	1070
macro avg	0.94	0.93	0.93	1070
weighted avg	0.94	0.93	0.93	1070

- Precision:** Temos que nosso modelo acerta 88% das predições positivas (classe 0 ou custo baixo nessa linha), ou seja temos poucos falsos positivos.
- Nesse caso **o modelo produziu uma quantidade pequena de falsos positivos** (indivíduos que foram erroneamente classificados como 0), portanto ainda segue como um bom modelo.

Modelando os Dados da aula_01_exemplo_01

- Por fim, vamos analisar o desempenho do modelo nos dados de treinamento com as novas métricas.

Código

```
[17]: # Predição no treino
y_pred_treino_arvore = melhor_modelo_arvore.predict(x_treino_transformado) # covariáveis de treino

# Avaliação
print("=== Métricas de Treino ===")
print("Acurácia:", accuracy_score(y_treino, y_pred_treino_arvore))
print("MCC:", matthews_corrcoef(y_treino, y_pred_treino_arvore))
print("\nRelatório de Classificação:\n", classification_report(y_treino, y_pred_treino_arvore))
```

Saída

```
=== Métricas de Treino ===
Acurácia: 0.9299065420560748
MCC: 0.8662301137279158

Relatório de Classificação:
              precision    recall  f1-score   support

     0       0.88         0.99         0.93         535
     1       0.99         0.87         0.93         535

 accuracy                   0.93         1070
 macro avg                  0.94         0.93         0.93         1070
 weighted avg               0.94         0.93         0.93         1070
```

- Recall:** Nosso modelo alcançou 99% dos indivíduos que são da classe 0 (custo baixo), ou seja poucos falsos negativos.
- Nesse caso o modelo produziu uma quantidade pequena de falsos negativos** (indivíduos que foram erroneamente classificados como 1 ou custo alto).

Modelando os Dados da aula_01_exemplo_01

- Por fim, vamos analisar o desempenho do modelo nos dados de treinamento com as novas métricas.

Código

```
[17]: # Predição no treino
y_pred_treino_arvore = melhor_modelo_arvore.predict(x_treino_transformado) # covariáveis de treino

# Avaliação
print("=== Métricas de Treino ===")
print("Acurácia:", accuracy_score(y_treino, y_pred_treino_arvore))
print("MCC:", matthews_corrcoef(y_treino, y_pred_treino_arvore))
print("\nRelatório de Classificação:\n", classification_report(y_treino, y_pred_treino_arvore))
```

Saída

```
=== Métricas de Treino ===
Acurácia: 0.9299065420560748
MCC: 0.8662301137279158

Relatório de Classificação:
              precision    recall  f1-score   support

     0       0.88         0.99         0.93         535
     1       0.99         0.87         0.93         535

 accuracy                   0.93         1070
 macro avg              0.94         0.93         0.93         1070
 weighted avg           0.94         0.93         0.93         1070
```

- F1-Score (0.93):** O F1-score indica um ótimo equilíbrio geral de 93% entre o precision e o recall do modelo.
- O support** é o número de observações usadas para a avaliação, nesse caso 535.

Modelando os Dados da aula_01_exemplo_01

- Por fim, vamos analisar o desempenho do modelo nos dados de treinamento com as novas métricas.

Código

```
[17]: # Predição no treino
y_pred_treino_arvore = melhor_modelo_arvore.predict(x_treino_transformado) # covariáveis de treino

# Avaliação
print("=== Métricas de Treino ===")
print("Acurácia:", accuracy_score(y_treino, y_pred_treino_arvore))
print("MCC:", matthews_corrcoef(y_treino, y_pred_treino_arvore))
print("\nRelatório de Classificação:\n", classification_report(y_treino, y_pred_treino_arvore))
```

Saída

```
=== Métricas de Treino ===
Acurácia: 0.9299065420560748
MCC: 0.8662301137279158

Relatório de Classificação:
              precision    recall  f1-score   support

     0       0.88        0.99        0.93         535
     1       0.99        0.87        0.93         535

 accuracy          0.93         1070
 macro avg         0.94         0.93         0.93         1070
 weighted avg      0.94         0.93         0.93         1070
```

- Considerando a **classe 1 (custo alto)** como a classe positiva.
- Precision:** Nosso modelo conseguiu acertar 99% das predições positivas, ou seja temos poucos falsos positivos.

Modelando os Dados da aula_01_exemplo_01

- Por fim, vamos analisar o desempenho do modelo nos dados de treinamento com as novas métricas.



Código

```
[17]: # Predição no treino
y_pred_treino_arvore = melhor_modelo_arvore.predict(x_treino_transformado) # covariáveis de treino

# Avaliação
print("=== Métricas de Treino ===")
print("Acurácia:", accuracy_score(y_treino, y_pred_treino_arvore))
print("MCC:", matthews_corrcoef(y_treino, y_pred_treino_arvore))
print("\nRelatório de Classificação:\n", classification_report(y_treino, y_pred_treino_arvore))
```

```
=== Métricas de Treino ===
Acurácia: 0.9299065420560748
MCC: 0.8662301137279158
```

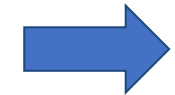
Relatório de Classificação:

	precision	recall	f1-score	support
0	0.88	0.99	0.93	535
1	0.99	0.87	0.93	535
accuracy			0.93	1070
macro avg	0.94	0.93	0.93	1070
weighted avg	0.94	0.93	0.93	1070

- Nesse caso o **modelo produziu uma quantidade pequena de falsos positivos** (indivíduos que foram erroneamente classificados como 1 ou custo alto), portanto ainda segue como um bom modelo.

Modelando os Dados da aula_01_exemplo_01

- Por fim, vamos analisar o desempenho do modelo nos dados de treinamento com as novas métricas.



Código

```
[17]: # Predição no treino
y_pred_treino_arvore = melhor_modelo_arvore.predict(x_treino_transformado) # covariáveis de treino

# Avaliação
print("=== Métricas de Treino ===")
print("Acurácia:", accuracy_score(y_treino, y_pred_treino_arvore))
print("MCC:", matthews_corrcoef(y_treino, y_pred_treino_arvore))
print("\nRelatório de Classificação:\n", classification_report(y_treino, y_pred_treino_arvore))
```



Saída

```
=== Métricas de Treino ===
Acurácia: 0.9299065420560748
MCC: 0.8662301137279158

Relatório de Classificação:
              precision    recall  f1-score   support

     0       0.88        0.99        0.93         535
     1       0.99        0.87        0.93         535

 accuracy          0.93         1070
 macro avg         0.94         1070
 weighted avg      0.94         1070
```

- Recall:** No Recall, nosso modelo conseguiu alcançar 87% dos indivíduos da classe positiva (custo alto), ou seja há poucos falsos negativos.
- Aqui o modelo produziu uma quantidade pequena de falsos negativos** (indivíduos que foram erroneamente classificados como 0 ou custo baixo).

Modelando os Dados da aula_01_exemplo_01

- Por fim, vamos analisar o desempenho do modelo nos dados de treinamento com as novas métricas.



Código

```
[17]: # Predição no treino
y_pred_treino_arvore = melhor_modelo_arvore.predict(x_treino_transformado) # covariáveis de treino

# Avaliação
print("=== Métricas de Treino ===")
print("Acurácia:", accuracy_score(y_treino, y_pred_treino_arvore))
print("MCC:", matthews_corrcoef(y_treino, y_pred_treino_arvore))
print("\nRelatório de Classificação:\n", classification_report(y_treino, y_pred_treino_arvore))
```

```
=== Métricas de Treino ===
Acurácia: 0.9299065420560748
MCC: 0.8662301137279158
```

```
Relatório de Classificação:
```

	precision	recall	f1-score	support
0	0.88	0.99	0.93	535
1	0.99	0.87	0.93	535
accuracy			0.93	1070
macro avg	0.94	0.93	0.93	1070
weighted avg	0.94	0.93	0.93	1070

- F1-Score (0.93):** O F1-score indica um ótimo equilíbrio geral de 93% entre o precision e o recall do modelo.
- Aqui o modelo produziu uma quantidade pequena de falsos negativos (indivíduos que foram erroneamente classificados como 0 ou custo baixo).

Modelando os Dados da aula_01_exemplo_01

- Por fim, vamos analisar o desempenho do modelo nos dados de treinamento com as novas métricas.



Código

```
[17]: # Predição no treino
y_pred_treino_arvore = melhor_modelo_arvore.predict(x_treino_transformado) # covariáveis de treino

# Avaliação
print("=== Métricas de Treino ===")
print("Acurácia:", accuracy_score(y_treino, y_pred_treino_arvore))
print("MCC:", matthews_corrcoef(y_treino, y_pred_treino_arvore))
print("\nRelatório de Classificação:\n", classification_report(y_treino, y_pred_treino_arvore))
```

```
=== Métricas de Treino ===
Acurácia: 0.9299065420560748
MCC: 0.8662301137279158
```

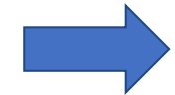
Relatório de Classificação:

	precision	recall	f1-score	support
0	0.88	0.99	0.93	535
1	0.99	0.87	0.93	535
accuracy			0.93	1070
macro avg	0.94	0.93	0.93	1070
weighted avg	0.94	0.93	0.93	1070

- Vemos que o relatório nos mostra a acurácia, que bate com a que calculamos anteriormente.

Modelando os Dados da aula_01_exemplo_01

- Por fim, vamos analisar o desempenho do modelo nos dados de treinamento com as novas métricas.



Código

```
[17]: # Predição no treino
y_pred_treino_arvore = melhor_modelo_arvore.predict(x_treino_transformado) # covariáveis de treino

# Avaliação
print("=== Métricas de Treino ===")
print("Acurácia:", accuracy_score(y_treino, y_pred_treino_arvore))
print("MCC:", matthews_corrcoef(y_treino, y_pred_treino_arvore))
print("\nRelatório de Classificação:\n", classification_report(y_treino, y_pred_treino_arvore))
```

```
=== Métricas de Treino ===
Acurácia: 0.9299065420560748
MCC: 0.8662301137279158
```

Relatório de Classificação:

	precision	recall	f1-score	support
0	0.88	0.99	0.93	535
1	0.99	0.87	0.93	535
accuracy			0.93	1070
macro avg	0.94	0.93	0.93	1070
weighted avg	0.94	0.93	0.93	1070

- A **média macro** é a **média aritmética** do precision, recall e F1-score das duas classes.

- Basta somar cada métrica em cada cenário de classe (quando 0 é positiva e quando 1 é positiva) e dividir por 2.

Modelando os Dados da aula_01_exemplo_01

- Por fim, vamos analisar o desempenho do modelo nos dados de treinamento com as novas métricas.



