




Ciclo 07 - Algoritmos baseado em árvores

Fundamentos Machine Learning


- [Introdução ao Decision Tree](#)
 - [Funcionamento](#)
 - [Treinamento da decision tree](#)
 - [Impureza](#)
 - [Os 6 passos do treinamento da Decision Tree](#)
- [Métricas de avaliação I: Curva Precision x Recall](#)
 - [Precision / Recall Trade-Off](#)
 - [Curva Precision x Recall](#)
- [Métricas de avaliação II: ROC Curve](#)
- [Métricas de Avaliação III: F1-Score](#)
- [Decision Tree Regressor](#)
- [Random Forest](#)

Introdução ao Decision Tree

A Árvore de Decisão é um algoritmo de Machine Learning utilizado tanto para regressão quanto para a classificação, na solução de problemas complexos. O funcionamento da Árvore de Decisão é baseado em dividir o conjunto de dados em subconjuntos menores, de forma recursiva, até chegar a um conjunto de dados homogêneo ou até o limite máximo de crescimento ser atingido. Para isso, é necessário escolher a melhor variável para dividir o conjunto de dados em cada etapa, utilizando critérios como o Gini, Entropia ou Ganho de Informação.

 As árvores de decisão conseguem endereçar problemas não-lineares devido ao seu baixo viés

As Decision Trees são fáceis de visualizar e interpretar, fazendo delas uma escolha popular para análise exploratória dos dados e explicação do modelo. Os nós da árvore representam pontos de decisão baseado nos valores das features, enquanto que as extremidades representam o resultado daquelas decisões. Uma das vantagens das Decision Trees é que elas conseguem lidar com dados categóricos e numéricos, e também valores faltantes. No entanto, elas tem a tendência de overfittar se a árvore for muito complexa, portanto é importante usar técnicas de regularização como “pruning” (aparamento) e “maximum depth” (profundidade máxima).

 Decision Trees podem ser utilizadas para problemas de classificação com classes binárias ou multi-classes e problemas de regressão. Ela também é comumente utilizada para seleção de features, pois ela consegue identificar as features mais importantes para a tarefa em questão baseado na sua importância na árvore.

Funcionamento

As Decision Trees divide o conjunto de dados recursivamente em subconjuntos baseado no valor de suas features, com o objetivo de minimizar a entropia ou a impureza de cada subconjunto (a impureza nos diz o quão homogênea é uma separação, isto é, se ela conseguiu separar bem as classes).

Uma Árvore de Decisão possui os seguintes componentes:

- Raiz (Root node):** O nó mais ao topo que representa o dataset inteiro e contém todas as possíveis features.
- Nós Internos (Internal node):** Os nós no meio da árvore que representam uma feature e contém a regra de decisão na qual a divisão em subgrupos é baseado.
- Galhos (Branches):** As linhas que conectam os nós e representam uma decisão ou caminho que leva a outro nó.
- Folhas (Leaf nodes):** Os nós mais abaixo da árvore e representam a decisão final do algoritmo, sendo ela a classe da observação ou o valor predito (regressão).
- Critério de Divisão (Splitting criteria):** A regra ou critério para determinar em qual feature a divisão será feita em cada nó interno. Alguns dos critérios de divisão são:
 - Ganho de Informação
 - Impureza de Gini
 - Teste do Chi-Quadrado


<https://external-content.duckduckgo.com/iu/?u=https%3A%2F%2Fk21academy.com%2Fwp-content%2Fuploads%2F2020%2F12%2Fdecision.png&f=1&nofb=1&ipt=88a70c03abef149ab7d5ccdb89d2b219876984cce35a00b6c90800c3c6b9ae5c&ipo=images>

Para entendermos melhor o funcionamento de uma Árvore de Decisão, vamos usar o conjunto de dados ao lado. Nele está contido o dados dos clientes de um determinado banco e devemos classificar baseado nas features se devemos liberar ou não o empréstimo para cada cliente.

Name	Age	Salary	Number of Kids	Loan
John Smith	25	\$ 2600	0	Yes
Mary Johnson	45	\$ 1900	3	No

Primeiro a árvore de decisão olha para o conjunto de dados inteiro e seleciona as features que melhor separa os dados entre as diferentes classes (**yes** ou **no**). Nesse processo a árvore seleciona um par (feature, valor), faz a divisão baseada nele e verifica qual é impureza nesses subconjuntos resultantes e guarde esse valor. O split vai ocorrer no par que retornar a menor impureza, esse par define o chamado **Critério de Divisão** e ele é responsável por criar novos galhos, o critério é guardado no **Nó interno**.

Name	Age	Salary	Number of Kids	Loan
Michael Jordan	32	\$ 2200	2	No
...
Chris Rock	33	\$ 5000	2	Yes

A árvore de decisão faz subdivisões do conjunto original em duas etapas, seleciona uma feature e avalia qual valor dessa feature selecionada retorna a menor impureza, esse processo é repetido até se alcançar a pureza máxima ou até que os parâmetros escolhidos na definição do modelo sejam satisfeitos.

