Limpeza de dados e conceitos gerais de ML

No mundo da tecnologia, o aprendizado de máquina (Machine Learning, ML) tem se destacado como uma das áreas mais revolucionárias, especialmente no contexto da ciência de dados. Este material foi criado com o objetivo de te guiar desde os conceitos fundamentais até à aplicação de ML, facilitando a transição do nível básico para o intermediário .

Ao longo deste ebook, veremos conceitos essenciais como features, samples e targets, que formam a base da modelagem em ML. Exploraremos a diferença entre features categóricas e numéricas e a importância de compreender cada tipo para o pré-processamento de dados. Além disso, discutiremos a distinção entre aprendizado supervisionado e não supervisionado, dois paradigmas cruciais que definem a abordagem e os algoritmos a serem utilizados em diferentes cenários de ML.

Um aspecto fundamental que será abordado é a limpeza de dados, uma etapa crítica no processo de ML, que envolve a manipulação e preparação de dados para garantir modelos precisos e eficientes. Também trataremos de conceitos avançados como data leakage, que pode comprometer a validade dos modelos se não for devidamente gerenciado.

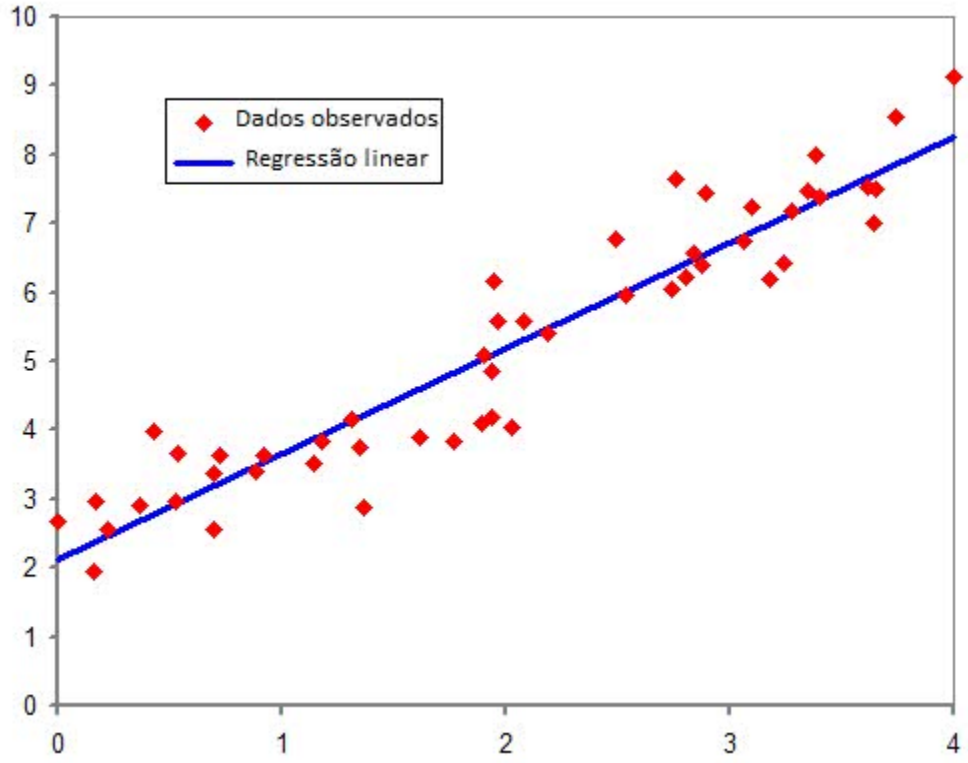
Nosso objetivo com este material é fornecer a você uma compreensão sólida e prática dos conceitos e técnicas de ML, capacitando-o a aplicar esses conhecimentos no desenvolvimento de novas soluções.

Leonardo Carvalho
leofacebook17@gmail.com

O que é um modelo e o que é machine learning

Um modelo matemático é basicamente uma equação projetada para identificar padrões e relações em um conjunto de dados para fazer previsões sobre novas informações. Na prática, isso significa analisar dados disponíveis para descobrir tendências que possam ser usadas para antecipar resultados futuros. Esses modelos são desenvolvidos por meio de algoritmos e técnicas específicas de aprendizado de máquina.

Para ilustrar, imagine que você tem dados de 100 estudantes, incluindo quantas horas eles dedicam aos estudos diariamente.Vamos supor que esses alunos fazem uma prova e depois colocamos em um gráfico a quantidade de horas estudadas versus a nota na prova. No eixo y temos a nota e no eixo x temos a quantidade de horas estudadas. Cada aluno é um pontinhos vermelhos no gráfico abaixo.



Note que se passarmos uma reta por cima desses pontos (reta azul) teríamos uma aproximação boa dos pontos. Nesse caso, se todos os alunos fossem representados por essa reta azul, eu conseguiria dizer, baseado nessa reta, qual a nota o aluno tiraria baseado no tempo de estudos que ele teve. Ou seja, conseguimos prever a nota do aluno baseado na feature “tempo de estudos”.

Veja essa sequência de posts para ter mais clareza no assunto: https://www.instagram.com/p/CquuZriq05Q/?utm_source=ig_web_copy_link&igsh=MzRlODBiNWFlZA==

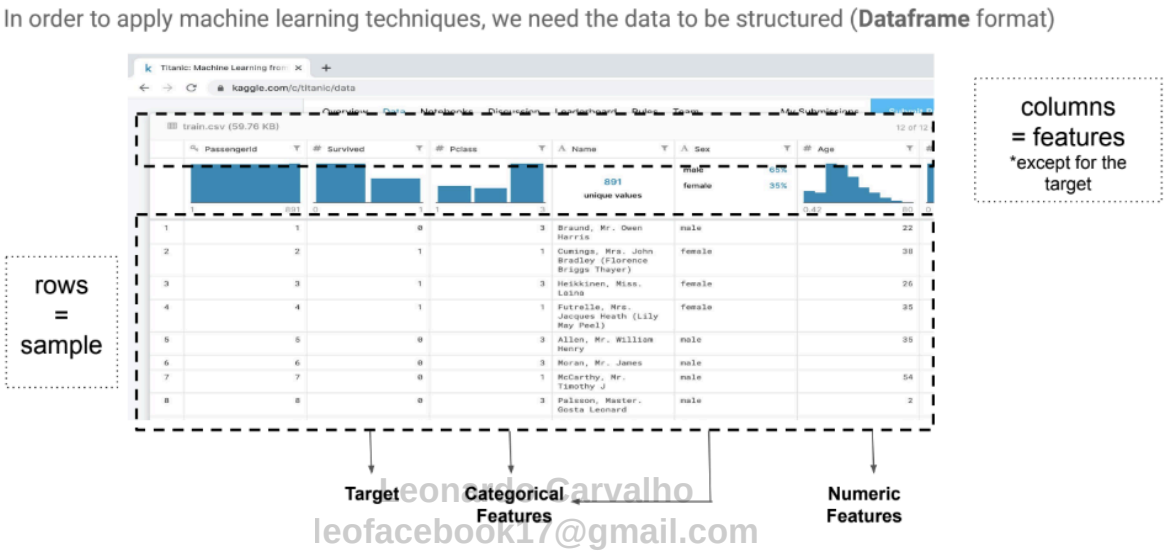
Porém, esses dados que passei acima são sintéticos - inventei esses resultados. Prever o desempenho dos estudantes em uma prova baseando-se somente nessa informação seria complicado. Alguns podem estudar a mesma quantidade de horas, mas ter desempenhos muito diferentes devido a fatores como condição familiar, qualidade da educação e foco em disciplinas específicas. Mesmo com um conjunto de dados mais completo, ainda seria desafiador fazer previsões precisas devido à complexidade das variáveis envolvidas.

Outro ponto, vamos considerar uma startup no ramo imobiliário que está coletando dados do mercado para prever preços de aluguel. A precisão dessa previsão depende diretamente da quantidade e da qualidade dos dados coletados. Se a base de dados contém informações limitadas, como apenas 50 propriedades de tipos variados, pode ser desafiador extrair padrões significativos para estabelecer preços de aluguel precisos.

Agora você percebe a importância de ter não apenas uma variedade adequada de características (features) para análise, mas também um volume suficiente de dados.

Além disso, a complexidade do problema a ser resolvido influencia diretamente a quantidade e o tipo de dados necessários. Prever o vencedor de um jogo de futebol pode ser relativamente simples em comparação com a previsão dos preços de ações na bolsa de valores, por exemplo. Cada problema tem sua própria complexidade, o que determina a necessidade de diferentes quantidades e tipos de dados.

Conceitos básicos



Machine learning

Nos dias de hoje é muito comum ouvirmos os termos “Inteligência Artificial” e “Machine Learning” nas redes sociais ou em pitchs de vendas de CEOs de empresas de tecnologia, tanto que hoje ela pode ser considerada como uma das “buzzwords” mais comuns dentro do ambiente corporativo, mas o que de fato é um aprendizado de máquina?