Código

```
[17]: # Predição no treino
y_pred_treino_arvore = melhor_modelo_arvore.predict(x_treino_transformado) # covariáveis de treino

# Avaliação
print("=== Métricas de Treino ===")
print("Acurácia:", accuracy_score(y_treino, y_pred_treino_arvore))
print("MCC:", matthews_corrcoef(y_treino, y_pred_treino_arvore))
print("\nRelatório de Classificação:\n", classification_report(y_treino, y_pred_treino_arvore))
```

```
=== Métricas de Treino ===
Acurácia: 0.9299065420560748
MCC: 0.8662301137279158
```

Relatório de Classificação:

	precision	recall	f1-score	support
0	0.88	0.99	0.93	535
1	0.99	0.87	0.93	535
accuracy			0.93	1070
macro avg	0.94	0.93	0.93	1070
weighted avg	0.94	0.93	0.93	1070

- Para o precision, somamos o 0,88 e o 0,99 e dividimos por 2.
- Note que é **um bom resumo geral para avaliar o modelo quando as classes estão equilibradas.**

Modelando os Dados da aula_01_exemplo_01

- Por fim, vamos analisar o desempenho do modelo nos dados de treinamento com as novas métricas.



Código

```
[17]: # Predição no treino
y_pred_treino_arvore = melhor_modelo_arvore.predict(x_treino_transformado) # covariáveis de treino

# Avaliação
print("=== Métricas de Treino ===")
print("Acurácia:", accuracy_score(y_treino, y_pred_treino_arvore))
print("MCC:", matthews_corrcoef(y_treino, y_pred_treino_arvore))
print("\nRelatório de Classificação:\n", classification_report(y_treino, y_pred_treino_arvore))
```

```
=== Métricas de Treino ===
Acurácia: 0.9299065420560748
MCC: 0.8662301137279158
```


Relatório de Classificação:

	precision	recall	f1-score	support
0	0.88	0.99	0.93	535
1	0.99	0.87	0.93	535
accuracy			0.93	1070
macro avg	0.94	0.93	0.93	1070
weighted avg	0.94	0.93	0.93	1070

- Precision (0.94):** Em média, quando o modelo previu uma classe (seja 0 ou 1), ele acertou em 94% das vezes.
- Recall (0.93):** Em média, o modelo conseguiu alcançar 93% de todas os indivíduos reais existentes de cada classe.


Modelando os Dados da aula_01_exemplo_01

- Por fim, vamos analisar o desempenho do modelo nos dados de treinamento com as novas métricas.

 Código

```
[17]: # Predição no treino
y_pred_treino_arvore = melhor_modelo_arvore.predict(x_treino_transformado) # covariáveis de treino

# Avaliação
print("=== Métricas de Treino ===")
print("Acurácia:", accuracy_score(y_treino, y_pred_treino_arvore))
print("MCC:", matthews_corrcoef(y_treino, y_pred_treino_arvore))
print("\nRelatório de Classificação:\n", classification_report(y_treino, y_pred_treino_arvore))
```

 Saída

```
=== Métricas de Treino ===
Acurácia: 0.9299065420560748
MCC: 0.8662301137279158

Relatório de Classificação:
              precision    recall  f1-score   support

     0       0.88         0.99         0.93         535
     1       0.99         0.87         0.93         535

 accuracy                   0.93         1070
 macro avg       0.94         0.93         0.93         1070
weighted avg       0.94         0.93         0.93         1070
```

- F1-Score (0.93):** A média ponderada do F1-score indica um ótimo equilíbrio geral de 93% entre o precision e o recall do modelo.

Modelando os Dados da aula_01_exemplo_01

- Por fim, vamos analisar o desempenho do modelo nos dados de treinamento com as novas métricas.

→ Código

```
[17]: # Predição no treino
y_pred_treino_arvore = melhor_modelo_arvore.predict(x_treino_transformado) # covariáveis de treino

# Avaliação
print("=== Métricas de Treino ===")
print("Acurácia:", accuracy_score(y_treino, y_pred_treino_arvore))
print("MCC:", matthews_corrcoef(y_treino, y_pred_treino_arvore))
print("\nRelatório de Classificação:\n", classification_report(y_treino, y_pred_treino_arvore))
```

→ Saída

```
=== Métricas de Treino ===
Acurácia: 0.9299065420560748
MCC: 0.8662301137279158

Relatório de Classificação:
              precision    recall  f1-score   support

     0       0.88        0.99        0.93        535
     1       0.99        0.87        0.93        535

 accuracy                   0.93        1070
 macro avg                  0.94        1070
 weighted avg               0.94        1070
```

- A média ponderada é mais utilizada quando temos **um desequilíbrio de classes**.
- Como sabemos que nesse cenário dividimos pela mediana e consequentemente temos categorias balanceadas, **o resultado da média ponderada converge para a média macro**.

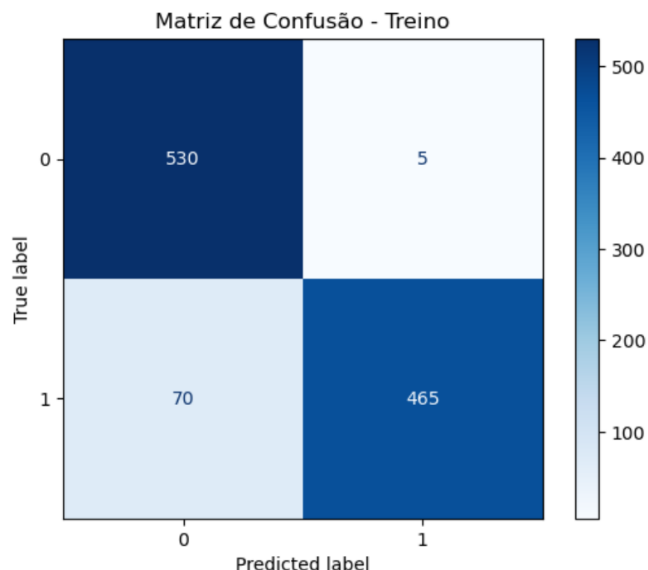
Modelando os Dados da aula_01_exemplo_01

- Por fim, vamos analisar o desempenho do modelo nos dados de **treinamento** com as novas métricas.

→
Código

```
[18]: ConfusionMatrixDisplay.from_estimator(
      melhor_modelo_arvore,      # modelo treinado
      x_treino_transformado,     # dados de treino
      y_treino,                  # target
      display_labels=[0, 1],    # rótulos das classes
      cmap=plt.cm.Blues,        # ou 'true' para normalizar
      normalize=None             # ou 'true' para normalizar
    )
plt.title("Matriz de Confusão - Treino")
plt.show()
```

→
Saída



- **Verdadeiros Negativos (530):** O modelo acertou 530 vezes ao prever a classe 0 quando a classe real era de fato 0 (custo baixo).
- **Verdadeiros Positivos (465):** O modelo acertou 465 vezes ao prever a classe 1 (custo alto) quando a classe real era de fato 1 (custo alto).

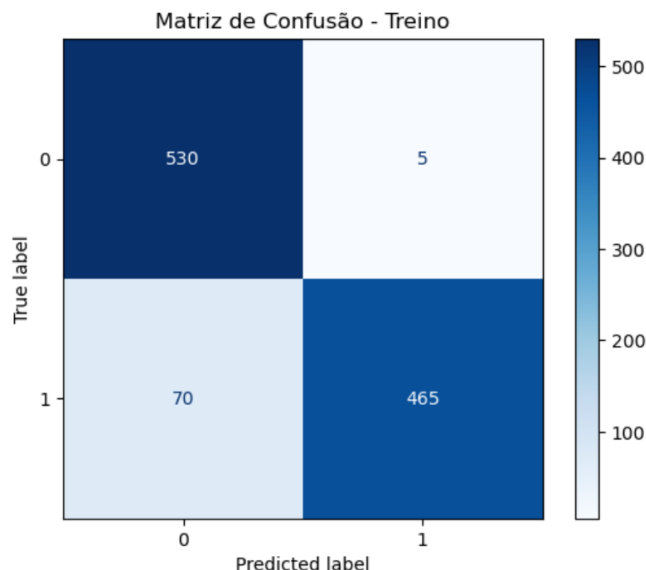
Modelando os Dados da aula_01_exemplo_01

- Por fim, vamos analisar o desempenho do modelo nos dados de **treinamento** com as novas métricas.

→
Código

```
[18]: ConfusionMatrixDisplay.from_estimator(
      melhor_modelo_arvore,      # modelo treinado
      x_treino_transformado,     # dados de treino
      y_treino,                  # target
      display_labels=[0, 1],    # rótulos das classes
      cmap=plt.cm.Blues,        # ou 'true' para normalizar
      normalize=None             # ou 'true' para normalizar
    )
plt.title("Matriz de Confusão - Treino")
plt.show()
```

→
Saída



- **Falsos Positivos (5):** O modelo errou 5 vezes, classificando algo como da classe 1 (custo alto) quando na verdade era da classe 0 (custo baixo).
- **Falsos Negativos (70):** O modelo errou 70 vezes, classificando algo como da classe 0 (custo baixo) quando na verdade era da classe 1 (custo alto).

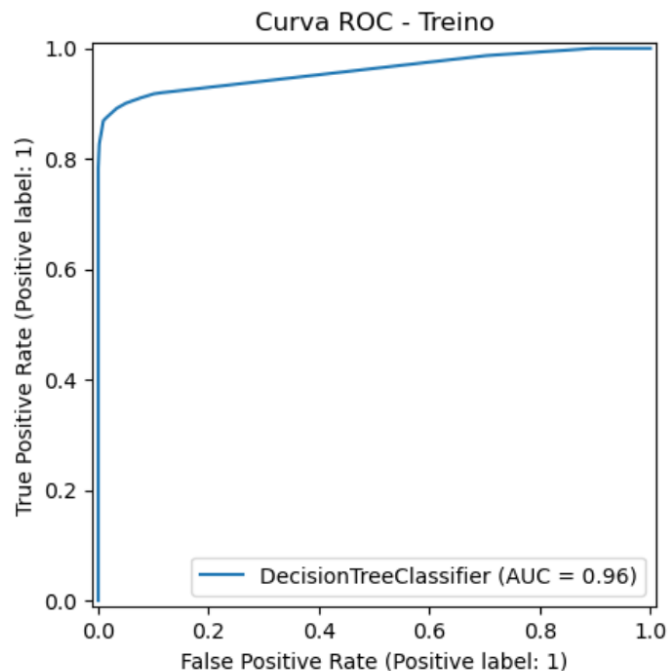
Modelando os Dados da aula_01_exemplo_01

- Por fim, vamos analisar o desempenho do modelo nos dados de treinamento com as novas métricas.

→
Código

```
[19]: # --- Curva ROC ---  
RocCurveDisplay.from_estimator(  
    melhor_modelo_arvore,      # modelo treinado  
    x_treino_transformado,     # dados de treino  
    y_treino                   # target  
)  
plt.title("Curva ROC - Treino")  
plt.show()
```

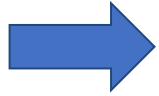
→
Saída



- **Curva ROC:** Note que a curva da linha azul está bem acentuada o que pode indicar que o modelo se ajustou bem aos dados.
- **AUC:** A área sob a curva de 0.96 demonstra que o modelo de árvore possui um alto desempenho em separar os dados entre as duas categorias (custo baixo e custo alto).