No nosso exemplo o primeiro critério de divisão é feito com base no par **(age, 30)**, ou seja, o split é feito em 2 subconjuntos:

- $age > 30$
- $age \leq 30$

Foi nesse par que árvore conseguiu dividir o conjunto original com maior grau de pureza, isso é, os subconjuntos resultantes com maior homogeneidade possível.

Após esse split, cada subconjunto será reavaliado e uma novo critério de decisão é encontrado para cada. No primeiro, o critério é **(salary, \$2500)**, portanto o algoritmo verificou que esse split é o que divide o subconjunto com menor impureza. O mesmo ocorre com o segundo subconjunto, só que dessa vez o split é realizado seguindo o par **(Number of Kids, 2)**, ou seja, esse conjunto é dividido em 2 menores tendo como parâmetro de divisão o número de crianças.


O raciocínio continua até que se atinja o maior grau de pureza em cada subconjunto ou até que os parâmetros de definição do modelo seja atingido, como profundidade da árvore, ou nível mínimo de pureza em cada folha (subconjunto final).

https://external-content.duckduckgo.com/iu/?u=https%3A%2F%2Fmiro.medium.com%2Fmax%2F1280%2F0*4QE-0kavxXfzF_bR.png&f=1&nofb=1&ipt=b04c340d2eee10d36496fdd1f0f9a517441f694da845b16f7271070a3b03c79a&ipo=images

No processo de treinamento da decision, o conjunto original é dividido em subconjuntos de acordo com os critérios de decisão encontrados. Podemos visualizar os critérios de divisão e o subconjuntos gerados observando espaço das features ou “feature space”.

No feature space, é plotado os datapoints com a devida classe explicitada, e a cada critério de decisão, um corte é feito nesse espaço, de modo a dividi-lo em 2 regiões com o maior nível de homogeneidade possível. No gráfico ao lado, por exemplo, o primeiro split foi feito de acordo com a regra $x_1 < 2$, isso por que, com esse é o corte que melhor separa as duas classes do dataset.

<https://external-content.duckduckgo.com/iu/?u=https%3A%2F%2Ftse1.mm.bing.net%2Fth%3Fid%3DOIP.3lQbjipGTuA0EZorZqU6kgHaDI%26pid%3DApi&f=1&ipt=1d434be512c20918f29e0465dbc3d006899876c33a3ab2c3aad8a81e6d3f2692&ipo=images>

As regiões de corte de uma árvore de decisão é paralelo aos eixos definido pelas features. As regiões obtidas após os cortes no espaço são chamadas de regiões de classificação.

Treinamento da decision tree

Os algoritmos de árvore de decisão exigem que o conjunto de dados seja dividido em subconjuntos menores em cada nó da árvore. Para isso, é necessário escolher o melhor atributo para dividir o conjunto de dados. A escolha é feita com base em uma medida de impureza. O objetivo é minimizar a impureza do conjunto de dados nos subconjuntos resultantes. As medidas mais comuns de impureza são a entropia, o índice de Gini e o ganho de informação.

O atributo que resulta na maior redução de impureza é escolhido para a divisão. Esse processo é repetido recursivamente em cada nó da árvore até que os subconjuntos resultantes sejam puros ou até que um critério de parada seja atingido.

Impureza

A impureza determina o quão misturados estão os dados em relação à classe de saída. Por exemplo, um conjunto de dados com 100 elementos, 50 da classe A e 50 da classe B, são subdivididos subsequentemente ao ponto dos 100 elementos originais serem distribuídos entre as folhas; se uma dessas folhas possui 10 elementos da classe A e 0 elementos da classe B, essa folha possui um grau de impureza de zero, pois todos os seus elementos pertencem a uma única classe. Por outro lado, se uma dessas folhas possui 3 elementos das classe A e 5 elementos da classe B, essa folha tem um certo grau de impureza, pois 37.5% dos seus dados pertencem a classe A e 62.5% pertencem a classe B.

Para determinar o grau de impureza de uma folha, são utilizadas algumas medidas, como por exemplo:

- O critério (índice) de Gini**
- Entropia**
- Ganho de Informação**

O Critério (índice) de Gini

O índice de Gini mede a impureza de um subconjunto de dados. Ele calcula a probabilidade de que duas observações selecionadas aleatoriamente pertençam a classes diferentes, portanto, quanto mais próximo de zero o valor de G_i for, melhor o split, pois índice de $G_i = 0$ indica que a probabilidade de observarmos duas classes diferentes no i -ésimo nó da árvore é nula.

O valor do índice de Gini varia entre 0 e 1:

- $G_i = 0 \rightarrow$ todas as observações pertencem a mesma classe.
- $G_i \approx 1 \rightarrow$ indica heterogeneidade.

Na prática nunca observaremos $G_i = 1$, pois não é possível garantir que sempre encontraremos duas classes distintas ao selecionarmos elas aleatoriamente, mesmo que o subconjunto tivesse sido distribuído igualmente (situação que maximiza G_i).