Bom, o aprendizado de máquina é um subcampo de estudo da área de inteligência artificial que consiste no desenvolvimento e aplicação de algoritmos que **são capazes de identificar padrões em dados para realizar previsões com base nestes padrões.**

O processo de desenvolvimento de um algoritmo de aprendizado de máquina costuma acontecer através de uma fase de treinamento, onde um conjunto de dados é apresentado ao algoritmo e ele abstrai os conceitos

sobre estes dados, para que então ele possa realizar inferências (ou previsões) sobre dados que ele nunca viu antes.

Esse processo de compreender a estrutura de um modelo é fundamental. Pense nisso como usar o passado para moldar expectativas futuras. Por exemplo, suponha que você queira estimar o aumento na procura de produtos em sua loja após um investimento de 10 mil reais em publicidade. Com dados históricos de quanto você já investiu e quanto teve de retorno, podemos entender o padrão matemático que te dirá “se você investe X, seu retorno será Y”

Mas é importante tomar certos cuidados aqui. Se no histórico da empresa, o investimento sempre foi de 50 mil reais. Sem variações anteriores para comparar, fica difícil prever o impacto desse aumento no investimento usando os métodos convencionais.

Como um modelo é estruturado?

Cada algoritmo tem sua maneira de operar, e nem todos seguem um padrão linear como o exemplo que falamos das notas dos alunos na prova. Contudo, algo que todos compartilham, especialmente aqueles que se baseiam em previsões, é a função de perda (loss function).

Note que a linha gerada pelo modelo de Regressão Linear não se ajusta perfeitamente a todos os pontos de dados. Isso ocorre porque é inviável encontrar uma fórmula que capture perfeitamente a relação entre notas de prova e horas de estudo, e isso se aplica a qualquer modelo ou relação entre variáveis e resultados. Sempre haverá discrepâncias, como dados atípicos ou situações que desafiam a norma.

Então, qual é a finalidade de um modelo que visa previsões? Naturalmente, é fazer previsões **o mais próximas possível da realidade.**

Dito de outra maneira, um modelo busca identificar padrões que aproximem o resultado real, com o objetivo de cometer o menor erro possível. De forma mais técnica, isso significa minimizar a função de perda - minimizar a loss function.

O que são funções de perda?

Funções de perda são ferramentas matemáticas que quantificam a diferença entre as previsões do modelo e os valores reais dos dados. Pode parecer simples calcular essa diferença, mas há mais nuance envolvida do que simplesmente subtrair um valor do outro.

Considere a construção de um modelo para prever nota em provas com base nas horas estudadas, onde o processo de treinamento envolve explorar diversas possibilidades para determinar a equação mais eficaz. Durante esse processo, o modelo avalia várias retas potenciais para encontrar a ideal, mostrando a complexidade envolvida na escolha de um modelo adequado.

A escolha da loss function depende do tipo de problema que está sendo abordado. Existem diferentes tipos de loss functions, como a função de erro quadrático médio (mean squared error, MSE) para problemas de regressão, a função de entropia cruzada (cross-entropy) para problemas de classificação binária ou multiclasse, entre outras. Tudo isso vamos abordar bem mais pra frente, ok?

Por hora, saiba que objetivo é minimizar a loss function durante o treinamento, ajustando os parâmetros do modelo de maneira a tornar as previsões cada vez mais precisas. No fundo, é quase como se estivéssemos realmente fazendo a comparação do que aconteceu com o que o modelo previu, só que a gente faz de maneira inteligente.

Um pouco sobre história

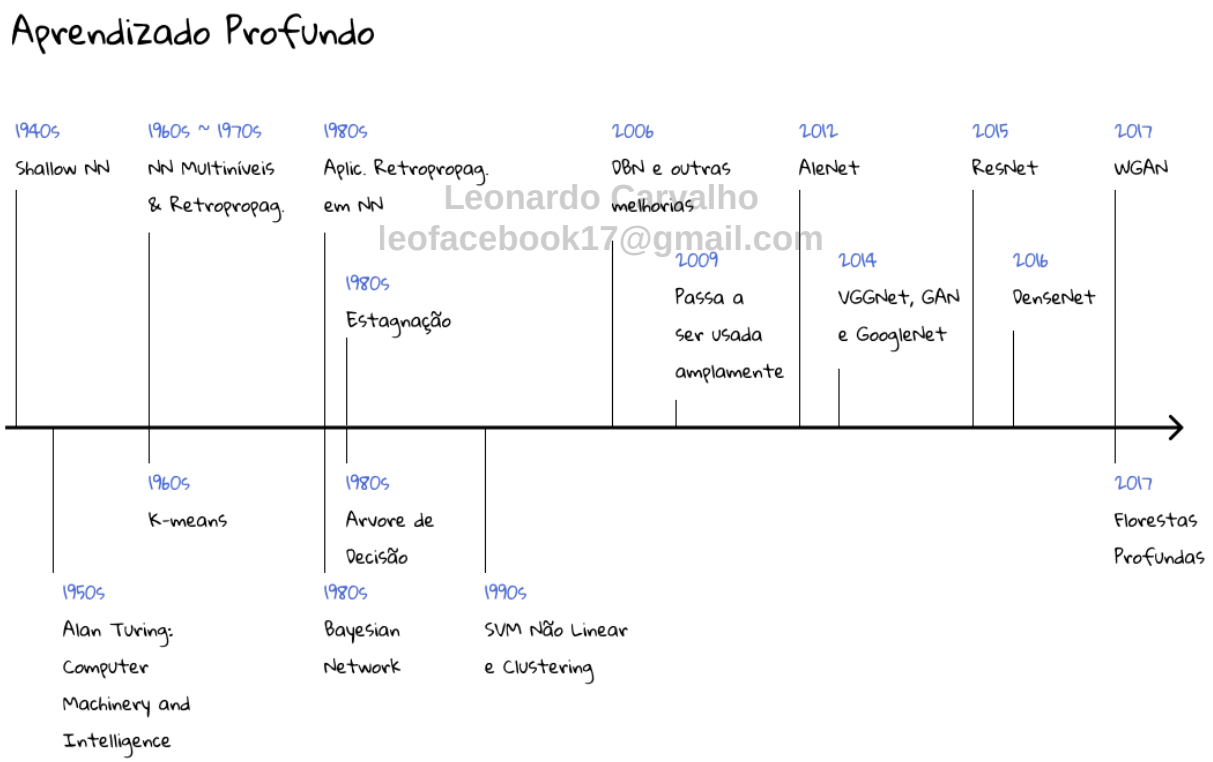
Diferente do que a maioria das pessoas pode pensar, este campo não é novo dentro da ciência da computação, os primeiros artigos acadêmicos neste campo datam de meados dos anos 40, praticamente ao mesmo tempo em que surgiam os primeiros computadores de uso geral.

O aprendizado de máquina foi concebido pela primeira vez a partir da modelagem matemática de redes neurais em um artigo do lógico Walter Pitts e do neurocientista Warren McCulloch, publicado em 1943, que tentou

mapear matematicamente os processos de pensamento e a tomada de decisão na cognição humana.

Em 1959, Arthur Samuel, pioneiro no campo, definiu inteligência artificial como o “campo de estudo que dá aos computadores a habilidade de aprender sem serem explicitamente programados”.

Grande parte dos algoritmos que são amplamente usados hoje foram concebidos na academia há muitos anos atrás, mas porque então eles demoraram tanto para chegar ao mercado? A resposta está na capacidade de processamento das máquinas disponíveis de forma acessível e no volume de dados gerados atualmente.



Linha do Tempo com os principais acontecimentos no campo do Aprendizado de Máquina

Até meados dos anos 2010, não era comum que todos tivessem acesso a computadores e celulares com alta capacidade de processamento. Com a