Modelando os Dados da aula_01_exemplo_01

- Vamos criar o modelo do KNN.



Código

```
[20]: # criando o modelo  
modelo_knn = KNeighborsClassifier() # criando o modelo KNN; aqui não há aleatoriedade fixa como na árvore
```




- Da mesma forma que criamos a árvore de decisão, conseguimos criar o KNN.

O código acima não retorna nada pra gente!

Modelando os Dados da aula_01_exemplo_01

- Agora vamos criar nossa grade de hiperparâmetros.

 Código

```
[21]: # Grade de hiperparâmetros
      param_grid_knn = {
          "n_neighbors": [3, 5, 7, 9, 11], # Número de vizinhos a considerar na média para prever
          "p": [1, 2] # Tipo de distância: 1 = Manhattan, 2 = Euclidiana
      }
```

- Perceba que são os mesmos hiperparâmetros que discutimos.

O código acima não retorna nada pra gente!

Modelando os Dados da aula_01_exemplo_01

- Agora vamos usar o Grid Search para encontrar os melhores hiperparâmetros.


Código

[22]:

```
# Configurando Grid Search
grid_search_knn = GridSearchCV(
    estimator=modelo_knn,          # modelo KNN a ser otimizado
    param_grid=param_grid_knn,    # Grade de hiperparâmetros
    cv=5,                          # 5-fold cross-validation: o dataset será dividido em 5 partes
    scoring="f1"                  # Métrica de comparação: RMSE (quanto menor, melhor)
)

# Treinando com Grid Search
grid_search_knn.fit(x_treino_transformado, y_treino) # usando as covariáveis já preprocessadas e nossa variável target

melhor_modelo_knn = grid_search_knn.best_estimator_ # Melhor modelo KNN encontrado
```

- Novamente, vamos utilizar a métrica F1-Score como métrica principal para escolher os melhores hiperparâmetros do modelo.

O código acima não retorna nada pra gente!

Modelando os Dados da aula_01_exemplo_01

- Por fim, vamos analisar o desempenho do modelo nos dados de treinamento com as novas métricas.

Código

```
[23]: # Predição no treino
y_pred_treino_knn = melhor_modelo_knn.predict(x_treino_transformado) # covariáveis de treino

# Avaliação
print("=== Métricas de Treino ===")
print("Acurácia:", accuracy_score(y_treino, y_pred_treino_knn))
print("MCC:", matthews_corrcoef(y_treino, y_pred_treino_knn))
print("\nRelatório de Classificação:\n", classification_report(y_treino, y_pred_treino_knn))
```

```
=== Métricas de Treino ===
Acurácia: 0.9018691588785047
MCC: 0.8063470194624572
```

```
Relatório de Classificação:
              precision    recall  f1-score   support

     0       0.87         0.94         0.91         535
     1       0.94         0.86         0.90         535

 accuracy                   0.90         1070
  macro avg                 0.90         0.90         0.90         1070
 weighted avg               0.90         0.90         0.90         1070
```

- **Acurácia:** temos uma taxa de acerto de 90%, o que aparentemente indica que o modelo consegue prever bem.
- **MCC:** temos uma correlação positiva forte de 80% entre as previsões do modelo e os valores reais, o que mostra que o modelo consegue prever bem.

Modelando os Dados da aula_01_exemplo_01

- Por fim, vamos analisar o desempenho do modelo nos dados de treinamento com as novas métricas.

→
Código

```
[23]: # Predição no treino
y_pred_treino_knn = melhor_modelo_knn.predict(x_treino_transformado) # covariáveis de treino

# Avaliação
print("=== Métricas de Treino ===")
print("Acurácia:", accuracy_score(y_treino, y_pred_treino_knn))
print("MCC:", matthews_corrcoef(y_treino, y_pred_treino_knn))
print("\nRelatório de Classificação:\n", classification_report(y_treino, y_pred_treino_knn))
```

→
Saída

```
=== Métricas de Treino ===
Acurácia: 0.9018691588785047
MCC: 0.8063470194624572
```

Relatório de Classificação:

	precision	recall	f1-score	support
0	0.87	0.94	0.91	535
1	0.94	0.86	0.90	535
accuracy			0.90	1070
macro avg	0.90	0.90	0.90	1070
weighted avg	0.90	0.90	0.90	1070

- **Precision (0.90):** Na média, considerando o número de amostras em cada classe, 90% das previsões feitas pelo modelo estavam corretas.
- **Recall (0.90):** Em média, considerando o número de amostras em cada classe, o modelo conseguiu alcançar 90% de todas as instâncias reais.

Modelando os Dados da aula_01_exemplo_01

- Por fim, vamos analisar o desempenho do modelo nos dados de treinamento com as novas métricas.

→
Código

```
[23]: # Predição no treino
y_pred_treino_knn = melhor_modelo_knn.predict(x_treino_transformado) # covariáveis de treino

# Avaliação
print("=== Métricas de Treino ===")
print("Acurácia:", accuracy_score(y_treino, y_pred_treino_knn))
print("MCC:", matthews_corrcoef(y_treino, y_pred_treino_knn))
print("\nRelatório de Classificação:\n", classification_report(y_treino, y_pred_treino_knn))
```

=== Métricas de Treino ===
Acurácia: 0.9018691588785047
MCC: 0.8063470194624572

Relatório de Classificação:

	precision	recall	f1-score	support
0	0.87	0.94	0.91	535
1	0.94	0.86	0.90	535
accuracy			0.90	1070
macro avg	0.90	0.90	0.90	1070
weighted avg	0.90	0.90	0.90	1070

- F1-Score (0.90):** A média ponderada do F1-score indica um ótimo equilíbrio geral de 90% entre o precision e o recall do modelo.

Modelando os Dados da aula_01_exemplo_01

- Por fim, vamos analisar o desempenho do modelo nos dados de treinamento com as novas métricas.

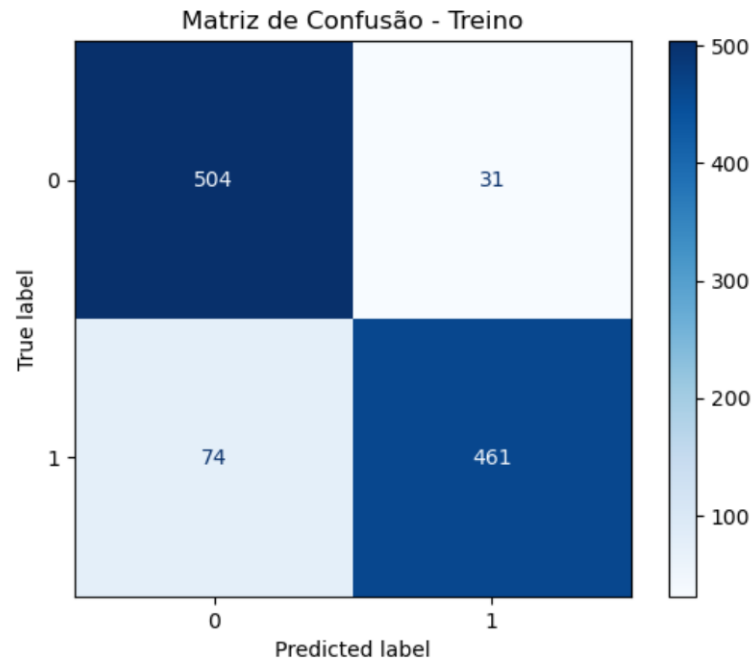


Código

```
[24]: ConfusionMatrixDisplay.from_estimator(  
    melhor_modelo_knn,      # modelo treinado  
    x_treino_transformado,  # dados de treino  
    y_treino,              # target  
    display_labels=[0, 1], # rótulos das classes  
    cmap=plt.cm.Blues,  
    normalize=None         # ou 'true' para normalizar  
)  
plt.title("Matriz de Confusão - Treino")  
plt.show()
```



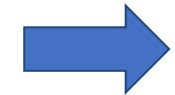
Saída



- **Verdadeiros Negativos (504):** O modelo previu corretamente a classe 0 (custo baixo) um total de 504 vezes.
- **Verdadeiros Positivos (461):** O modelo previu corretamente a classe 1 (custo alto) um total de 461 vezes.

Modelando os Dados da aula_01_exemplo_01

- Por fim, vamos analisar o desempenho do modelo nos dados de treinamento com as novas métricas.

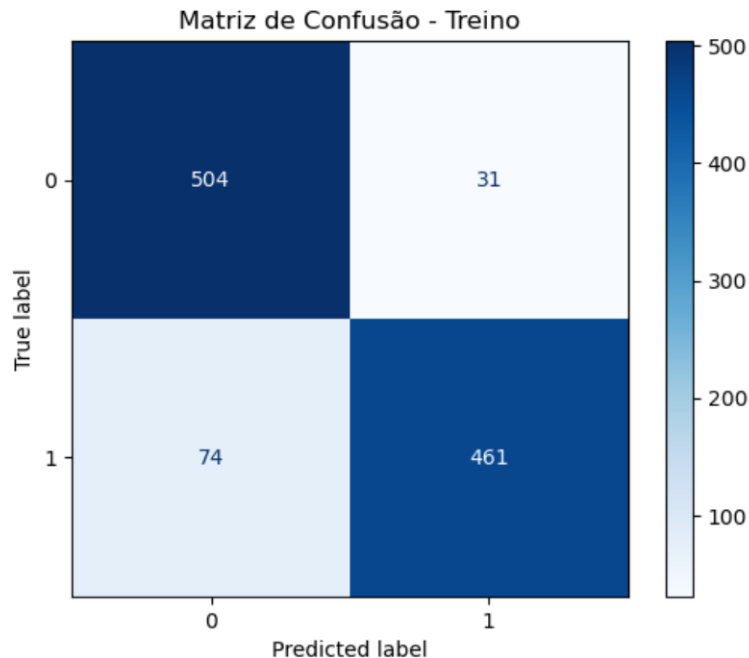


Código

```
[24]: ConfusionMatrixDisplay.from_estimator(  
    melhor_modelo_knn,      # modelo treinado  
    x_treino_transformado,  # dados de treino  
    y_treino,              # target  
    display_labels=[0, 1], # rótulos das classes  
    cmap=plt.cm.Blues,  
    normalize=None         # ou 'true' para normalizar  
)  
plt.title("Matriz de Confusão - Treino")  
plt.show()
```



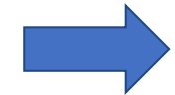
Saída



- **Falsos Positivos (31):** O modelo errou 31 vezes ao prever a classe 1 (custo alto) quando o correto era a classe 0 (custo baixo).
- **Falsos Negativos (74):** O modelo errou 74 vezes ao prever a classe 0 (custo baixo) quando o correto era a classe 1 (custo alto).