Fórmula:

$$G_i = 1 - \sum_{k=1}^n p_{i,k}^2$$

Onde $p_{i,k}$ representa a probabilidade de encontrarmos um elemento da classe k no i -ésimo nó. Essa probabilidade é encontrada fazendo a divisão entre o número de elementos da classe k pelo número total de elementos contidos no i -ésimo nó.

$$p_{i,k} = \frac{\# \text{ classe } k}{\# \text{ total}}$$



Quando é utilizado o critério de Gini existe a tendência da árvore crescer apenas para um lado, isso é, um dos galhos da árvore tende a ser maior que o outro.

Entropia

Nos algoritmo de Árvore de Decisão, a entropia é utilizada para medir a impureza ou desordem de um conjunto de dados.

$$E_i = - \sum_{k=1}^n p_{i,k} \log_2(p_{i,k})$$

Ganho de Informação

O ganho de informação é uma medida de quanta informação é ganha sobre a classe de saída ao dividir um conjunto de dados de acordo com um atributo específico.

A fórmula do ganho de informação para o cálculo da impureza de uma folha da Decision Tree é:

$$IG(D_p, a) = I(D_p) - \sum_{j=1}^v \frac{N_j}{N_p} I(D_j)$$

Onde:

- a é o atributo considerado.
- v é o número de valores distintos de a .
- N_p é o número total de exemplos em D_p .
- N_j é o número total de exemplos em D_j .
- D_j é o subconjunto de D_p com valores distintos de a a j .
- $I(D_j)$ é a impureza de D_j .
- $I(D_p)$ é a impureza de D_p .

A escolha do atributo na separação

O algoritmo Decision Tree realiza o treinamento separando o conjunto de dados em duas partes, segundo um único atributo (k) e um valor (t_k), por exemplo, $k = \text{"Petal Lenght"}$ e $t_k < 2.45 \text{ cm}$, ou seja, "Petal Lenght < 2.45 cm".

Ao realizar as subdivisões do conjunto de dados, o algoritmo busca minimizar o valor da seguinte função matemática, também conhecido como **função de custo**:

$$J(k, t_k) = \frac{m_{left}}{m} G_{left} + \frac{m_{right}}{m} G_{right}$$

Onde:

- $G_{left / right}$ mede a impureza dos subconjuntos à direita e à esquerda.
- $m_{left / right}$ é o número de exemplos nos conjuntos da direita ou da esquerda.

Os 6 passos do treinamento da Decision Tree

1. Escolha um atributo (feature) do conjunto de dados.
2. Para cada valor do atributo selecionado, use a função de custo para encontrar o valor da impureza da separação.
3. Repita os passos 1 e 2 para todas as combinações de atributo e valores, a fim de encontrar a combinação atributo-valor que retorne o menor valor da função custo.
4. Uma vez definido o par atributo-valor, faça a separação do conjunto de dados em dois nós filhos.

- 5. Repita os passos de 1 a 3 até encontrar a segunda combinação atributo-valor para causar uma nova separação dos dados.
- 6. Repita o 5 até os valores dos parâmetros serem atendidos.

Métricas de avaliação I: Curva Precision x Recall

A Precisão e o Recall são duas métricas importantes para analisar a performance de um algoritmo de machine learning. Na prática elas respondem as seguintes perguntas:

Dos itens que o algoritmo classificou como positivo, quantos deles eram realmente positivos?

$$precision = \frac{\#TP}{\#TP + \#FP}$$

De todos os itens positivos do dataset (que de fato pertencem a classe positiva), quantos deles o algoritmo classificou corretamente?

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

A identificação das classes positiva e negativa é feita pela pessoa que está estudando o fenômeno, e a diferença entre as classificações feitas pelo modelo e o valor real pode ser melhor entendidos com o auxílio da matriz de confusão:

		Previsão do Rótulo (modelo)	
		Classe Positiva (P)	Classe Negativa (N)
Rotúlo Real	Classe Positiva	True Positive (TP)	False Negative (FN)
	Classe Negativa	False Positive (FP)	True Negative (TN)

- **Total de previsões** = P + N
- **True Positive** = Classificação correta da classe positiva.
- **True Negative** = Classificação correta da classe negativa.
- **False Positive** = Classificação errada da classe positiva.
- **False Negative** = Classificação errada da classe negativa.

Precision / Recall Trade-Off


Em alguns problemas de negócio, a métrica mais importante é a precisão, enquanto em outros a métrica mais importante é o recall.

Por exemplo, a empresa onde você trabalha desenvolve uma plataforma de hospedagem de vídeos de produtores de conteúdo com foco em crianças. Sua principal responsabilidade com Cientista de Dados da empresa, é detectar os vídeos seguros para as crianças assistirem.

Nesse cenário, seria preferível um classificador que rejeitasse muito bons vídeos classificando-os como “inapropriados”, mas conseguisse classificar todos os vídeos realmente inapropriados para menores como “inapropriados” (alta precisão).

Por outro lado, imagine que você trabalhe em uma empresa de vigilância e sua principal responsabilidade como Cientista de Dados é treinar um classificador para detectar ladrões em imagens de vigilância: provavelmente está tudo bem se o seu classificador tiver apenas 30% precisão (muitos falso positivos), desde que tenha 99% de recall.