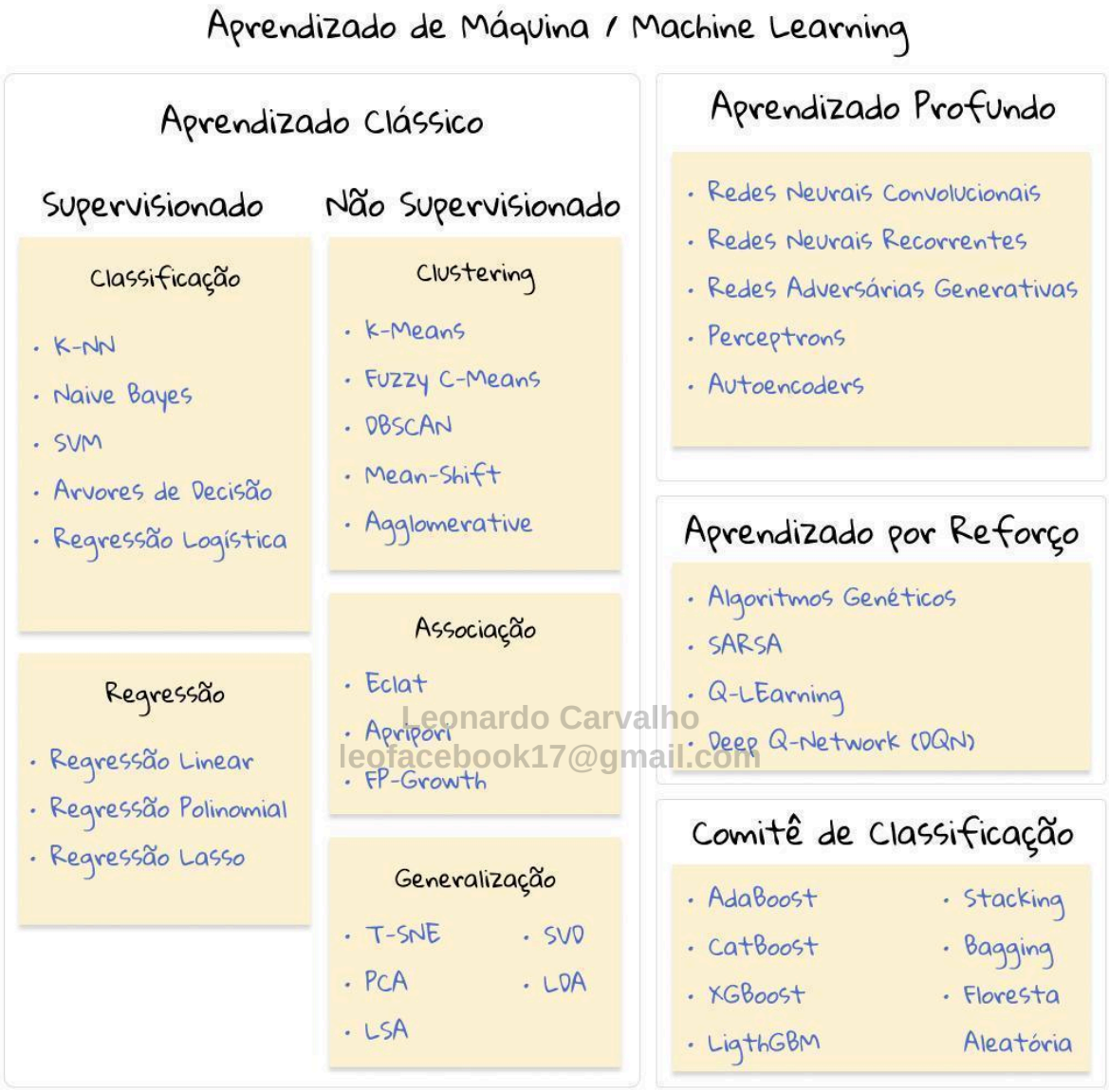
revolução dos smartphones - que são verdadeiros computadores portáteis - as empresas de tecnologia finalmente puderam ter acessos a dados abundantes sobre os usuários que poderiam ser usados para alimentar estes algoritmos e então explorá-los de forma comercial.

E como o poder computacional e o volume de dados coletados cresce exponencialmente a cada ano, o custo para colocar este tipo de tecnologia em produção acaba reduzindo e a demanda por novos modelos cresce. Com isso, novos modelos são desenvolvidos por cientistas de dados para resolver ou melhorar a abordagem de solução dos mais diversos problemas, como o processamento de linguagem natural, detecção e rastreio de objetos em imagens e vídeos, detecção de fraudes, recomendação de produtos e serviços, refinamento de mecanismos de buscas, entre outros.

Nos próximos capítulos vamos falar mais sobre os principais grupos de algoritmos aplicados amplamente no mercado e suas propriedades:

- Algoritmos de Aprendizado Profundo (Deep Learning)
- Algoritmos de Aprendizado Supervisionado
- Algoritmos de Aprendizado Não Supervisionado

Para facilitar a compreensão, dividi os principais grupos nesta ilustração abaixo:



Ao longo deste módulo, nós vamos abordar alguns dos principais algoritmos dentro do aprendizado clássico, fique tranquilo.

Como funciona o dia a dia de quem constrói modelos?

A construção de um modelo analítico no cenário real exige uma abordagem estratégica e bem fundamentada.

1. Identificação do problema: Esta etapa é crucial, mas frequentemente subestimada ou mal compreendida. O primeiro passo é ter uma

compreensão cristalina do que se pretende alcançar com o modelo. Por exemplo, se a tarefa for estabelecer um preço que otimize os lucros, o foco será puramente preditivo, priorizando a precisão das previsões sobre a facilidade de interpretação do modelo. Em contraste, na modelagem de riscos sujeita a regulamentações específicas, pode ser necessário optar por modelos com explicabilidade, onde a contribuição de cada variável (coeficientes) é transparente. Essa diferenciação é vital porque a preparação das variáveis e a compreensão do problema vão além da mera coleta de dados; é necessário entender as nuances, como a janela de tempo relevante para a previsão.

Pensemos em um cenário onde o objetivo seja avaliar a eficácia de uma campanha de marketing digital - saber quanto vamos vender a partir de uma certa campanha. Simplesmente decidir modelar essa eficácia não é suficiente; é crucial especificar se o foco está na análise imediata pós-campanha ou no impacto a longo prazo, digamos, em 15 ou 60 dias. Esta decisão afeta significativamente a maneira como os dados serão coletados e interpretados, pois a definição do período influencia diretamente os indicadores de sucesso a serem analisados.

Vamos também considerar a tarefa de prever o sucesso de um novo aplicativo móvel no mercado. Sem um planejamento cuidadoso, pode-se iniciar a coleta de dados de uso e feedback dos usuários nos primeiros meses após o lançamento. Contudo, se o intuito do modelo for antecipar quais funcionalidades serão mais apreciadas pelos usuários para priorizar atualizações futuras, informações como a frequência de uso de determinadas funcionalidades ou o feedback inicial podem não ser suficientes ou adequadas. Isso enfatiza a necessidade de um entendimento profundo do propósito do modelo e de como ele será empregado, assegurando que as variáveis selecionadas sejam relevantes e disponíveis para análise no momento necessário.

2. Coleta e preparação dos dados: Essencial para a construção de qualquer modelo analítico, essa fase envolve a aquisição e o processamento dos dados, incluindo a limpeza, o tratamento de dados faltantes, a normalização e a codificação de variáveis categóricas. A adequação dos dados é crítica, principalmente em modelos que dependem da distância entre as observações para fazer previsões. Por exemplo, a diferença na escala das

variáveis pode distorcer a importância relativa dos atributos, um desafio que precisa ser abordado durante a preparação dos dados. Porém, vamos falar disso mais pra frente

3. Validação do modelo: Imagine que você possui dados coletados até julho e está desenvolvendo um modelo na primeira semana de agosto. Se todos os dados forem usados para treinamento, como validar a eficácia do modelo? A solução é reservar uma parte dos dados, chamada conjunto de teste, para testar o modelo. Esse conjunto simula novas informações que o modelo encontrará quando estiver operacional, garantindo que o modelo seja avaliado em condições que mimetizam seu uso futuro. Essa prática assegura que o modelo seja robusto e confiável, evitando surpresas quando aplicado a dados reais.
Também vamos ver mais pra frente algumas técnicas para fazer validação de modelos (e split de dados!).

4. Seleção de um método de modelagem: Após prepararmos os dados, o próximo passo é escolher um método de modelagem adequado. Por exemplo, no começo, introduzimos a regressão linear como um método que tenta encontrar a linha mais adequada para representar a relação entre as variáveis. Há uma variedade de outros métodos que já exploramos, incluindo árvores de decisão e o algoritmo k-means. Vale a pena revisitar esses conceitos. Isso será visto no próximo material!

5. Treino do modelo: Com o método escolhido, prosseguimos para o treinamento do modelo. Esse processo envolve ajustar o modelo para que ele encontre os melhores parâmetros que reduzam o erro de previsão, similares aos coeficientes na equação que discutimos anteriormente no contexto de um exemplo de recursos humanos.

6. Ajuste e validação do modelo: O próximo passo é validar o desempenho do modelo usando um conjunto de dados de validação. Se for necessário, ajustamos os hiperparâmetros, como a taxa de aprendizado ou a quantidade de camadas, para melhorar o desempenho do modelo.

7. Avaliação final do modelo: Com o modelo ajustado, é importante avaliar sua eficácia utilizando um conjunto de testes. Dependendo do problema em

questão, diferentes métricas, como precisão, recall, pontuação F1 ou erro quadrático médio, podem ser aplicadas.