Modelando os Dados da aula_01_exemplo_01

- Por fim, vamos analisar o desempenho do modelo nos dados de treinamento com as novas métricas.

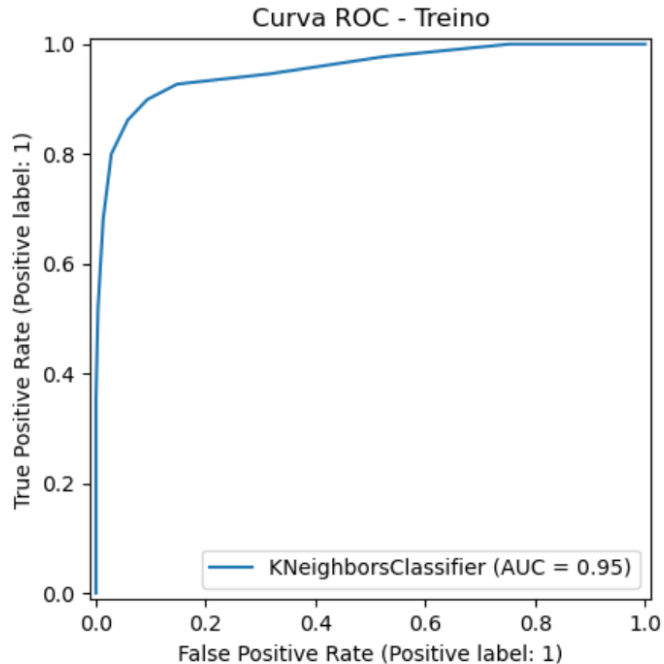


Código

```
[25]: # --- Curva ROC ---  
RocCurveDisplay.from_estimator(  
    melhor_modelo_knn,      # modelo treinado  
    x_treino_transformado,  # dados de treino  
    y_treino                # target  
)  
plt.title("Curva ROC - Treino")  
plt.show()
```



Saída



- **Curva ROC:** A forma da curva, bem próxima ao canto superior esquerdo, mostra que o KNN tem um ótimo desempenho, conseguindo uma alta taxa de verdadeiros positivos (Recall) sem aumentar significativamente a taxa de falsos positivos.

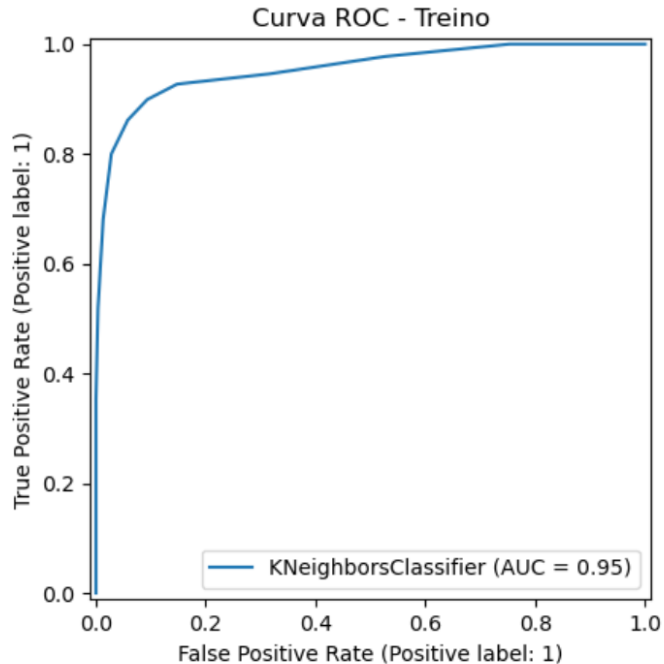
Modelando os Dados da aula_01_exemplo_01

- Por fim, vamos analisar o desempenho do modelo nos dados de treinamento com as novas métricas.

➔
Código

```
[25]: # --- Curva ROC ---  
RocCurveDisplay.from_estimator(  
    melhor_modelo_knn,      # modelo treinado  
    x_treino_transformado,  # dados de treino  
    y_treino                # target  
)  
plt.title("Curva ROC - Treino")  
plt.show()
```

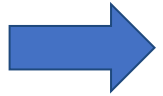
➔
Saída



- A área sob a **curva de 0.95** demonstra que o KNN possui um alto desempenho em separar os dados entre as duas categorias (custo baixo e custo alto).

Modelando os Dados da aula_01_exemplo_01

- Vamos criar o modelo do SVC.



```
[26]: modelo_svc = SVC(probability=True) # probability=True é necessário para curva ROC
```



Código

- Da mesma forma que criamos o KNN, conseguimos criar o SVC.

O código acima não retorna nada pra gente!

Modelando os Dados da aula_01_exemplo_01

- Agora vamos criar nossa grade de hiperparâmetros.


Código

```
[27]: # --- Grade de hiperparâmetros ---  
param_grid_svc = {  
    "C": [0.1, 1, 10],          # regularização  
    "kernel": ["linear", "rbf"], # tipo de kernel  
    "gamma": ["scale", "auto"]  # alcance da influência do ponto (para rbf)  
}
```

- Perceba que são os mesmos hiperparâmetros que discutimos.

O código acima não retorna nada pra gente!

Modelando os Dados da aula_01_exemplo_01

- Agora vamos usar o Grid Search para encontrar os melhores hiperparâmetros.


Código

```
[28]: # --- Configurando Grid Search ---
      grid_search_svc = GridSearchCV(
          estimator=modelo_svc,
          param_grid=param_grid_svc,
          cv=5,
          scoring='f1' # métrica de comparação para classificação binária
      )

      # --- Treinando com Grid Search ---
      grid_search_svc.fit(x_treino_transformado, y_treino)

      # --- Melhor modelo ---
      melhor_modelo_svc = grid_search_svc.best_estimator_
```

- Novamente, vamos utilizar a métrica F1-Score como métrica principal para escolher os melhores hiperparâmetros do modelo.

O código acima não retorna nada pra gente!

Modelando os Dados da aula_01_exemplo_01

- Por fim, vamos analisar o desempenho do modelo nos dados de treinamento com as novas métricas.

→
Código

```
[29]: # --- Predição no treino ---
      y_pred_treino_svc = melhor_modelo_svc.predict(x_treino_transformado)

      # --- Avaliação ---
      print("=== Métricas de Treino SVC ===")
      print("Acurácia:", accuracy_score(y_treino, y_pred_treino_svc))
      print("MCC:", matthews_corrcoef(y_treino, y_pred_treino_svc))
      print("\nRelatório de Classificação:\n", classification_report(y_treino, y_pred_treino_svc))
```

→
Saída

```
=== Métricas de Treino SVC ===
Acurácia: 0.9411214953271028
MCC: 0.885106495782046
```

Relatório de Classificação:

	precision	recall	f1-score	support
0	0.91	0.98	0.94	535
1	0.98	0.90	0.94	535
accuracy			0.94	1070
macro avg	0.94	0.94	0.94	1070
weighted avg	0.94	0.94	0.94	1070

- **Acurácia:** temos uma taxa de acerto de 94%, o que aparentemente indica que o modelo consegue prever bem.
- **MCC:** temos uma correlação positiva forte de 88% entre as previsões do modelo e os valores reais, o que mostra que o modelo consegue prever bem.

Modelando os Dados da aula_01_exemplo_01

- Por fim, vamos analisar o desempenho do modelo nos dados de treinamento com as novas métricas.

→
Código

```
[29]: # --- Predição no treino ---
y_pred_treino_svc = melhor_modelo_svc.predict(x_treino_transformado)

# --- Avaliação ---
print("=== Métricas de Treino SVC ===")
print("Acurácia:", accuracy_score(y_treino, y_pred_treino_svc))
print("MCC:", matthews_corrcoef(y_treino, y_pred_treino_svc))
print("\nRelatório de Classificação:\n", classification_report(y_treino, y_pred_treino_svc))
```

→
Saída

```
=== Métricas de Treino SVC ===
Acurácia: 0.9411214953271028
MCC: 0.885106495782046
```

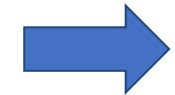
```
Relatório de Classificação:
```

	precision	recall	f1-score	support
0	0.91	0.98	0.94	535
1	0.98	0.90	0.94	535
accuracy			0.94	1070
macro avg	0.94	0.94	0.94	1070
weighted avg	0.94	0.94	0.94	1070

- **Precision (0.94):** Na média, considerando o número de amostras em cada classe, 94% das previsões feitas pelo modelo estavam corretas.
- **Recall (0.94):** Em média, considerando o número de amostras em cada classe, o modelo conseguiu alcançar 94% de todas as instâncias reais.

Modelando os Dados da aula_01_exemplo_01

- Por fim, vamos analisar o desempenho do modelo nos dados de treinamento com as novas métricas.



Código

```
[29]: # --- Predição no treino ---
      y_pred_treino_svc = melhor_modelo_svc.predict(x_treino_transformado)

      # --- Avaliação ---
      print("=== Métricas de Treino SVC ===")
      print("Acurácia:", accuracy_score(y_treino, y_pred_treino_svc))
      print("MCC:", matthews_corrcoef(y_treino, y_pred_treino_svc))
      print("\nRelatório de Classificação:\n", classification_report(y_treino, y_pred_treino_svc))
```



Saída

```
=== Métricas de Treino SVC ===
Acurácia: 0.9411214953271028
MCC: 0.885106495782046
```

```
Relatório de Classificação:
```

	precision	recall	f1-score	support
0	0.91	0.98	0.94	535
1	0.98	0.90	0.94	535
accuracy			0.94	1070
macro avg	0.94	0.94	0.94	1070
weighted avg	0.94	0.94	0.94	1070

- **F1-score (0.94):** A média ponderada do F1-score indica um ótimo equilíbrio geral de 94% entre o precision e o recall do modelo.

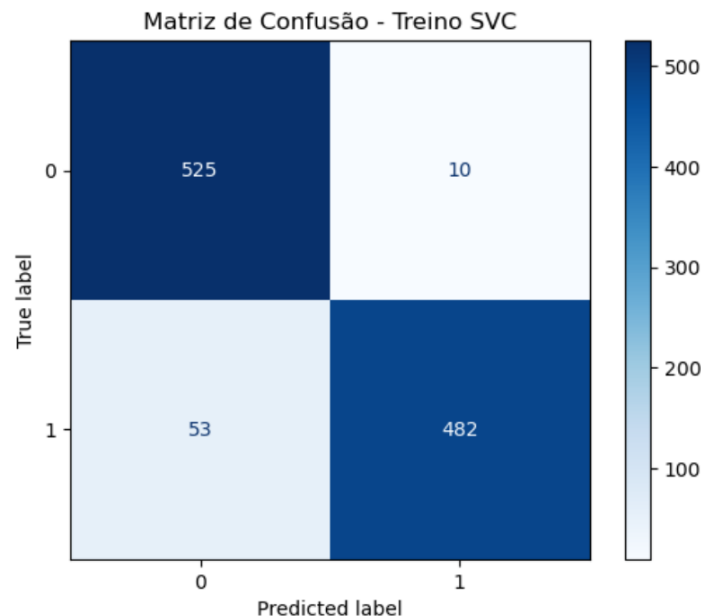
Modelando os Dados da aula_01_exemplo_01

- Por fim, vamos analisar o desempenho do modelo nos dados de treinamento com as novas métricas.