Nesse caso, os seguranças receberiam alguns alertas falsos, mas quase todos os ladrões seriam pegos.



Em resumo, você não pode ter as duas métricas altas ao mesmo tempo, aumentar a precisão reduz o recall e vice-versa. Isso é chamado de trade-off de precisão / recall.

Portanto, como escolher o valor para as métricas de Precisão e para o Recall?

Curva Precision x Recall

Os classificadores de Machine Learning, originalmente, classificam as observações do conjunto de dados como a probabilidade de um ponto pertencer a uma classe ou a outra.

É de responsabilidade do Cientista de Dados definir um limiar ou “threshold” de probabilidade que atribua a observação para uma classe ou outra, dependendo do valor definido. Por exemplo, se o valor do limiar é de 0.80, todas as observações com probabilidade acima de 80% pertencem à classe A, enquanto que as observações abaixo de 80% pertencem à classe B.

Os valores das métricas Precision e Recall mudam de acordo com o valor do limiar de classificação definido pelo Cientista de Dados. Portanto, precisamos de uma ferramenta visual para ajudar na escolha desse Threshold.

Quando temos um **baixo valor de Threshold**, a probabilidade mínima exigida para se classificar uma observação com a classe 1 é pequena e portanto o nosso modelo classificará muitos pontos com essa classe.

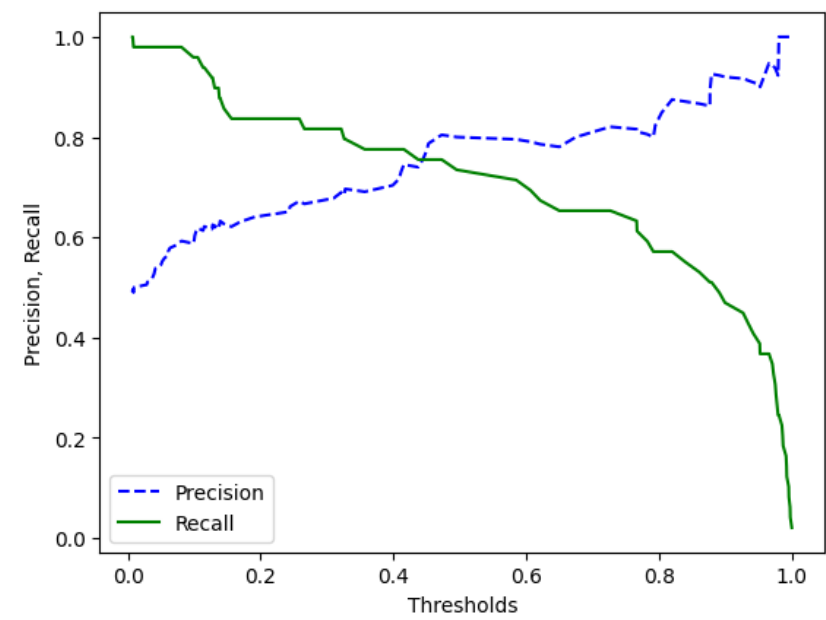
Quando isso acontece, aumenta-se o número de elementos erroneamente classificados com 1 (classe positiva), ou seja, o número de falsos positivos é grande. Nessa situação o denominador da fórmula de precisão se torna um número grande e portanto a precisão do modelo diminui.

$$\downarrow Precision = \frac{\#TP}{\#TP + \uparrow \#FP}$$

Gráfico do Thresholds vs Precision / Recall

O contrário ocorre com o Recall, pois como o modelo classifica muitos pontos com classe positiva, poucos pontos são classificados com a classe negativa e portanto teremos um número baixo de elementos classificados erroneamente com a classe negativa (**false negative** - situação em que um ponto que era de fato positivo foi classificado como negativo).

$$\uparrow Recall = \frac{\#TP}{\#TP + \downarrow \#FN}$$



O inverso ocorre quando **aumentamos o valor do Threshold**, nessa situação o número de falsos positivos diminui (pois aumentamos o requisito mínimo para pertencer a classe positiva, nesse cenário é mais difícil um registro ser classificado como positivo e portanto as chances de o modelo classificar errado a classe positiva diminui) e então a **Precision aumenta**, inversamente o número de falsos negativos aumenta, pois como o número de elementos classificados como positivo diminuiu naturalmente aumentará o número de elementos classificados com a classe negativa, de modo que a classificação errada dentro dessa classe se torna mais comum, e portanto **Recall diminui**.

$$\uparrow Precision = \frac{\#TP}{\#TP + \downarrow \#FP}$$

$$\downarrow Recall = \frac{\#TP}{\#TP + \uparrow \#FN}$$

Percebemos portanto que conforme variamos o Threshold, uma das métricas aumenta e a outra diminui, concluímos então que **Precision e Recall são inversamente proporcionais conforme variamos o Threshold**.