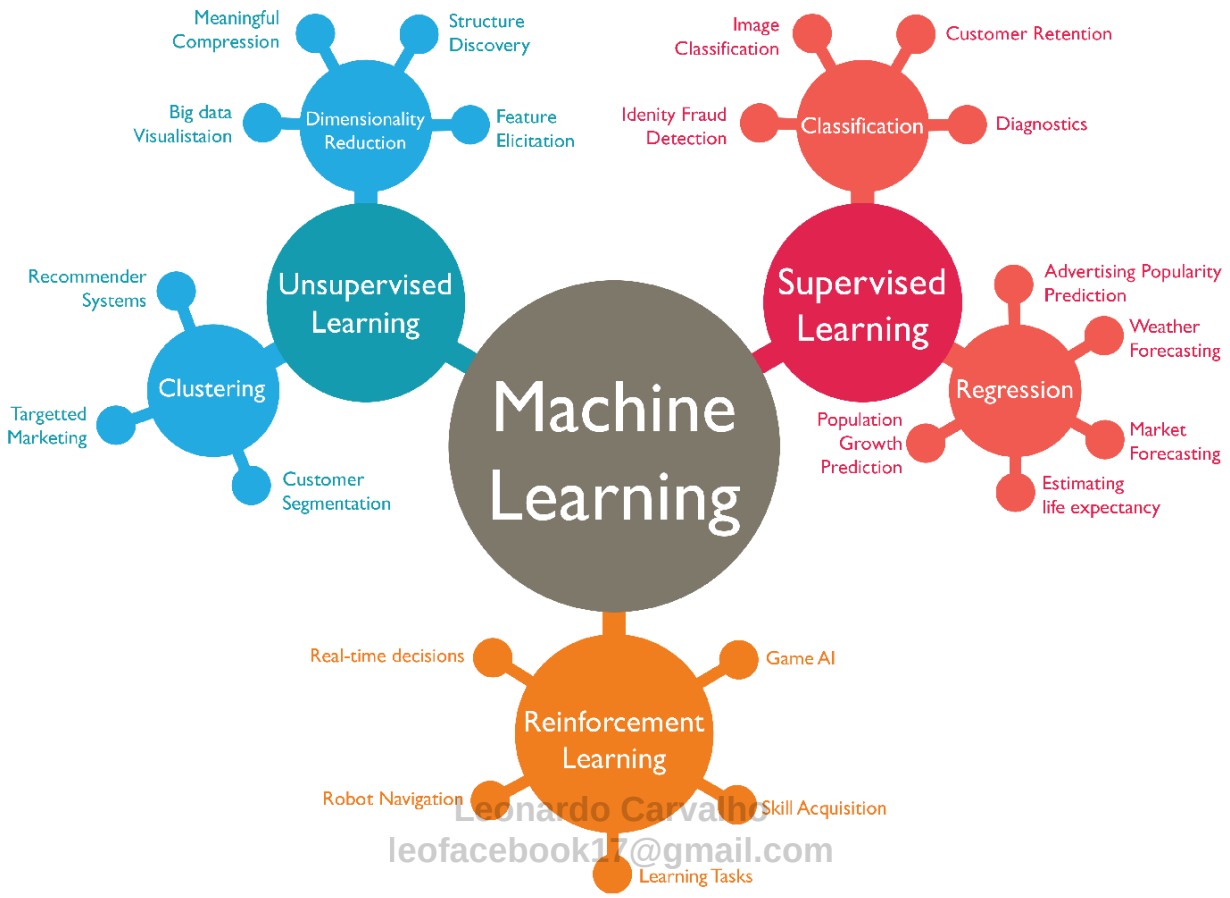
8. Implementação e acompanhamento: O último passo é colocar o modelo em operação no ambiente de produção e monitorar seu desempenho continuamente. É crucial reavaliar e ajustar o modelo regularmente à medida que novos dados são coletados.

Agora vamos entrar em pontos mais específicos de machine learning.

Diferença entre aprendizado supervisionado e não supervisionado

Em termos práticos, hoje podemos entender a área da inteligência artificial prática para o mercado de trabalho em 3 grandes grupos:

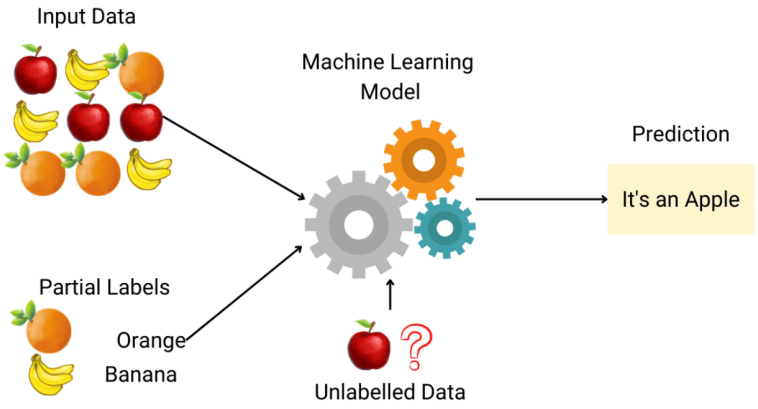
- Aprendizado Supervisionado: Consiste em utilizar algoritmos de aprendizado de máquina clássicos e dados já classificados por humanos para identificar padrões e realizar previsões.
- Aprendizado Não Supervisionado: Semelhante ao aprendizado supervisionado, porém são fornecidos dados não classificados e os padrões são identificados pelo próprio algoritmo.
- Reinforcement Learning: O aprendizado por reforço é um tipo de aprendizado de máquina em que um agente aprende a tomar decisões por meio de tentativa e erro, interagindo com um ambiente. O agente é recompensado por ações que se aproximam de um objetivo desejado e penalizado por aquelas que não o fazem, permitindo-lhe aprender estratégias ótimas ao longo do tempo. Essa abordagem é amplamente utilizada em situações que requerem uma sequência de decisões, como jogos, navegação de robôs, e otimização de sistemas, onde o agente deve explorar o ambiente e aprender com as consequências de suas ações para maximizar alguma noção de recompensa cumulativa.



Costumamos dizer que grande parte dos seus problemas - e o que é pedido em processos seletivos hoje em dia - você vai resolver com supervisionado e não supervisionado. Portanto, vamos a eles!

Aprendizado Supervisionado

Os algoritmos deste grupo tratam dados que são etiquetados - ou classificados - previamente, ou seja, existe um pessoa ou agente que cumpre o papel de professor ou **supervisor** deste algoritmo através da marcação dos dados de um conjunto de treino.



Existem dois tipos de tarefas que estes algoritmos realizam:

- **Classificação:** Ele determina a classe de um objeto ou registro.
- **Regressão:** Ele prevê um valor com base em uma relação entre dois valores em uma escala numérica.

No próximo capítulo nós vamos entrar em mais detalhes sobre estes algoritmos, onde vou dar mais alguns exemplos e explicar sobre suas características.

Em um exemplo de um algoritmo de classificação:
Um conjunto de dados de treino deste grupo costuma possuir as features (ou campos) que são as características deste dado (podem ser a cor da pelagem, formato das orelhas, formato do focinho, tamanho) e a classificação prévia destes registros (gato, cachorro ou porco).

Ao realizar uma inferência em dados futuros, o algoritmo interpreta a relação entre as features e as classes previamente treinadas para então estimar se ele está diante de um gato, cachorro ou porco.

É claro que na vida real os algoritmos possuem muito mais features e suas classes podem ser extremamente complexas, mas a abordagem neste grupo de algoritmo é sempre a mesma.

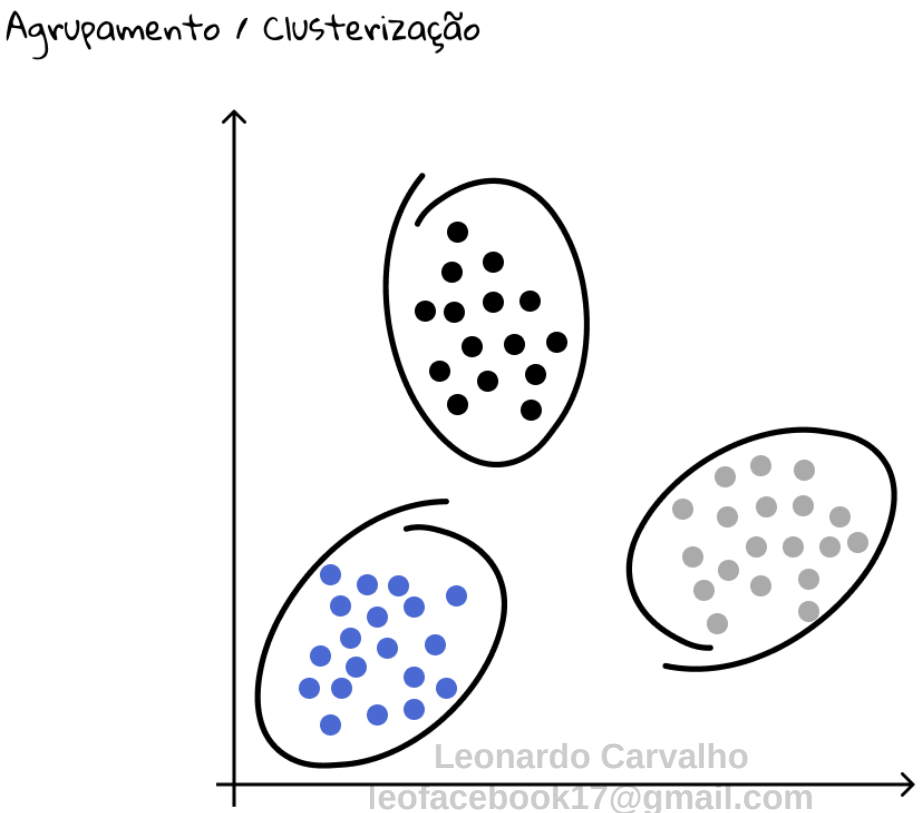
Aprendizado Não Supervisionado

Na contramão dos algoritmos supervisionados, esta classe de algoritmos tenta descobrir por si própria os padrões em um determinado conjunto de dados.