➡
Código

```
[30]: ConfusionMatrixDisplay.from_estimator(  
    melhor_modelo_svc,  
    x_treino_transformado,  
    y_treino,  
    display_labels=[0, 1],  
    cmap=plt.cm.Blues,  
    normalize=None  
)  
plt.title("Matriz de Confusão - Treino SVC")  
plt.show()
```

➡
Saída



- **Verdadeiros Negativos (525):** O modelo acertou 525 vezes ao classificar corretamente uma instância como pertencente à classe 0 (custo baixo).
- **Verdadeiros Positivos (482):** O modelo acertou 482 vezes ao classificar corretamente uma instância como pertencente à classe 1 (custo alto).

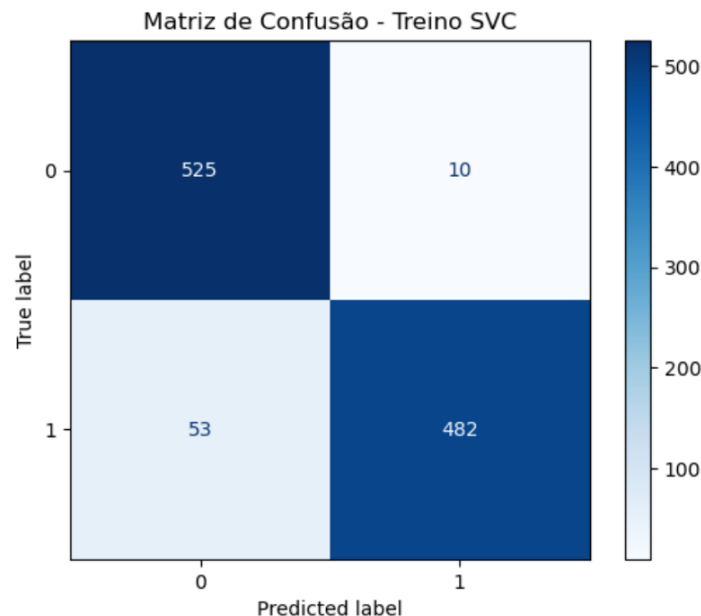
Modelando os Dados da aula_01_exemplo_01

- Por fim, vamos analisar o desempenho do modelo nos dados de treinamento com as novas métricas.

➔
Código

```
[30]: ConfusionMatrixDisplay.from_estimator(  
    melhor_modelo_svc,  
    x_treino_transformado,  
    y_treino,  
    display_labels=[0, 1],  
    cmap=plt.cm.Blues,  
    normalize=None  
)  
plt.title("Matriz de Confusão - Treino SVC")  
plt.show()
```

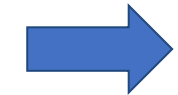
➔
Saída



- **Falsos Positivos (10):** O modelo errou 10 vezes, classificando incorretamente uma instância como 1 (custo alto) quando ela era, na verdade, 0 (custo baixo).
- **Falsos Negativos (53):** O modelo errou 53 vezes, classificando incorretamente uma instância como 0 (custo baixo) quando ela era, na verdade, 1 (custo alto).

Modelando os Dados da aula_01_exemplo_01

- Por fim, vamos analisar o desempenho do modelo nos dados de treinamento com as novas métricas.

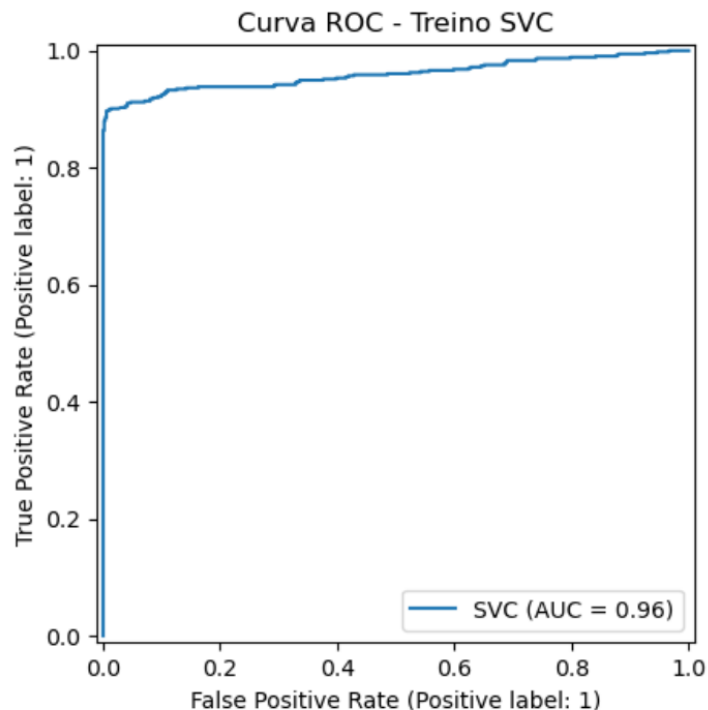


Código

```
[31]: # --- Curva ROC ---  
RocCurveDisplay.from_estimator(  
    melhor_modelo_svc,  
    x_treino_transformado,  
    y_treino  
)  
plt.title("Curva ROC - Treino SVC")  
plt.show()
```



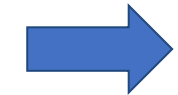
Saída



- **Curva ROC:** A forma da curva, bem próxima ao canto superior esquerdo, mostra que o SVC tem um ótimo desempenho, conseguindo uma alta taxa de verdadeiros positivos (Recall) sem aumentar significativamente a taxa de falsos positivos.

Modelando os Dados da aula_01_exemplo_01

- Por fim, vamos analisar o desempenho do modelo nos dados de treinamento com as novas métricas.

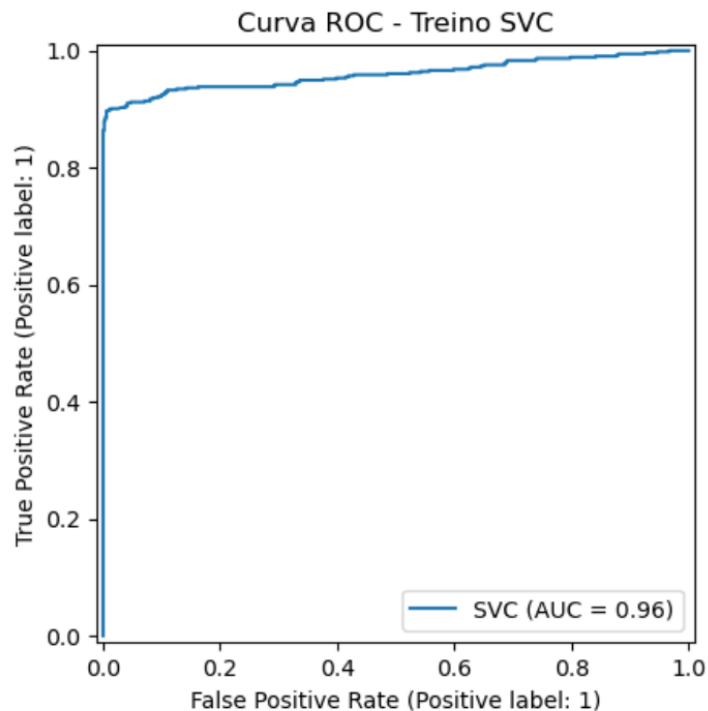


Código

```
[31]: # --- Curva ROC ---  
RocCurveDisplay.from_estimator(  
    melhor_modelo_svc,  
    x_treino_transformado,  
    y_treino  
)  
plt.title("Curva ROC - Treino SVC")  
plt.show()
```



Saída



- A área sob a curva de 0.96 demonstra que o SVC possui um **alto desempenho em separar os dados entre as duas categorias** (custo baixo e custo alto).

Modelando os Dados da aula_01_exemplo_01

- Vamos criar o modelo da Regressão Logística.


Código

```
[32]: # --- Criando o modelo Logistic Regression ---  
modelo_lr = LogisticRegression(max_iter=1000) # aumenta o max_iter para garantir convergência
```

- Da mesma forma que criamos o SVC, conseguimos criar a Regressão Logística. O parâmetro **max_iter = 1000** serve para garantir que a RL consiga fazer sua predição.

O código acima não retorna nada pra gente!

Modelando os Dados da aula_01_exemplo_01

- Agora vamos criar nossa grade de hiperparâmetros.



Código

```
[33]: # --- Grade de hiperparâmetros ---  
param_grid_lr = {  
    "C": [0.01, 0.1, 1, 10, 100], # regularização inversa (quanto maior, menos regularização)  
    "penalty": ["l2"], # tipo de penalização L2 (ridge); L1 exige solver diferente  
    "solver": ["lbfgs"] # solver compatível com L2  
}
```

- Perceba que são os mesmos hiperparâmetros que discutimos.

O código acima não retorna nada pra gente!

Modelando os Dados da aula_01_exemplo_01

- Agora vamos usar o Grid Search para encontrar os melhores hiperparâmetros.


Código

```
[34]: # --- Configurando Grid Search ---
      grid_search_lr = GridSearchCV(
          estimator=modelo_lr,
          param_grid=param_grid_lr,
          cv=5,
          scoring='f1' # métrica para classificação binária
      )
      # --- Treinando com Grid Search ---
      grid_search_lr.fit(x_treino_transformado, y_treino)

      # --- Melhor modelo ---
      melhor_modelo_lr = grid_search_lr.best_estimator_
```

- Novamente, vamos utilizar a métrica F1-Score como métrica principal para escolher os melhores hiperparâmetros do modelo.

O código acima não retorna nada pra gente!

Modelando os Dados da aula_01_exemplo_01

- Por fim, vamos analisar o desempenho do modelo nos dados de treinamento com as novas métricas.

→
Código

```
[35]: # --- Predição no treino ---
y_pred_treino_lr = melhor_modelo_lr.predict(x_treino_transformado)

# --- Avaliação ---
print("=== Métricas de Treino Logistic Regression ===")
print("Acurácia:", accuracy_score(y_treino, y_pred_treino_lr))
print("MCC:", matthews_corrcoef(y_treino, y_pred_treino_lr))
print("\nRelatório de Classificação:\n", classification_report(y_treino, y_pred_treino_lr))
```

=== Métricas de Treino Logistic Regression ===
Acurácia: 0.9074766355140187
MCC: 0.8149660839416788

Relatório de Classificação:

	precision	recall	f1-score	support
0	0.91	0.90	0.91	535
1	0.91	0.91	0.91	535
accuracy			0.91	1070
macro avg	0.91	0.91	0.91	1070
weighted avg	0.91	0.91	0.91	1070

- **Acurácia:** temos uma taxa de acerto de 90%, o que aparentemente indica que o modelo consegue prever bem.
- **MCC:** temos uma correlação positiva forte de 81% entre as previsões do modelo e os valores reais, o que mostra que o modelo consegue prever bem.