- $\uparrow Threshold \Rightarrow \uparrow Precision \ \& \ \downarrow Recall$
- $\downarrow Threshold \Rightarrow \downarrow Precision \ \& \ \uparrow Recall$

<https://external-content.duckduckgo.com/iu/?u=https%3A%2F%2Fmachinelearningmastery.com%2Fwp-content%2Fuploads%2F2018%2F08%2FPrecision-Recall-Plot-for-a-No-Skill-Classifer-and-a-Logistic-Regression-Model4.png&f=1&nofb=1&ipt=95699ca62d8add24412c0fc2072f7e9f8e3ee2a50fa83dbf29a34faa88f22f2b&ipo=images>


Esse comportamento pode ser observado no gráfico ao lado, nele os valores de Threshold usados para calcular a Precision e o Recall foram ordenados do maior para o menor.

Métricas de avaliação II: ROC Curve

		Previsão do Rótulo (modelo)	
		Classe Positiva (P)	Classe Negativa (N)
Rotúlo Real	Classe Positiva	True Positive (TP)	False Negative (FN)
	Classe Negativa	False Positive (FP)	True Negative (TN)

No **TPR (True Positive Rate)** estamos interessados na taxa de acertos dos pontos que realmente pertencem a classe positiva, portanto, dividimos a quantidades de pontos da classe positiva que o algoritmo acertou ($\#TP$) pelo número total de pontos no dataset que de fato pertencem a classe positiva ($\#TP + \#FN$).

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

 **TPR** - Taxa de **acertos** da classe **positiva**.


A Curva ROC (Receiver Operating Characteristic) é a representação gráfica da performance de um modelo de classificação binária.

A curva plota o **True Positive Rate (TPR)** contra o **False Positive Rate (FPR)** para vários valores de thresholds decisão, isso é, a porcentagem mínima de certeza que tem modelo sobre a predição de uma classe.

- **Total de previsões** = P + N
- **True Positive** = Classificação correta da classe positiva.
- **True Negative** = Classificação correta da classe negativa.
- **False Positive** = Classificação errada da classe positiva.
- **False Negative** = Classificação errada da classe negativa.

No **FPR (False Positive Rate)** estamos interessados em determinar a taxa de erros dos pontos que realmente pertencem a classe negativa, em outras palavras, qual a porcentagem de pontos que realmente pertence a classe negativa ($\#FP + \#TN$) o algoritmo erroneamente classificou como positiva ($\#FP$).

$$FPR = \frac{\#FP}{\#FP + \#TN}$$

 **FPR** - Taxa de **erros** da classe **negativa**.


<https://external-content.duckduckgo.com/iu/?u=https%3A%2F%2Fvitalflux.com%2Fwp-content%2Fuploads%2F2020%2F09%2FScreenshot-2020-09-01-at-3.44.15-PM.png&f=1&nofb=1&ipt=092011acd911fc6c98114bec126efd37940d31146675524ad17d6553b6184433&ipo=images>


Para se criar a Curva ROC as predições do modelo são ordenadas do maior para menor de acordo com a probabilidade de pertencerem a classe positiva. O threshold é então variado de 0 a 1, e o TPR e FPR são calculados para cada valor de threshold. O resultado é uma curva que plota o TPR no eixo y e o FPR no eixo x .

Uma boa classificação deve ter um alto TPR e um baixo FPR . O modelo ideal teria $TPR = 1$ (o modelo prevê corretamente todas as classes positivas) e um valor de $FPR = 0$ (o modelo não classifica erroneamente nenhuma observação da classe negativa). Essa combinação resulta em um ponto no canto superior esquerdo do gráfico.

ROC Curves & AUC: The Ultimate Guide | Built In

ROC curves are commonly used in classification models. This guide will help you to truly understand how ROC curves and AUC work together.

 <https://builtin.com/data-science/roc-curves-auc>



Métricas de Avaliação III: F1-Score

▼ Média harmônica


A média harmônica é um tipo de medida estatística usada para calcular a média de grandezas inversamente proporcionais. Ela é calculada dividindo-se o número de elementos pela soma dos inversos desses elementos.

$$\text{Média Harmônica} = \frac{n}{\frac{1}{x_1} + \frac{1}{x_2} + \dots + \frac{1}{x_n}}$$

Onde:

- n é o número de elementos.
- x_1, x_2, \dots, x_n são os valores dos elementos.

A medida F1-score é a média harmônica da precisão e recall, uma vez que as duas medidas são inversamente proporcionais.



O F1-score sempre terá o valor mais próximo da métrica com pior avaliação

Fórmula do F1-score

$$F_1 = \frac{2}{\frac{1}{\text{precision}} + \frac{1}{\text{recall}}} = 2 \times \frac{\text{precision} \times \text{recall}}{\text{precision} + \text{recall}}$$

Características do F1-score:

- Equilíbrio entre precisão e recall:** O F1-score é uma medida que combina tanto a precisão quanto o recall de um modelo de classificação binária, e é particularmente útil em casos em que ambas as métricas sejam importantes.
- Interpretação intuitiva:** O F1-score é uma métrica fácil de interpretar, pois fornece um valor único que indica o desempenho geral do modelo. Além disso, o valor do F1-score é compreensível para usuários não técnicos, o que pode ser útil em casos que a avaliação do modelo precisa ser comunicada a pessoas que não possuem conhecimento técnico.