Neste caso, o conjunto não foi classificado ou etiquetado, muitas vezes por não ser uma possibilidade, dado um contexto de negócio. Seja por conta do grande volume de dados, ou por conta do custo de classificação manual.

Existem três tipos de tarefas que estes algoritmos realizam:

- **Agrupamento / Clustering:** Trata-se da técnica de dividir o conjunto de dados em grupos ou classes com base em features desconhecidas.
- **Associação:** Ele cria regras que encontram associações entre pontos de dados. Então ele encontra as relações entre as variáveis, identificando itens que tendem a ocorrer juntos.
- **Generalização / Dimensionality Reduction:** Ele tenta interpretar conjunto de dados gigantescos para reduzir o número de features (ou dimensões) deste conjunto para melhor compreensão e visualização destes dados.



Clustering consiste em classificar registros próximos uns dos outros em grupos.

Eles são comumente usados em aplicações como:

- Segmentação de mercado (identificar tipos de clientes / usuários)
- Analisar e classificar dados novos
- Detecção de anomalias no conjunto de dados
- Sistemas de recomendação de compras
- Análise de imagens falsas

Iremos explorar melhor estes algoritmos nas próximas semanas do curso, logo após explorarmos o grupo de algoritmos supervisionados.

Supervisionado: Diferença entre regressão e classificação

Vamos começar a aprofundar o entendimento dos tipos de algoritmos do grupo de aprendizado supervisionado: Classificação e Regressão.

Classificação

Como vimos brevemente no anteriormente, os algoritmos de classificação utilizam alguns atributos, também chamados de dimensões ou features, para classificar um objeto ou registro. Por exemplo, ele pode separar frutas baseadas por cor, formato e valor nutricional; músicas baseadas no gênero; raças de cachorro baseadas em tamanho, tipo de pelo, cor, formato das orelhas e focinho, e assim por diante.

Estes tipos de algoritmos são utilizados na maior parte das aplicações, visto que a maior parte do valor econômico agregado por inteligências artificiais no mundo hoje são concentradas em atividades que relacionam uma entrada A com uma saída B.

Alguns dos algoritmos mais usados são:

- K Vizinhos mais Próximos / K-Nearest Neighbor (KNN)
- Máquina de vetores de suporte / Support Vector Machines
- Árvore de Decisão / Decision Tree
- Regressão Logística / Logistic Regression

Vamos à alguns exemplos de aplicações reais na qual este grupo de algoritmos é muito bom:

- **Gmail:** Entrada: Contém determinadas palavras → Saída: É Spam
- **Bancos:** Entrada: Pedido de Empréstimo → Saída: Será pago?
- **Google Ads:** Entrada: Usuário de um site → Saída: Clicará no anúncio?
- **Alexa:** Entrada: Áudio → Saída: Transcrição em texto
- **Tesla:** Entrada: Radar / Câmera → Saída: Posição do outro carro

Em todos estes casos, o algoritmo sempre precisará de um supervisor, em outras palavras, o conjunto de dados deve ser marcado com as categorias e os atributos para que a máquina possa fazer uma previsão em um novo registro com base neste conjunto de treino.

Regressão

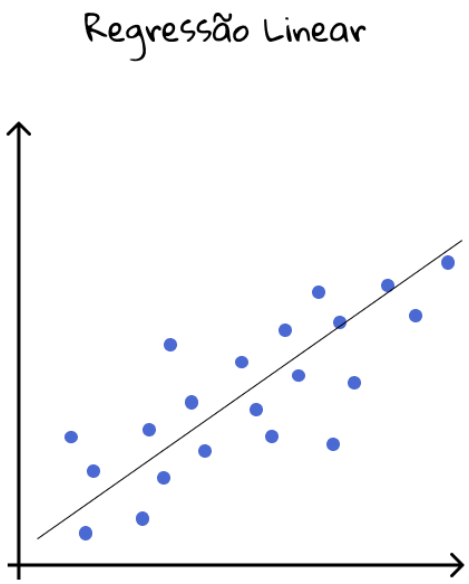
Este grupo de algoritmos permite que consigamos estimar um determinado valor numérico em uma escala Y dado um valor X. Neste grupo, os valores de X e Y são sempre numéricos e contínuos, isto é, não existem lacunas dentro destas faixas de valores.

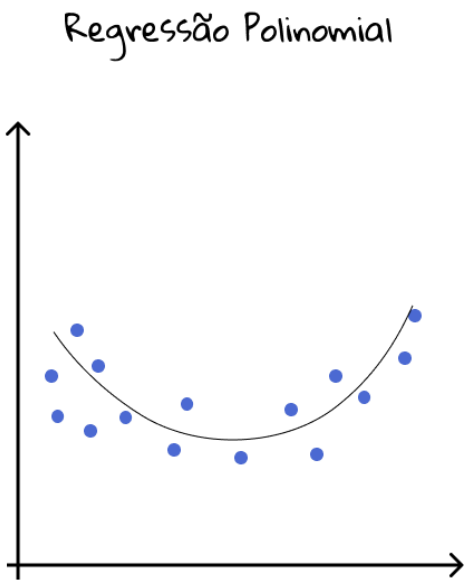
Por exemplo de escala contínua: Altura e peso são valores que não possuem limites dentro de sua escala, sempre existirá um valor intermediário entre dois pontos específicos desta escala: 10 kg, 10.1 kg, 10.11 kg, 10.111 kg.

Um exemplo de escala numérica de valores não contínuos é o número de crianças que alguém pode ter: Você pode ter 1 ou 2 crianças, mas nunca 1.2 ou 1.5 crianças, existe uma lacuna entre números inteiros que não permite continuidade deste conjunto.

Vamos à alguns exemplos de aplicações reais na qual este grupo de algoritmos é muito bom:

- Estimar preços de casas
- Nível de Engarrafamento
- Estimar preços de ações





Uma regressão sempre poderá ser visualizada em um plano cartesiano de duas dimensões (X e Y) devido à sua natureza. Na figura abaixo, cada ponto é um registro no conjunto de dados e a linha representa a regressão neste conjunto. O tipo de regressão é definido pelo formato da curva: Se ela for reta, é uma Regressão Linear, se ela for curva, então é uma Regressão Polinomial.

Material complementar

O que é Machine Learning e como funciona

What is Machine Learning?

Qual a diferença entre Inteligência Artificial, Machine Learning, Data Science,

Deep Learning, etc?

Supervised vs. Unsupervised Learning

Tratando dados

Conforme vimos anteriormente, os algoritmos de aprendizado de máquina utilizam atributos dos dados de treino para compreender os padrões e realizar inferências em dados futuros. Acontece que geralmente existe uma grande diferença entre o formato dos dados que são disponibilizados pelos sistemas de coleta e o formato aceito pelo algoritmo. E para modelar estes dados de coleta no formato do algoritmo, realizamos várias etapas de transformação a fim de gerar os atributos (ou features) necessários para treinar nosso modelo, este processo é chamado de Feature Engineering.

Existem algumas etapas extremamente comuns neste processo, e todas elas convergem para o objetivo de simplificar o dado para o treinamento do algoritmo ou para aumentar a acurácia do modelo. Algumas destas etapas já foram apresentadas lá em estatística, mas vamos a alguns resumos e, posteriormente, a explicação detalhada de cada um

Análise Exploratória dos Dados: Também conhecida como Exploratory Data Analysis (EDA), é uma técnica poderosa e simples que pode ser usada para melhorar sua compreensão de seus dados, explorando suas propriedades. A técnica é frequentemente aplicada quando o objetivo é criar novas hipóteses ou encontrar padrões (como correlação) nos dados. Se você ainda tem dúvidas sobre EDA, volte algumas casas e vá para o módulo de estatística!

Criação de Features: É muito comum que o atributo que você quer utilizar não esteja disponível nos dados coletados, então é necessário que você os crie a partir dos dados disponíveis. Exemplos de features que podem ser criadas: Idade dos Usuários de uma plataforma quando temos apenas a data de início dele, Frequência de acessos em um site quando temos em várias linhas separadas todos os registros que ele entrou, etc.