Modelando os Dados da aula_01_exemplo_01

- Por fim, vamos analisar o desempenho do modelo nos dados de treinamento com as novas métricas.

→
Código

```
[35]: # --- Predição no treino ---
      y_pred_treino_lr = melhor_modelo_lr.predict(x_treino_transformado)

      # --- Avaliação ---
      print("=== Métricas de Treino Logistic Regression ===")
      print("Acurácia:", accuracy_score(y_treino, y_pred_treino_lr))
      print("MCC:", matthews_corrcoef(y_treino, y_pred_treino_lr))
      print("\nRelatório de Classificação:\n", classification_report(y_treino, y_pred_treino_lr))
```

→
Saída

```
=== Métricas de Treino Logistic Regression ===
Acurácia: 0.9074766355140187
MCC: 0.8149660839416788
```

Relatório de Classificação:

	precision	recall	f1-score	support
0	0.91	0.90	0.91	535
1	0.91	0.91	0.91	535
accuracy			0.91	1070
macro avg	0.91	0.91	0.91	1070
weighted avg	0.91	0.91	0.91	1070

- **Precision (0.91):** Na média, considerando o número de amostras em cada classe, 91% das previsões feitas pelo modelo estavam corretas.
- **Recall (0.91):** Em média, considerando o número de amostras em cada classe, o modelo conseguiu alcançar 91% de todas as instâncias reais.

Modelando os Dados da aula_01_exemplo_01

- Por fim, vamos analisar o desempenho do modelo nos dados de treinamento com as novas métricas.


Código

```
[35]: # --- Predição no treino ---
y_pred_treino_lr = melhor_modelo_lr.predict(x_treino_transformado)

# --- Avaliação ---
print("=== Métricas de Treino Logistic Regression ===")
print("Acurácia:", accuracy_score(y_treino, y_pred_treino_lr))
print("MCC:", matthews_corrcoef(y_treino, y_pred_treino_lr))
print("\nRelatório de Classificação:\n", classification_report(y_treino, y_pred_treino_lr))
```


Saída

```
=== Métricas de Treino Logistic Regression ===
Acurácia: 0.9074766355140187
MCC: 0.8149660839416788
```

Relatório de Classificação:

	precision	recall	f1-score	support
0	0.91	0.90	0.91	535
1	0.91	0.91	0.91	535
accuracy			0.91	1070
macro avg	0.91	0.91	0.91	1070
weighted avg	0.91	0.91	0.91	1070

- **F1-score (0.91):** A média ponderada do F1-score indica um ótimo equilíbrio geral de 91% entre o precision e o recall do modelo.

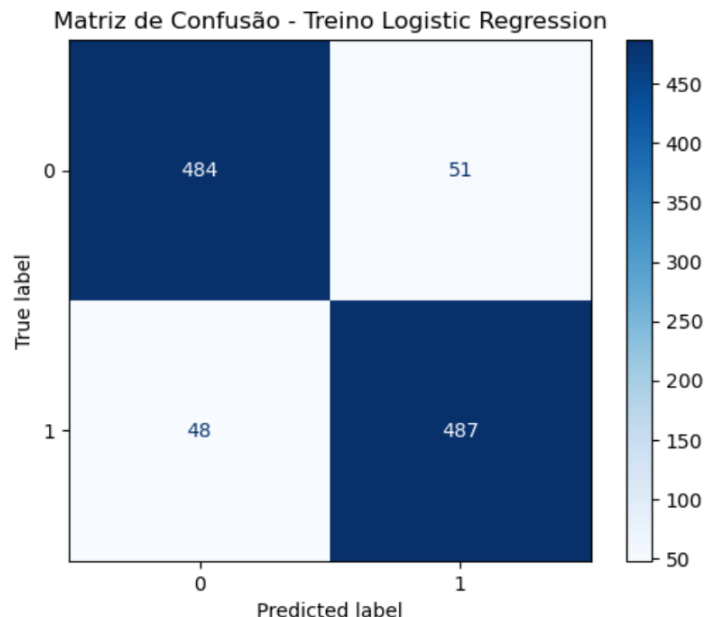
Modelando os Dados da aula_01_exemplo_01

- Por fim, vamos analisar o desempenho do modelo nos dados de treinamento com as novas métricas.

➔
Código

```
[36]: # --- Matriz de Confusão ---
ConfusionMatrixDisplay.from_estimator(
    melhor_modelo_lr,
    x_treino_transformado,
    y_treino,
    display_labels=[0, 1],
    cmap=plt.cm.Blues,
    normalize=None
)
plt.title("Matriz de Confusão - Treino Logistic Regression")
plt.show()
```

➔
Saída



- **Verdadeiros Negativos (484):** O modelo acertou 484 vezes ao prever a classe 0 (custo baixo) quando a classe real era de fato 0 (custo baixo).
- **Verdadeiros Positivos (487):** O modelo acertou 487 vezes ao prever a classe 1 (custo alto) quando a classe real era de fato 1 (custo alto).

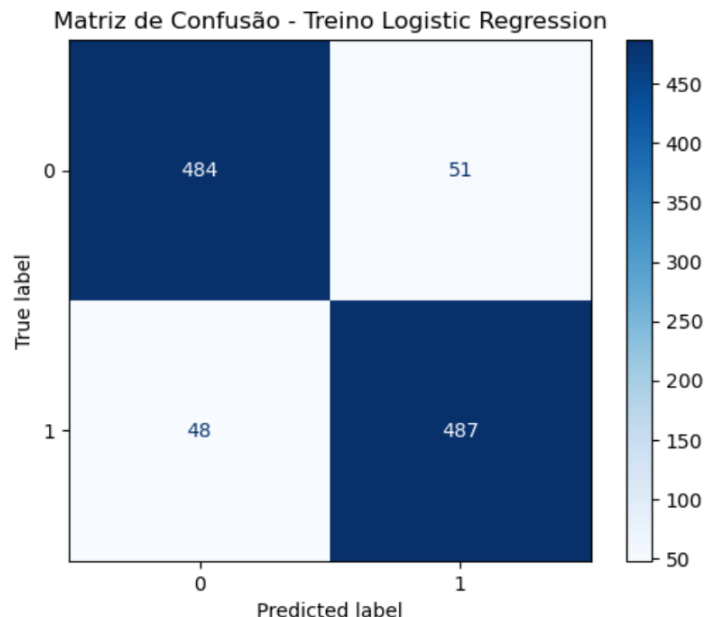
Modelando os Dados da aula_01_exemplo_01

- Por fim, vamos analisar o desempenho do modelo nos dados de treinamento com as novas métricas.

➔
Código

```
[36]: # --- Matriz de Confusão ---  
ConfusionMatrixDisplay.from_estimator(  
    melhor_modelo_lr,  
    x_treino_transformado,  
    y_treino,  
    display_labels=[0, 1],  
    cmap=plt.cm.Blues,  
    normalize=None  
)  
plt.title("Matriz de Confusão - Treino Logistic Regression")  
plt.show()
```

➔
Saída



- **Falsos Positivos (51):** O modelo errou 51 vezes, classificando algo como da classe 1 (custo alto) quando na verdade era da classe 0 (custo baixo).
- **Falsos Negativos (48):** O modelo errou 48 vezes, classificando algo como da classe 0 (custo baixo) quando na verdade era da classe 1 (custo alto).

Modelando os Dados da aula_01_exemplo_01

- Por fim, vamos analisar o desempenho do modelo nos dados de treinamento com as novas métricas.

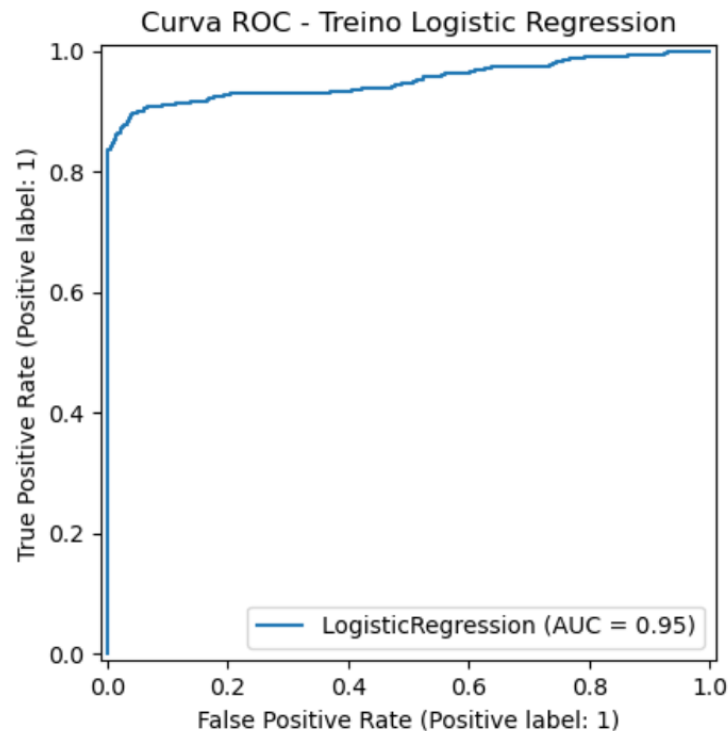


Código

```
[37]: # --- Curva ROC ---  
RocCurveDisplay.from_estimator(  
    melhor_modelo_lr,  
    x_treino_transformado,  
    y_treino  
)  
plt.title("Curva ROC - Treino Logistic Regression")  
plt.show()
```



Saída



- **Curva ROC:** A forma da curva, bem próxima ao canto superior esquerdo, mostra que a regressão logística tem um ótimo desempenho, conseguindo uma alta taxa de verdadeiros positivos (Recall) sem aumentar significativamente a taxa de falsos positivos.

Modelando os Dados da aula_01_exemplo_01

- Por fim, vamos analisar o desempenho do modelo nos dados de treinamento com as novas métricas.