Como interpretar o F1-score:

O intervalo do F1-score varia de 0 a 1 ($F_1 \in [0, 1]$), onde 0 indica um desempenho muito ruim e 1 um desempenho excelente do modelo. O valor 0 é obtido quando a precision ou o recall é igual a zero, enquanto 1 é obtido quando a precision e o recall são iguais a 1.

Um **valor alto de F1-score** indica que o modelo tem um bom equilíbrio entre a precisão e o recall, o que significa que ele está acertando muitos exemplos positivos e negativos. Por outro lado, um **valor baixo de F1-score** indica que o modelo está tendo dificuldades em classificar corretamente as instâncias positivas ou negativas, ou que está tendo um **desempenho desequilibrado entre as métricas**.

<https://towardsdatascience.com/micro-macro-weighted-averages-of-f1-score-clearly-explained-b603420b292f>

Decision Tree Regressor

O algoritmo Decision Tree Regressor é utilizado para problemas de regressão. O processo de treinamento é semelhante ao de problema de classificação, porém a escolha do atributo de separação é feita de forma a minimizar o erro entre os pontos da folha e o seu valor médio.

O algoritmo Decision Tree Regressor possui 6 passos para o seu treinamento, que consistem em escolher um atributo do conjunto de dados e, para cada possível valor do atributo selecionado, encontrar o valor da impureza de separação. repetindo os passos para todas as combinações de atributo e valores a fim de encontrar a combinação atributo-valor que retorne o menor valor da função custo da Decision Tree. Uma vez definido o par atributo-valor, faz-se a separação do conjunto de dados em dois nós filhos. Repete-se o processo até que os valores dos parâmetros sejam atendidos.



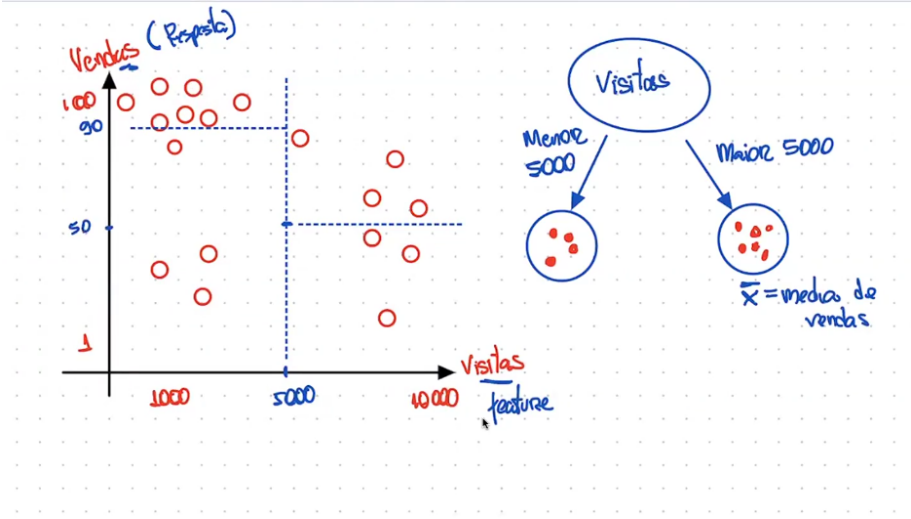
A árvore de decisão é um algoritmo com alto viés e baixa variância.

Suponha que queremos estimar o número de vendas que um e-commerce terá baseado no número de visitas que site recebe. Podemos utilizar o algoritmo Decision Tree Regressor para resolver esse problema. Nesse caso estaremos prevendo o valor da feature **Vendas** (**Variável Resposta**) a partir do valor da feature **Visitas** (**Feature Preditora**).

Primeiro selecionamos um valor de x (**Vendas**) que melhor divide o conjunto de dados, essa divisão é feita de modo a minimizar uma medida de erro calculada em cima dos subconjuntos resultantes. Nesse exemplo, vamos supor que o valor encontrado seja de $x = 5000$. A partir desse valor, traçamos uma reta vertical que divide o espaço de features em 2. O par (**Visitas, 5000**) é utilizado para fazer o primeiro split da árvore de decisão e partir dele obtemos 2 subconjuntos, um à esquerda da reta vertical $x = 5000$ e outro à direita.

A primeira predição da variável resposta y (**Vendas**) é feita a partir da média desses subconjuntos obtidos após o split. Nesse caso, com apenas um nó, teríamos que o valor predito para vendas seria de:

- $x < 5000 \Rightarrow \bar{y} = 90$
- $x \geq 5000 \Rightarrow \bar{y} = 50$



Após calcularmos o valor da variável preditora, temos que estimar o erro que estamos cometendo em cada subconjunto. Isso é feito usando a métrica do **Mean-Squared Error (MSE)** na qual estimamos o desvio quadrático médio em relação a média do conjunto.

$$MSE = \frac{1}{N} \sum_{i=1}^N (y_i - \bar{y})^2$$

Tanto o subconjunto da esquerda quanto o direito possui um MSE, então o erro obtido pelo split é dados pela soma do MSE de cada subconjunto.

$$Error = MSE_{left} + MSE_{right}$$

💡 O valor da feature preditora, nesse caso x , selecionado para se fazer o split (x_{split}) é aquele para o qual temos o menor valor do *Error*.