Tratamento de Nulos: Quando se trata de preparar seus dados para aprendizado de máquina, os valores ausentes são um dos problemas mais comuns. Erros humanos, interrupções no fluxo de dados, dados anonimizados e outros fatores podem contribuir para a ocorrência de registros vazios. Estas falhas podem gerar impacto no seu modelo

Tratamento de Outliers: Outro problema muito comum são os registros numéricos que estão muito distantes do restante dos registros, os chamados Outliers. Em alguns casos será necessário removê-los, e em outros será necessário substituí-los ou tratá-los. Em muitos casos, não devemos mexer nisso.

Normalização ou Escalonamento: O escalonamento de features é um dos problemas mais difundidos e difíceis no aprendizado de máquina, mas é uma das coisas mais importantes para acertar. Isto ocorre porque para treinar alguns tipos de modelo (como os que são baseados em distância), precisamos que os dados de diversas features estejam na mesma escala para evitar que uma feature tenha mais peso do que outra na decisão do modelo.

Tratamento de Valores Categóricos: Um dos métodos mais comuns para atributos categóricos é transformar cada atributo em uma representação numérica. A transformação de dados categóricos em dados numéricos geralmente é chamada de “categorical-column encoding” e cria um novo atributo numérico que é tratável para muitos algoritmos de aprendizado de máquina. A codificação permite que atributos categóricos sejam incorporados de forma rápida e flexível em um modelo. O objetivo de criar uma representação numérica é capturar e conservar o relacionamento entre o atributo categórico e a classe de destino.

Criando Variáveis (Feature Engineering)

Bom, vamos começar falando sobre a criação de novas features, ou novas variáveis, a depender da literatura. É muito comum que os dados que você recebeu não estejam agrupados da forma correta, ou então você precise criar um índice ou taxa que representa algum fato da realidade, mas que não está no seu conjunto original. Em outros casos, você pode entender que precisa criar novas features porque você percebeu alguma distribuição, colinearidade ou então um relacionamento complicado indesejado durante sua análise exploratória. Ainda, mais comumente, existe a possibilidade de você querer features que potencializem o poder preditivo do seu modelo, ou seja, que melhore a resposta do seu modelo.

Vou exemplificar aqui algumas das abordagens mais comuns na hora de criar novas features utilizando a biblioteca Pandas do Python. O conjunto de dados utilizado neste exercício está disponível para download e você poderá repetir os passos junto comigo no seu Jupyter Notebook!

Transformações Matemáticas

A criação de novos atributos a partir de operações com outros atributos numéricos podem ser representados como fórmulas matemáticas no Pandas, como se os atributos fossem números simples.

No nosso conjunto de dados de exemplo, nós temos uma coluna com a Distância em Milhas, mas digamos que por algum motivo nós precisaremos dela em quilômetros para montar algum índice, neste caso nós podemos realizar o cálculo da seguinte maneira:

```
df['Distancia(km)'] = df['Distancia(mi)'] * 1.60934
df.head()
```

	Endereco	Distancia(mi)	Descrição	Distancia(km)
0	Outerbelt E | Dublin, OH, US	3.230	Between Sawmill Rd/Exit 20 and OH-315/Olentang...	5.198168
1	I-70 E | Dayton, OH, US	0.747	At OH-4/OH-235/Exit 41 - Accident.	1.202177
2	I-75 S | Cincinnati, OH, US	0.055	At I-71/US-50/Exit 1 - Accident.	0.088514
3	I-77 N | Akron, OH, US	0.123	At Dart Ave/Exit 21 - Accident.	0.197949
4	I-75 S | Cincinnati, OH, US	0.500	At Mitchell Ave/Exit 6 - Accident.	0.804670

Note que a simples conversão de milhas para quilometragem não teria impacto no nosso modelo simplesmente, mas este exemplo serve para entender que podemos criar features realizando cálculos matemáticos simples entre as features já existentes.

Concatenar ou Dividir Features

Digamos que você possui um atributo categórico no qual você precisa remover parte do texto, ou então extrair algum prefixo/sufixo ou até mesmo combinar mais de um atributo para gerar uma feature que faça mais sentido no seu modelo.

Existem algumas operações no Pandas que podem ser feitas para realizar estas transformações, vamos entender melhor algumas delas.

No nosso conjunto de exemplo, existe uma coluna chamada Endereço, que consiste basicamente no compilado textos no formato (Rua | Cidade, Estado, País), e digamos que no nosso modelo nós precisaremos dos registros por cidade apenas, então vamos removê-la do bloco de texto onde ela está inserida:

```
# Quebrando o Texto de Endereço em uma lista à partir do caracter '|'
# Neste momento temos um lista com dois elementos: [Rua, Cidade com Estado e
País]
df['ListaEndereco'] = df['Endereco'].str.split('|')

# Agora vamos pegar o segundo elemento da lista, e quebrar pelo caracter ','
# Uma nova lista é gerada, sendo o primeiro elemento a cidade
df['Cidade'] = df['ListaEndereco'].str[1].str.split(', ').str[0]
df.head()
```

	Endereco	Distancia(mi)	Descrição	Distancia(km)	ListaEndereco	Cidade
0	Outerbelt E | Dublin, OH, US	3.230	Between Sawmill Rd/Exit 20 and OH-315/Olentang...	5.198168	[Outerbelt E , Dublin, OH, US]	Dublin
1	I-70 E | Dayton, OH, US	0.747	At OH-4/OH-235/Exit 41 - Accident.	1.202177	[I-70 E , Dayton, OH, US]	Dayton
2	I-75 S | Cincinnati, OH, US	0.055	At I-71/US-50/Exit 1 - Accident.	0.088514	[I-75 S , Cincinnati, OH, US]	Cincinnati
3	I-77 N | Akron, OH, US	0.123	At Dart Ave/Exit 21 - Accident.	0.197949	[I-77 N , Akron, OH, US]	Akron
4	I-75 S | Cincinnati, OH, US	0.500	At Mitchell Ave/Exit 6 - Accident.	0.804670	[I-75 S , Cincinnati, OH, US]	Cincinnati

Para fins ditáticos, podemos concatenar o estado a país usando este procedimento:

```
# Vamos pegar o segundo elemento da lista de endereço (Cidade com Estado e País),
e quebrar pelo caracter ', '
# Teremos uma nova lista, agora com cada elemento representando a cidade, o
estado e o país
df['ListaEndereco'] = df['ListaEndereco'].str[1].str.split(', ')

# Vamos atribuir cada elemento à uma coluna, para fins ditáticos
df['Estado'] = df['ListaEndereco'].str[1]
df['Pais'] = df['ListaEndereco'].str[2]

# Opção 1: Depois podemos concatenar as duas colunas da seguinte maneira
df['EstadoPais'] = df['Estado'] + ', ' + df['Pais']

# Opção 2: Outra forma de fazer a concatenação é usando a lista diretamente
df['EstadoPais'] = df['ListaEndereco'].str[1:].str.join(', ')
df.head()
```


	Endereco	Distancia(mi)	Descrição	Distancia(km)	ListaEndereco	Cidade	Estado	Pais	EstadoPais
0	Outerbelt E | Dublin, OH, US	3.230	Between Sawmill Rd/Exit 20 and OH-315/Olentang...	5.198168	[Dublin, OH, US]	Dublin	OH	US	OH, US
1	I-70 E | Dayton, OH, US	0.747	At OH-4/OH-235/Exit 41 - Accident.	1.202177	[Dayton, OH, US]	Dayton	OH	US	OH, US
2	I-75 S | Cincinnati, OH, US	0.055	At I-71/US-50/Exit 1 - Accident.	0.088514	[Cincinnati, OH, US]	Cincinnati	OH	US	OH, US
3	I-77 N | Akron, OH, US	0.123	At Dart Ave/Exit 21 - Accident.	0.197949	[Akron, OH, US]	Akron	OH	US	OH, US
4	I-75 S | Cincinnati, OH, US	0.500	At Mitchell Ave/Exit 6 - Accident.	0.804670	[Cincinnati, OH, US]	Cincinnati	OH	US	OH, US

Agrupamento de Registros

Certo, até aqui já fizemos alguns atributos novos no nosso conjunto de dados, mas digamos que meu modelo quer classificar as cidades, e eu já tenho os labels em outro conjunto de dados agrupado por cidade, então eu preciso das seguintes features: Contagem e incidentes e distância média para cada

cidade. Para isso, precisaremos utilizar o método groupby do Pandas, que tem um comportamento semelhante ao SQL.

```
# Agrupando dados por cidade e agregando a distância pela média e contando o
total de registros através da coluna 'Descrição'
df = df.groupby('Cidade').agg({'Distancia(km)': 'mean', 'Descrição':
'count'}).reset_index()

#Renomeando a coluna 'Descrição' para 'Total de Registros', para fazer mais
sentido no contexto
df.rename(columns={'Descrição': 'Total de Registros'}, inplace=True)
```

	Cidade	Distancia(km)	Total de Registros
0	Aaronsburg	0.478779	2
1	Abbeville	0.325930	21
2	Abbotsford	1.536920	4
3	Abbottstown	1.203071	9
4	Aberdeen	2.986087	93