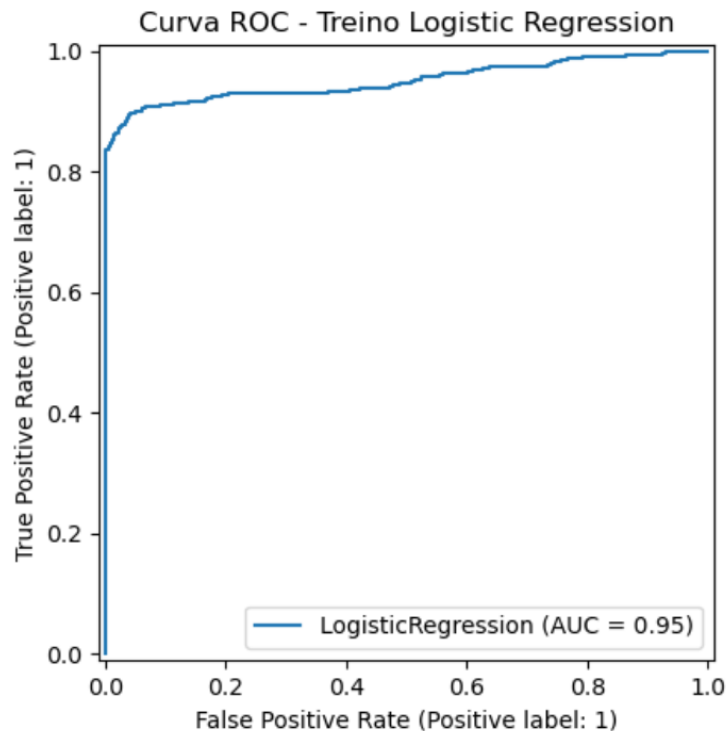


Código

```
[37]: # --- Curva ROC ---  
RocCurveDisplay.from_estimator(  
    melhor_modelo_lr,  
    x_treino_transformado,  
    y_treino  
)  
plt.title("Curva ROC - Treino Logistic Regression")  
plt.show()
```



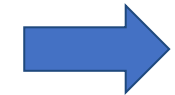
Saída



- A área sob a curva de 0.95 demonstra que a regressão logística **possui um alto desempenho em separar os dados entre as duas categorias** (custo baixo e custo alto).

Modelando os Dados da aula_01_exemplo_01

- Agora vamos testar nossos modelos. Começamos com a árvore de decisão:



Código

```
[38]: # Predição no teste
y_pred_teste_arvore = melhor_modelo_arvore.predict(x_teste_transformado) # covariáveis de teste

# --- Avaliação ---
print("=== Métricas de Teste AD ===")
print("Acurácia:", accuracy_score(y_teste, y_pred_teste_arvore))
print("MCC:", matthews_corrcoef(y_teste, y_pred_teste_arvore))
print("\nRelatório de Classificação:\n", classification_report(y_teste, y_pred_teste_arvore))
```



Saída

=== Métricas de Teste AD ===

Acurácia: 0.9328358208955224

MCC: 0.8691638320240436

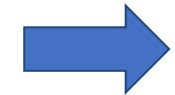
Relatório de Classificação:

	precision	recall	f1-score	support
0	0.90	0.98	0.94	134
1	0.98	0.89	0.93	134
accuracy			0.93	268
macro avg	0.94	0.93	0.93	268
weighted avg	0.94	0.93	0.93	268

- **Acurácia:** temos uma taxa de acerto de 93%, o que aparentemente indica que o modelo consegue prever bem.
- **MCC:** temos uma correlação positiva forte de 86% entre as previsões do modelo e os valores reais, o que mostra que o modelo consegue prever bem.

Modelando os Dados da aula_01_exemplo_01

- Agora vamos testar nossos modelos. Começamos com a árvore de decisão:



Código

```
[38]: # Predição no teste
y_pred_teste_arvore = melhor_modelo_arvore.predict(x_teste_transformado) # covariáveis de teste

# --- Avaliação ---
print("=== Métricas de Teste AD ===")
print("Acurácia:", accuracy_score(y_teste, y_pred_teste_arvore))
print("MCC:", matthews_corrcoef(y_teste, y_pred_teste_arvore))
print("\nRelatório de Classificação:\n", classification_report(y_teste, y_pred_teste_arvore))
```



Saída

```
=== Métricas de Teste AD ===
Acurácia: 0.9328358208955224
MCC: 0.8691638320240436
```

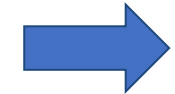
Relatório de Classificação:

	precision	recall	f1-score	support
0	0.90	0.98	0.94	134
1	0.98	0.89	0.93	134
accuracy			0.93	268
macro avg	0.94	0.93	0.93	268
weighted avg	0.94	0.93	0.93	268

- **Precision (0.94):** Na média, considerando o número de amostras em cada classe, 94% das previsões feitas pelo modelo estavam corretas.
- **Recall (0.93):** Em média, considerando o número de amostras em cada classe, o modelo conseguiu alcançar 93% de todas as instâncias reais.

Modelando os Dados da aula_01_exemplo_01

- Agora vamos testar nossos modelos. Começamos com a árvore de decisão:



Código

```
[38]: # Predição no teste
y_pred_teste_arvore = melhor_modelo_arvore.predict(x_teste_transformado) # covariáveis de teste

# --- Avaliação ---
print("=== Métricas de Teste AD ===")
print("Acurácia:", accuracy_score(y_teste, y_pred_teste_arvore))
print("MCC:", matthews_corrcoef(y_teste, y_pred_teste_arvore))
print("\nRelatório de Classificação:\n", classification_report(y_teste, y_pred_teste_arvore))
```



Saída

```
=== Métricas de Teste AD ===
Acurácia: 0.9328358208955224
MCC: 0.8691638320240436
```

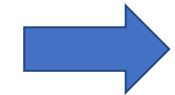
Relatório de Classificação:

	precision	recall	f1-score	support
0	0.90	0.98	0.94	134
1	0.98	0.89	0.93	134
accuracy			0.93	268
macro avg	0.94	0.93	0.93	268
weighted avg	0.94	0.93	0.93	268

- **F1-score (0.93):** A média ponderada do F1-score indica um ótimo equilíbrio geral de 93% entre o precision e o recall do modelo.

Modelando os Dados da aula_01_exemplo_01

- Agora vamos testar o KNN.



Código

```
[39]: # Predição no teste usando o melhor modelo KNN
y_pred_teste_knn = melhor_modelo_knn.predict(x_teste_transformado) # covariáveis de teste

# --- Avaliação ---
print("=== Métricas de Teste KNN ===")
print("Acurácia:", accuracy_score(y_teste, y_pred_teste_knn))
print("MCC:", matthews_corrcoef(y_teste, y_pred_teste_knn))
print("\nRelatório de Classificação:\n", classification_report(y_teste, y_pred_teste_knn))
```



Saída

=== Métricas de Teste KNN ===

Acurácia: 0.8992537313432836

MCC: 0.8022919325102592

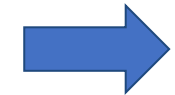
Relatório de Classificação:

	precision	recall	f1-score	support
0	0.86	0.95	0.90	134
1	0.94	0.85	0.89	134
accuracy			0.90	268
macro avg	0.90	0.90	0.90	268
weighted avg	0.90	0.90	0.90	268

- **Acurácia:** temos uma taxa de acerto de 90%, o que aparentemente indica que o modelo consegue prever bem.
- **MCC:** temos uma correlação positiva forte de 80% entre as previsões do modelo e os valores reais, o que mostra que o modelo consegue prever bem.

Modelando os Dados da aula_01_exemplo_01

- Agora vamos testar o KNN.



Código

```
[39]: # Predição no teste usando o melhor modelo KNN
y_pred_teste_knn = melhor_modelo_knn.predict(x_teste_transformado) # covariáveis de teste

# --- Avaliação ---
print("=== Métricas de Teste KNN ===")
print("Acurácia:", accuracy_score(y_teste, y_pred_teste_knn))
print("MCC:", matthews_corrcoef(y_teste, y_pred_teste_knn))
print("\nRelatório de Classificação:\n", classification_report(y_teste, y_pred_teste_knn))
```



Saída

=== Métricas de Teste KNN ===

Acurácia: 0.8992537313432836

MCC: 0.8022919325102592

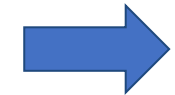
Relatório de Classificação:

	precision	recall	f1-score	support
0	0.86	0.95	0.90	134
1	0.94	0.85	0.89	134
accuracy			0.90	268
macro avg	0.90	0.90	0.90	268
weighted avg	0.90	0.90	0.90	268

- **Precision (0.90):** Na média, considerando o número de amostras em cada classe, 90% das previsões feitas pelo modelo estavam corretas.
- **Recall (0.90):** Em média, considerando o número de amostras em cada classe, o modelo conseguiu alcançar 90% de todas as instâncias reais.

Modelando os Dados da aula_01_exemplo_01

- Agora vamos testar o KNN.



Código

```
[39]: # Predição no teste usando o melhor modelo KNN
y_pred_teste_knn = melhor_modelo_knn.predict(x_teste_transformado) # covariáveis de teste

# --- Avaliação ---
print("=== Métricas de Teste KNN ===")
print("Acurácia:", accuracy_score(y_teste, y_pred_teste_knn))
print("MCC:", matthews_corrcoef(y_teste, y_pred_teste_knn))
print("\nRelatório de Classificação:\n", classification_report(y_teste, y_pred_teste_knn))
```



Saída

=== Métricas de Teste KNN ===

Acurácia: 0.8992537313432836

MCC: 0.8022919325102592

Relatório de Classificação:

	precision	recall	f1-score	support
0	0.86	0.95	0.90	134
1	0.94	0.85	0.89	134
accuracy			0.90	268
macro avg	0.90	0.90	0.90	268
weighted avg	0.90	0.90	0.90	268

- **F1-score (0.90):** A média ponderada do F1-score indica um ótimo equilíbrio geral de 90% entre o precision e o recall do modelo.

Modelando os Dados da aula_01_exemplo_01

- Agora vamos testar o SVC.

➡
Código

```
[40]: # Predição no teste usando o melhor modelo KNN
y_pred_teste_svc = melhor_modelo_svc.predict(x_teste_transformado) # covariáveis de teste

# --- Avaliação ---
print("=== Métricas de Teste SVC ===")
print("Acurácia:", accuracy_score(y_teste, y_pred_teste_svc))
print("MCC:", matthews_corrcoef(y_teste, y_pred_teste_svc))
print("\nRelatório de Classificação:\n", classification_report(y_teste, y_pred_teste_svc))
```

```
=== Métricas de Teste SVC ===
Acurácia: 0.9440298507462687
MCC: 0.8910670910179896
```

Relatório de Classificação:

	precision	recall	f1-score	support
0	0.91	0.99	0.95	134
1	0.98	0.90	0.94	134
accuracy			0.94	268
macro avg	0.95	0.94	0.94	268
weighted avg	0.95	0.94	0.94	268

➡
Saída

- **Acurácia:** temos uma taxa de acerto de 94%, o que aparentemente indica que o modelo consegue prever bem.
- **MCC:** temos uma correlação positiva forte de 89% entre as previsões do modelo e os valores reais, o que mostra que o modelo consegue prever bem.

Modelando os Dados da aula_01_exemplo_01

- Agora vamos testar o SVC.