Um dos problemas de algoritmos baseados em árvores para problemas de regressão é que eles não são capazes de extrapolar o valor da variável reposta para fora intervalo que a árvore foi treinada. Por exemplo, se no nosso caso treinássemos a árvore com um dataset que o número de vendas variasse no intervalo $y \in [1, 1000]$, o maior valor possível de vendas seria $y_{max} = 1000$, mesmo que o valor da variável preditora x (visitas) fosse um valor muito grande (fora do intervalo usado no treinamento).

💡 Algoritmos baseados em árvores não são capazes de extrapolar o valor da variável resposta para fora do intervalo de treinamento.

Impureza de uma árvore

Como vimos acima o algoritmo Decision Tree também pode ser utilizado para problemas de regressão. Nesse caso, a impureza dos nós é medida utilizando o erro quadrático médio (MSE) ou o erro absoluto médio ao invés dos critérios de impureza utilizados em problemas de classificação.

A fórmula do erro quadrático médio (MSE) para o cálculo da impureza de uma folha da Decision Tree é:

$$MSE = \frac{1}{N} \sum_{i=1}^N (y_i - \bar{y})^2$$

A fórmula do erro absoluto médio (MAE) para o cálculo da impureza de uma folha da Decision Tree é:

$$MAE = \frac{1}{N} \sum_{i=1}^N |y_i - \bar{y}|$$

Os 5 passos de treinamento da Decision Tree Regressor

No algoritmo Decision Tree Regressor, a impureza dos nós é medida utilizando o erro quadrático médio (MSE). A redução da impureza acontece da seguinte forma:

1. Primeiro, a árvore avalia qual é o melhor atributo para fazer a divisão dos dados em subconjuntos. Essa escolha é feita com base em qual atributo resultaria na maior redução do erro de predição.
2. Após encontrar o melhor atributo, a árvore faz a divisão dos dados em subconjuntos com base nos valores desse atributo.
3. Para cada subconjunto, é calculado o erro quadrático médio (MSE) entre os valores reais e os valores preditos.
4. Em seguida, é calculada a impureza desse nó utilizando a média desses erros quadráticos médios.
5. A árvore continua fazendo a divisão dos subconjuntos recursivamente até que os subconjuntos resultantes sejam puros ou até que um critério de parada seja atingido.

Random Forest

Conceito de Weak Learning

O conceito de de Weak Learning é uma ideia fundamental por trás do algoritmo de Random Forest. Em vez de criar uma única árvore de decisão muito complexa que se ajusta perfeitamente aos dados de treinamento, o algoritmo de Random Forest cria muitas árvores simples, com desempenho ligeiramente melhor do que aleatório.

O termo “Weak” refere-se ao fato de que cada árvore individual pode ter uma precisão limitada, ou seja, são árvores que não conseguem separar perfeitamente os dados de treino. No entanto, a combinação das previsões de todas as árvores individuais resulta em um modelo robusto e geralmente mais preciso.

Isso acontece porque as **árvores individuais são construídas de maneira diferente**, usando diferentes subconjuntos dos dados de treinamento e variáveis de entrada. Além disso, as árvores são treinadas de forma independente, o que significa que cada árvore aprende com os dados de treinamento de maneira diferente.

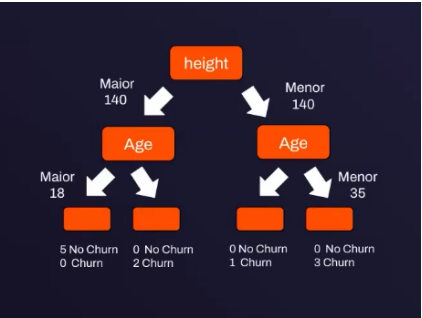
Ao combinar as previsões de todas as árvores individuais, a Random Forest é capaz de reduzir o impacto de erros individuais e produzir um modelo mais geral e preciso. Essa abordagem é especialmente útil quando lidamos com conjuntos de dados grandes e complexos, onde uma única árvore de decisão pode não ser suficiente para capturar todas as nuances e padrões dos dados.

Como a Random Forest funciona?

Imagine que tenhamos um conjunto de dados simples composto por 5 colunas, em que 4 delas contenham informações sobre os clientes (idade, gênero, altura e peso) e uma de rótulo que identifica se o cliente continuará usando o serviço ou não (Churn).

Um algoritmo de árvore comum, iria varrer o conjunto de dados e escolher o par feature-valor que melhor dividisse a variável classificadora (Churn) e esse processo seria repetido até que atingíssemos o grau de pureza desejado em cada folha ou o tamanho da árvore especificado na definição do modelo.