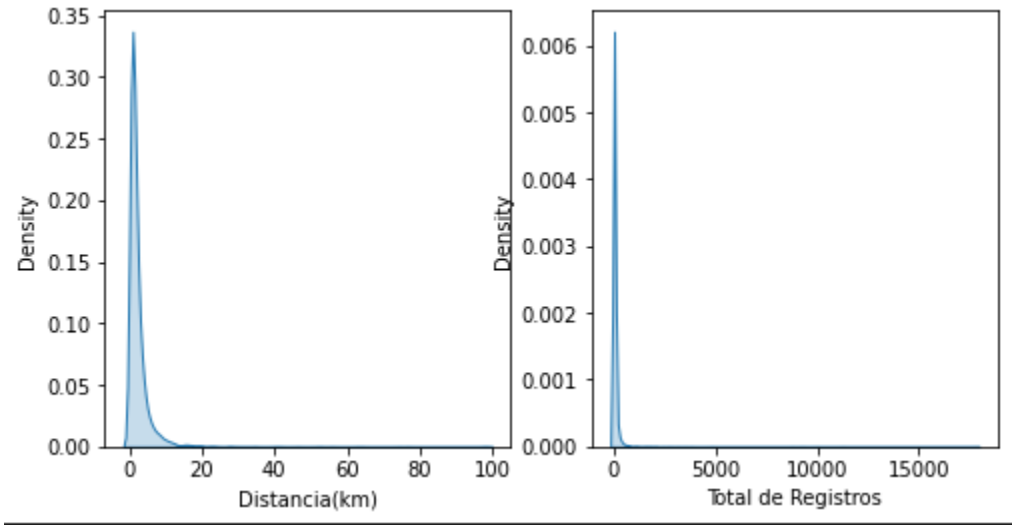
Transformação de Log

Em alguns casos, os dados precisam obedecer a uma distribuição normal para ser interpretado corretamente pelo algoritmo, mas este dado não está distribuído corretamente, ele apresenta algumas distorções. Nestes casos transformamos o algarismo em seu logaritmo natural para normalizar a distribuição destes dados.

Geralmente você nota que existe uma normalização para ser realizada através da visualização dos dados, como no exemplo abaixo, onde podemos perceber que as métricas não obedecem a uma distribuição normal.

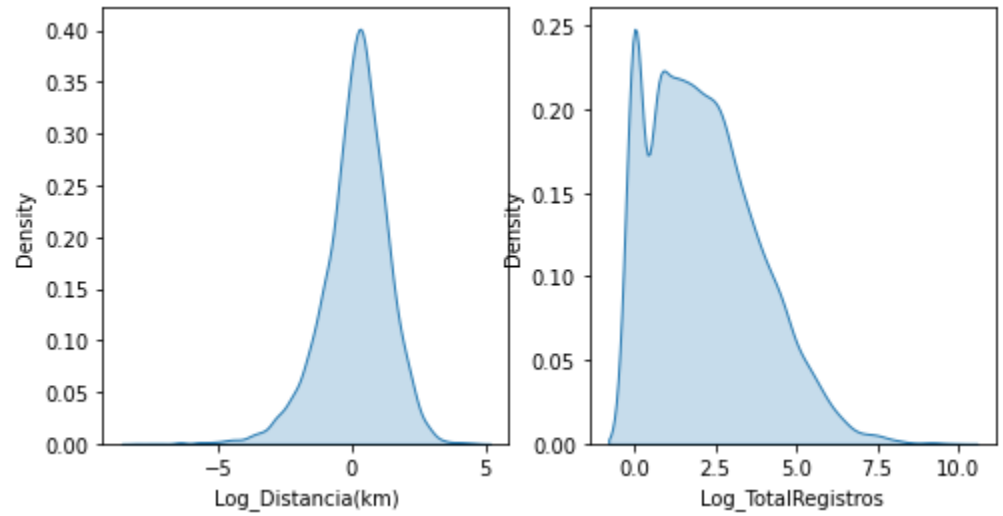
```
# exibição das métricas de distância média e total de registros
fig, axs = plt.subplots(1, 2, figsize=(8, 4))
sns.kdeplot(df['Distancia(km)'], shade=True, label='Distância Média', ax=axs[0])
```

```
sns.kdeplot(df['Total de Registros'], shade=True, label='Total de Registros',
ax=axs[1])
```




```
# substituição dos valores por seus logaritmos naturais
df['Log_Distancia(km)'] = np.log(df['Distancia(km)'])
df['Log_TotalRegistros'] = np.log(df['Total de Registros'])

fig, axs = plt.subplots(1, 2, figsize=(8, 4))
sns.kdeplot(df['Log_Distancia(km)'], shade=True, label='Distância Média',
ax=axs[0])
sns.kdeplot(df['Log_TotalRegistros'], shade=True, label='Total de Registros',
ax=axs[1])
```



Note que aproximamos as métricas a uma distribuição normal, em nem todos os casos este tipo de transformação será perfeita.

Devemos cuidar com este tipo de transformação, pois é recomendado utilizá-la apenas em alguns contextos e algoritmos que realmente necessitam de distribuição normal, pois este tipo de prática pode remover algumas correlações entre as features do modelo se não for aplicada corretamente.

Então, em Resumo:

Atributos podem ser criados de várias maneiras e de acordo com a necessidade que seu modelo e do seu contexto de negócio. Note que toda feature fictícia é criada como uma nova coluna do DataFrame, e até aqui isto não é um problema, porque no final do seu processo de feature engineering será necessário que você selecione apenas as features desejadas para seu modelo.

Até aqui, nós compilamos todas as features que potencialmente serão utilizadas pelo modelo de machine learning, sejam elas originais do nosso conjunto de dados, ou fictícias criadas durante o processo de Feature Engineering. Acontece que estas features ainda são estão configuradas da melhor maneira para a interpretação do modelo, então nos próximos capítulos vamos tratar destas transformações em features numéricas e categóricas.

Tratando Valores Numéricos

Apesar de já entendermos como criar novas features à partir de dados numéricos e até realizar algumas transformações, é necessário que nos atentemos à algumas coisas quando tratamos de features numéricas nos modelos de machine learning, principalmente modelos que utilizam-se da distância entre os registros durante seu aprendizado.

Tratamento de Nulos

Dados nulos, também chamados de ausentes ou de missing, são dados que, literalmente, não foram preenchidos. Esses dados podem atrapalhar - e muito - nossas análises. Porém, decidir o que fazer com eles nem sempre é simples, e tudo depende do que aquele seu nulo quer dizer.

Idade	Gênero	Entrega de doc	Salário
39	B		R$ 5.000
	A	1	R$ 3.000
41	B	1	R$ 4.000
40	B	1	R$ 4.000
	A		

Em geral temos 3 tipos de nulos:

1. Missing not at random (MNAR)
Ocorre quando o motivo de um valor estar ausente é devido ao próprio valor em si. Por exemplo, em um estudo, podemos perceber que alguns valores ausentes aparecem na coluna "Entrega de doc", que indica se a pessoa entregou determinado documento para averiguação. Após investigação, pode-se constatar que a não entrega daquele documento ocasiona um nulo em nossa base de dados, e essa mesma não entrega é um indicativo imenso de que por trás daquele dado existe um fraudador.

2. Missing at random (MAR)
Ocorre quando o valor ausente é devido outra variável. Por exemplo, na tabela acima, podemos perceber que os valores de idade estão frequentemente ausentes para as pessoas do gênero "A", o que pode ser porque as pessoas desse gênero não gostam de divulgar sua idade neste estudo.

3. Missing completely at random (MCAR)
Ocorre quando não há um padrão na ausência do valor. Por exemplo, podemos supor que os valores ausentes para a coluna "Salário" são completamente aleatórios, não devido ao próprio salário e nem por qualquer outra variável. Às vezes, as pessoas simplesmente esquecem de preencher esse valor sem motivo específico. No entanto, é importante observar que esse tipo de ausência é muito raro. Geralmente, existem razões pelas quais certos valores estão ausentes e você deve investigá-las.

Além de terem significado de alguma forma, conforme descrito acima, alguns algoritmos não suportam estes registros nulos (dão erro), logo, é necessário que nós os tratemos de alguma forma. Nós podemos eliminá-los ou substituí-los de alguma forma.