➡
Código

```
[40]: # Predição no teste usando o melhor modelo KNN
y_pred_teste_svc = melhor_modelo_svc.predict(x_teste_transformado) # covariáveis de teste

# --- Avaliação ---
print("=== Métricas de Teste SVC ===")
print("Acurácia:", accuracy_score(y_teste, y_pred_teste_svc))
print("MCC:", matthews_corrcoef(y_teste, y_pred_teste_svc))
print("\nRelatório de Classificação:\n", classification_report(y_teste, y_pred_teste_svc))
```

=== Métricas de Teste SVC ===
Acurácia: 0.9440298507462687
MCC: 0.8910670910179896

Relatório de Classificação:

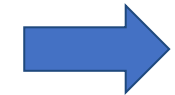
	precision	recall	f1-score	support
0	0.91	0.99	0.95	134
1	0.98	0.90	0.94	134
accuracy			0.94	268
macro avg	0.95	0.94	0.94	268
weighted avg	0.95	0.94	0.94	268

➡
Saída

- **Precision (0.95):** Na média, considerando o número de amostras em cada classe, 95% das previsões feitas pelo modelo estavam corretas.
- **Recall (0.94):** Em média, considerando o número de amostras em cada classe, o modelo conseguiu alcançar 94% de todas as instâncias reais.

Modelando os Dados da aula_01_exemplo_01

- Agora vamos testar o SVC.



Código

```
[40]: # Predição no teste usando o melhor modelo KNN
y_pred_teste_svc = melhor_modelo_svc.predict(x_teste_transformado) # covariáveis de teste

# --- Avaliação ---
print("=== Métricas de Teste SVC ===")
print("Acurácia:", accuracy_score(y_teste, y_pred_teste_svc))
print("MCC:", matthews_corrcoef(y_teste, y_pred_teste_svc))
print("\nRelatório de Classificação:\n", classification_report(y_teste, y_pred_teste_svc))
```

```
=== Métricas de Teste SVC ===
Acurácia: 0.9440298507462687
MCC: 0.8910670910179896
```

```
Relatório de Classificação:
```

	precision	recall	f1-score	support
0	0.91	0.99	0.95	134
1	0.98	0.90	0.94	134

accuracy			0.94	268
macro avg	0.95	0.94	0.94	268
weighted avg	0.95	0.94	0.94	268

- **F1-score (0.94):** A média ponderada do F1-score indica um ótimo equilíbrio geral de 94% entre o precision e o recall do modelo.

Modelando os Dados da aula_01_exemplo_01

- Agora vamos testar o LR.

→
Código

```
[41]: # Predição no teste usando o melhor modelo KNN
y_pred_teste_lr = melhor_modelo_lr.predict(x_teste_transformado) # covariáveis de teste

# --- Avaliação ---
print("=== Métricas de Teste RL ===")
print("Acurácia:", accuracy_score(y_teste, y_pred_teste_lr))
print("MCC:", matthews_corrcoef(y_teste, y_pred_teste_lr))
print("\nRelatório de Classificação:\n", classification_report(y_teste, y_pred_teste_lr))
```

→
Saída

```
=== Métricas de Teste RL ===
Acurácia: 0.914179104477612
MCC: 0.8283812762533432
```

```
Relatório de Classificação:
              precision    recall  f1-score   support

     0       0.91      0.92      0.91       134
     1       0.92      0.91      0.91       134

 accuracy          0.91
 macro avg         0.91
 weighted avg      0.91
```

- **Acurácia:** temos uma taxa de acerto de 91%, o que aparentemente indica que o modelo consegue prever bem.
- **MCC:** temos uma correlação positiva forte de 82% entre as previsões do modelo e os valores reais, o que mostra que o modelo consegue prever bem.

Modelando os Dados da aula_01_exemplo_01

- Agora vamos testar o LR.

→
Código

```
[41]: # Predição no teste usando o melhor modelo KNN
      y_pred_teste_lr = melhor_modelo_lr.predict(x_teste_transformado) # covariáveis de teste

      # --- Avaliação ---
      print("=== Métricas de Teste RL ===")
      print("Acurácia:", accuracy_score(y_teste, y_pred_teste_lr))
      print("MCC:", matthews_corrcoef(y_teste, y_pred_teste_lr))
      print("\nRelatório de Classificação:\n", classification_report(y_teste, y_pred_teste_lr))
```

→
Saída

```
=== Métricas de Teste RL ===
Acurácia: 0.914179104477612
MCC: 0.8283812762533432

Relatório de Classificação:
              precision    recall  f1-score   support

     0       0.91         0.92         0.91         134
     1       0.92         0.91         0.91         134

 accuracy          0.91
 macro avg         0.91
 weighted avg      0.91
```

- **Precision (0.91):** Na média, considerando o número de amostras em cada classe, 91% das previsões feitas pelo modelo estavam corretas.
- **Recall (0.91):** Em média, considerando o número de amostras em cada classe, o modelo conseguiu alcançar 91% de todas as instâncias reais.

Modelando os Dados da aula_01_exemplo_01

- Agora vamos testar o LR.

→
Código

```
[41]: # Predição no teste usando o melhor modelo KNN
y_pred_teste_lr = melhor_modelo_lr.predict(x_teste_transformado) # covariáveis de teste

# --- Avaliação ---
print("=== Métricas de Teste RL ===")
print("Acurácia:", accuracy_score(y_teste, y_pred_teste_lr))
print("MCC:", matthews_corrcoef(y_teste, y_pred_teste_lr))
print("\nRelatório de Classificação:\n", classification_report(y_teste, y_pred_teste_lr))
```

→
Saída

```
=== Métricas de Teste RL ===
Acurácia: 0.914179104477612
MCC: 0.8283812762533432

Relatório de Classificação:
              precision    recall  f1-score   support

     0       0.91      0.92      0.91       134
     1       0.92      0.91      0.91       134

 accuracy          0.91          0.91          0.91          268
 macro avg         0.91          0.91          0.91          268
 weighted avg      0.91          0.91          0.91          268
```

- **F1-score (0.91):** A média ponderada do F1-score indica um ótimo equilíbrio geral de 91% entre o precision e o recall do modelo.

Considerações Finais

- Vamos analisar uma tabela resumo do treino e teste de algumas métricas métricas:

Modelo	Conjunto	Acurácia	Precision (Média Ponderada)	Recall (Média Ponderada)	F1-Score (Média Ponderada)
RL	Treino	0.91	0.91	0.91	0.91
	Teste	0.91	0.91	0.91	0.91
AD	Treino	0.93	0.94	0.93	0.93
	Teste	0.93	0.94	0.93	0.93
SVC	Treino	0.94	0.94	0.94	0.94
	Teste	0.94	0.95	0.94	0.94
KNN	Treino	0.90	0.90	0.90	0.90
	Teste	0.90	0.90	0.90	0.90

Considerações Finais

- A **Árvore de Decisão** e o **SVC** tem os melhores valores tanto no treino quanto no teste.

Modelo	Conjunto	Acurácia	Precision (Média Ponderada)	Recall (Média Ponderada)	F1-Score (Média Ponderada)
RL	Treino	0.91	0.91	0.91	0.91
	Teste	0.91	0.91	0.91	0.91
AD	Treino	0.93	0.94	0.93	0.93
	Teste	0.93	0.94	0.93	0.93
SVC	Treino	0.94	0.94	0.94	0.94
	Teste	0.94	0.95	0.94	0.94
KNN	Treino	0.90	0.90	0.90	0.90
	Teste	0.90	0.90	0.90	0.90

Qual escolher?

Considerações Finais

- **Qual modelo escolher?**
- **A Árvore de Decisão (AD)** é considerada um modelo altamente interpretável , já que o caminho para a previsão de uma amostra pode ser rastreado e visualizado através das suas ramificações.
- É possível **que um leigo entenda exatamente as regras de negócio que levaram a uma decisão.**
- Note que para o **SVC É extremamente difícil explicar o porquê de uma predição individual**, logo vamos recorrer ao modelo mais simples (Navalha de Ockham).
- Na próxima aula entraremos mais afundo sobre a interpretabilidade do modelo, mostrando que existem alternativas viáveis para que possamos compreender de que forma as variáveis impactam no modelo.

Disclaimer: propriedade intelectual

Este material foi criado pela professora Roberta Moreira Wichmann e é de sua propriedade intelectual.

É destinado exclusivamente ao uso dos alunos para fins educacionais no contexto das aulas.

Qualquer reprodução, distribuição ou utilização deste material, no todo ou em parte, sem a expressa autorização prévia da autora, é estritamente proibida.

O não cumprimento destas condições poderá resultar em medidas legais.

Referências Bibliográficas

- ABU-MOSTAFA, Yaser S.; MAGDON-ISMAIL, Malik; LIN, Hsuan-Tien. Learning from Data: A Short Course. Pasadena: California Institute of Technology (AMLBook), 2012.
- DEMÉTRIO, Clarice Garcia Borges; ZOCCHI, Sílvio Sandoval. Modelos de regressão. Piracicaba: Departamento de Ciências Exatas, ESALQ/USP, 2011. Disponível em: https://www.researchgate.net/publication/266233241_Modelos_de_Regressao. Acesso em: 23 set. 202
- GERON, Aurélien. Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems. 2. ed. Sebastopol, CA: O'Reilly Media, 2019.
- HARRIS, C. R. et al. Array programming with NumPy. Nature, v. 585, p. 357–362, 2020. DOI: 10.1038/s41586-020-2649-2.
- HASTIE, Trevor; TIBSHIRANI, Robert; FRIEDMAN, Jerome. The elements of statistical learning: data mining, inference, and prediction. 2. ed. New York: Springer, 2009.
- KAPOOR, Sayash; NARAYANAN, Arvind. Leakage and the reproducibility crisis in machine-learning-based science. Patterns, v. 4, n. 9, 2023.

Referências Bibliográficas

- IZBICKI, Rafael; DOS SANTOS, Tiago Mendonça. Aprendizado de máquina: uma abordagem estatística. Rafael Izbicki, 2020.
- MORETTIN, Pedro Alberto; SINGER, Júlio da Motta. Estatística e ciência de dados. 2. ed. Rio de Janeiro: LTC, 2022.
- PEDREGOSA, F. et al. Scikit-learn: Machine Learning in Python. Journal of Machine Learning Research, v. 12, p. 2825–2830, 2011.
- PYTHON SOFTWARE FOUNDATION. Python Language Reference. Disponível em: <https://docs.python.org/3/reference/index.html>. Acesso em: 10 set. 2025.
- THE PANDAS DEVELOPMENT TEAM. pandas-dev/pandas: Pandas. Zenodo, 2024. Disponível em: <https://doi.org/10.5281/zenodo.10537285>. Acesso em: 10 set. 2025.

Referências Bibliográficas

- VON LUXBURG, Ulrike; SCHÖLKOPF, Bernhard. Statistical Learning Theory: Models, Concepts, and Results. In: GABBAY, D. M.; HARTMANN, S.; WOODS, J. H. (eds.). Handbook of the History of Logic, vol. 10: Inductive Logic. Amsterdam: Elsevier North Holland, 2011. p. 651–706. DOI: 10.1016/B978-0-444-52936-7.50016-1.

Classificação no Aprendizado Supervisionado: Métodos e Avaliação

Obrigada!

Profa. Dra. Roberta Wichmann

roberta.wichmann@idp.edu.br



INSTITUTO BRASILEIRO DE ENSINO,
DESENVOLVIMENTO E PESQUISA