No exemplo da árvore ao lado, conseguimos separar todos os dados do conjuntos após 2 splits. No entanto, se retrainarmos a mesma árvore, usando o mesmo conjunto de testes e os mesmos parâmetros obteríamos sempre o mesmo resultado, pois a nossa Decision Tree sempre terá a mesma visão de mundo.



Dados Originais

Age	Gender	Height (cm)
11	male	140
22	female	165
35	male	163
21	female	168
15	female	170
61	male	175
47	female	160
18	female	162

💡 Por ter uma visão fraca de separação de dados (baseada em um único conjunto), árvores de decisão têm a tendência de overfitar ou underfitar.

Em algoritmos de Random Forest, criaremos árvores mais fracas nas quais induziremos um certo viés ao seu treinamento. Desse modo, cada árvore conseguirá ter uma visão de mundo (registros do conjunto) diferente do conjunto de dados, e portanto cada árvore trará uma opinião diferente na hora de escolher de fato quais registros pertencem ou não a classe Churn. A maneira escolhida para induzirmos o viés é através de uma ferramenta chamada de **bootstrap**, que se trata de uma técnica de amostragem de um conjunto de dados que permite a repetição de registro, desse modo estaremos introduzindo uma variabilidade ao conjunto de treinamento de cada árvore.

No técnica de bootstrap o conjunto de dados é refeito, embaralhando e sorteando os registros do conjunto original, de modo a permitir repetição. No conjunto ao lado, obtido após a técnica de bootstrap, vemos que as linhas pintadas de laranja representam o mesmo registro. Agora nesse conjunto temos **6 registros Churn** e **2 registros Não-Churn**, diferente do original que tinha um balanço de 5 e 3 respectivamente, portanto introduzimos uma variabilidade no conjunto.

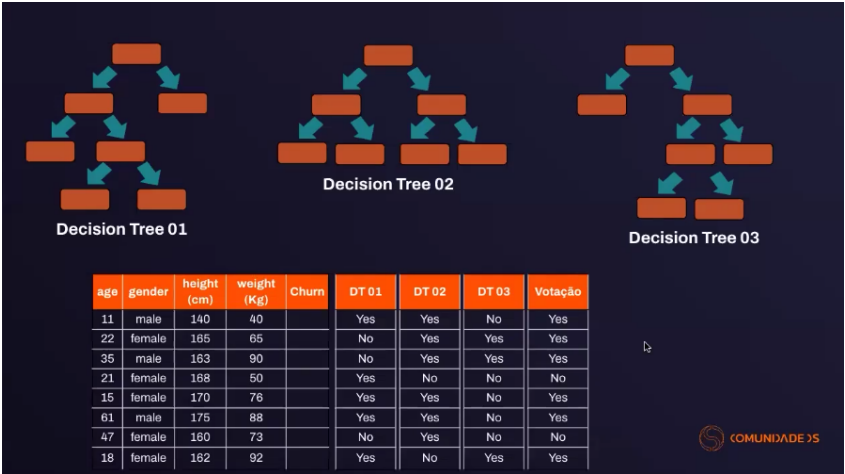
Bootstrapped Data

Age	Gender	Height (cm)	Weight (kg)	Churn
11	male	140	40	Yes
47	female	160	73	Yes
61	male	175	88	Yes
61	male	175	88	Yes
22	female	165	65	No
47	female	160	73	Yes
35	male	163	90	No
18	female	162	92	Yes

Search Space

Após realizar o bootstrap, o algoritmo de Random Forest adiciona mais uma camada de variabilidade ao selecionar aleatoriamente quais colunas do conjunto bootstraped serão utilizadas para treinar o modelo. O treinamento é então feito dentro de um espaço reduzido chamado de **Search Space**. Geralmente o método escolhido para selecionar o número de features que serão incluídas no Search Space é tirando a raiz quadrada do número de features do conjunto original ($N_{search\ space} = \sqrt{N_{original}}$). Nesse caso o nosso Search Space será composto de duas colunas.

Com os dados do Search Space uma nova árvore de decisão é construída. Podemos repetir o processo descrito acima para adicionar mais árvores a nossa Random Forest, onde cada árvore trará uma opinião diferente sobre a classe que cada registro pertence. Para avaliarmos a nossa Random Forest em um conjunto de teste (dados ainda não vistos), nós “perguntamos” para cada árvore da floresta qual classe cada registro do conjunto pertence e a atribuição final é feita para classe que receber mais votos, ou seja, o rótulo na qual a maioria das árvores acreditam que sejam a classe verdadeira.



💡 A Random Forest treina classificadores fracos mas que juntando a opinião deles geram um classificador forte.

Pontos principais:

1. A Random Forest é a composição de resultados das Decision Tree.
2. Em problemas de classificação, usamos o sistema de votação.
3. Em problemas de regressão, usamos a média dos resultados das Decision Tree Regressor.
4. Random Forest é robusta contra outliers.
5. Para o treinamento, não é necessário normalização ou reescala dos atributos.
6. Tanto a Random Forest quanto as Decision Tree não são capazes de extrapolar um valor.