No Pandas, você pode conferir o volume de vazios com um simples comando:

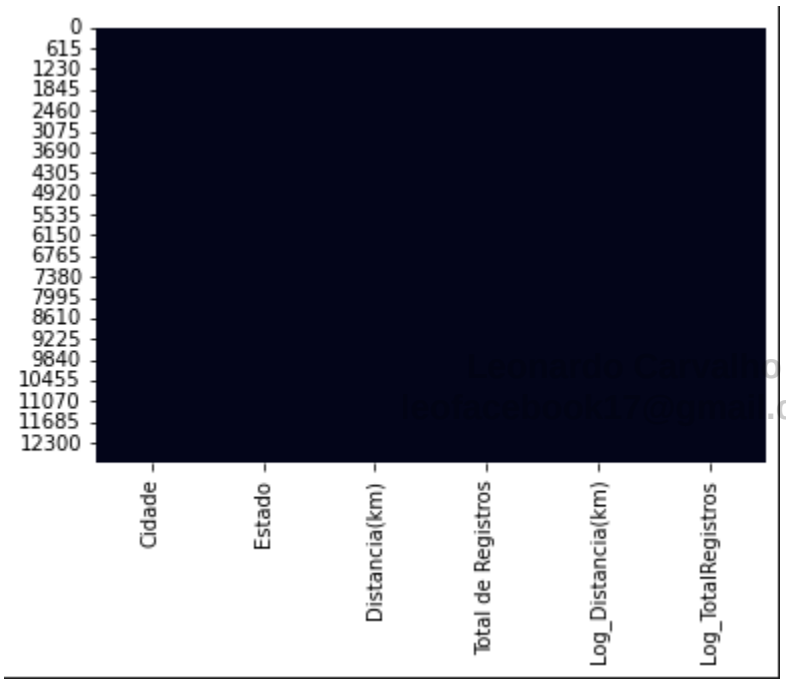
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 12896 entries, 0 to 12895
Data columns (total 6 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   Cidade              12896 non-null  object
 1   Estado              12896 non-null  object
 2   Distancia(km)       12896 non-null  float64
 3   Total de Registros  12896 non-null  int64
 4   Log_Distancia(km)   12896 non-null  float64
 5   Log_TotalRegistros  12896 non-null  float64
dtypes: float64(3), int64(1), object(2)
memory usage: 604.6+ KB
```

Você pode comparar o total de registros do conjunto de dados com a contagem de não vazios de cada coluna, assim você saberá se existem vazios no seu conjunto.

Outra maneira é plotar isto visualmente para você ter uma melhor percepção dos valores vazios do seu conjunto:

```
# visualizando preenchimento através de uma matriz
sns.heatmap(df.isnull(), cbar=False)
```



No nosso exemplo, todas as colunas estão 100% preenchidas, por isso não aparecem falhas no nosso bloco.

Muitas análises ou modelos precisam que a gente lide com os nulos de alguma forma. Vou te dar aqui 3 alternativas do que fazer quando você enfrentar esse problema

1. Remoção
A remoção de nulos é a grande preferida dos candidatos que já entrevistei, não porque seja um método melhor, mas porque é mais fácil de fazer. Nesse caso, você pode:

• Por coluna: se uma variável tiver muitos valores ausentes, basta remover essa coluna inteira. A desvantagem dessa abordagem é que você pode estar excluindo informações importantes e reduzindo a precisão do seu modelo.
• Por linha: se uma amostra tiver valores ausentes, basta remover essa amostra. Esse método pode funcionar quando os valores ausentes são completamente aleatórios (MCAR) e o número de exemplos com valores ausentes é pequeno e se você tem um bom número de amostras preenchidas. No entanto, a remoção de linhas de dados também pode eliminar informações importantes que o seu modelo precisa para fazer revisões, especialmente se os valores ausentes não forem aleatórios (MNAR).

2. Input
Embora a exclusão seja tentadora porque é fácil de fazer, excluir dados pode levar à perda de informações importantes e introduzir vieses no seu modelo. Se você não quiser excluir os valores ausentes, precisará imputá-los, ou seja, "preenchê-los com certos valores". Decidir quais "certos valores" usar é uma parte difícil.

Uma prática comum é preencher os valores ausentes com a média, mediana ou moda da própria coluna. Por exemplo, se não temos alguns valores de salário, poderíamos preencher o nulo com a média de salário de todas as pessoas da empresa. Outro caso, suponhamos que não temos o valor da temperatura para uma amostra de dados cujo mês é julho - nesse caso, não é uma má ideia preenchê-lo com a temperatura média de julho.

Note que a estatística escolhida depende muito da distribuição dos seus dados. Se você tem outliers, por exemplo, pode preferir a mediana ao invés da média.

```
df['notas']= df['notas'].fillna(df['notas'].median())
```

Outro caso é preencher com algo que faça mais sentido, por exemplo, se sei que aquela pessoa é da área de RH, poderíamos preencher com médias de salário do RH. Nesse sentido, existem técnicas mais avançadas de imputação também, como o KNN inputer. KNN significa "K-Nearest Neighbors" (K-Vizinhos Mais Próximos). É um algortimo de machine learning que vamos falar mais sobre mais a frente, mas aqui está como ele funciona, em termos bem simples:

Primeiro, você decide um número K que representa quantos vizinhos próximos você quer considerar. Por exemplo, se K=3, você vai olhar para os 3 vizinhos mais próximos. Para cada ponto de dado que está faltando (como uma nota ausente), o KNN Imputer olha para os outros pontos de dados que são mais similares ou "próximos" a ele (por exemplo, outros alunos com notas em disciplinas semelhantes). Depois de encontrar os K vizinhos mais próximos, o imputer calcula o valor faltante com base nos valores desses vizinhos. Geralmente, isso é feito tirando a média dos valores desses vizinhos.

```
imputer = KNNImputer(n_neighbors=2)
df_imputed = imputer.fit_transform(df[['notas']])
```

3. New feature

Uma técnica excelente que pode ser realizada mesmo que você decida remover ou imputar o dado é a criação de uma feature booleana (0 ou 1) indicando se a feature ao lado é missing.
Isso é muito bom nos casos MNAR e MAR. Por exemplo, podemos criar uma coluna indicando se "Entrega de doc" estava missing ou não - se estava, preenchemos com 1. Se não estava, preenchemos com 0.

Independentemente da maneira das técnicas que você usa, uma coisa é certa: não existe uma perfeita de lidar com valores ausentes. Com a remoção, você corre o risco de perder informações importantes ou vieses acentuados. Com a imputação, você corre o risco de adicionar ruído aos seus dados ou, pior ainda, vazamento de dados. Mas esse é um assunto para um próximo post!

Tratamento de Outliers

Já falamos sobre outliers na parte de estatística, então serei mais breve aqui. Basicamente você precisa entender se seu outlier é um erro ou não. Se for um erro, remova! Se não for, muito cuidado, pois ele pode ser justamente o padrão que você está procurando. Eu raramente recomendo a remoção de um outlier a não ser que seja realmente muito bem justificado - por exemplo, em alguns business dados durante o período de covid são considerados um outlier. Como sabemos (ou esperamos) que isso não se repita, poderíamos remove-los.

Normalização ou Escalonamento (Scalers)

A maioria dos modelos que medem distância entre registros, como K vizinhos mais próximos, ou que são baseados no gradiente descendente, podem ser influenciados pelos intervalos de valores entre diferentes atributos.

Imagine que você está colocando o peso em gramas com o preço de um produto lado a lado em seu modelo. Você vai perceber que os valores de gramas são muito maiores do que os de preço, logo, o modelo tende a dar mais importância para esta feature em detrimento da outra.

Name	Weight	Price
Orange	15	1
Apple	18	3
Banana	12	2
Grape	10	5

Para corrigir este comportamento, é necessário realizar o escalonamento destas features para um padrão onde uma feature não tenha dominância sobre a outra.

Note que alguns algoritmos não necessitam deste tipo de tratamento, mas vamos discutir isso quando chegarmos em cada algoritmo. Por hora, atente-se ao fato de ser importante para algoritmos baseados em distancia ou gradiente descendente.

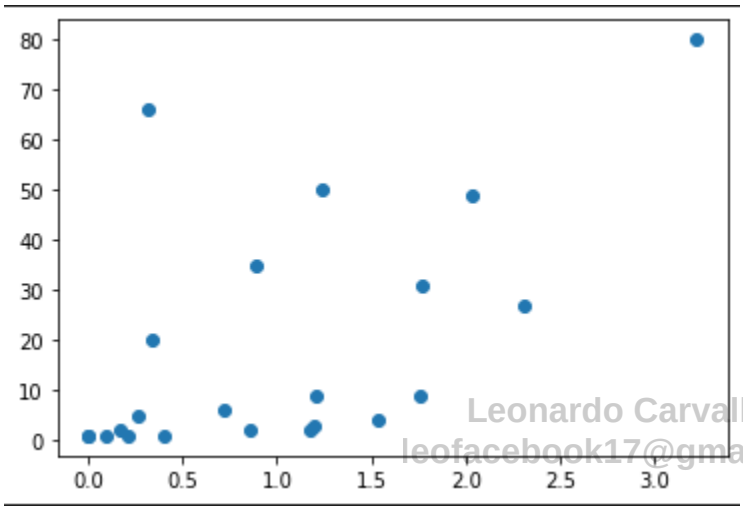
Existem algumas maneiras de escalar nossas features, vamos aprender aqui as principais, utilizando Pandas e Scikit-learn do Python. Vamos continuar tratando as mesmas features dos exemplos anteriores, mas agora vou simular os diversos escalares na mesma feature para que possamos compreender suas diferenças.

Min-Max scaler

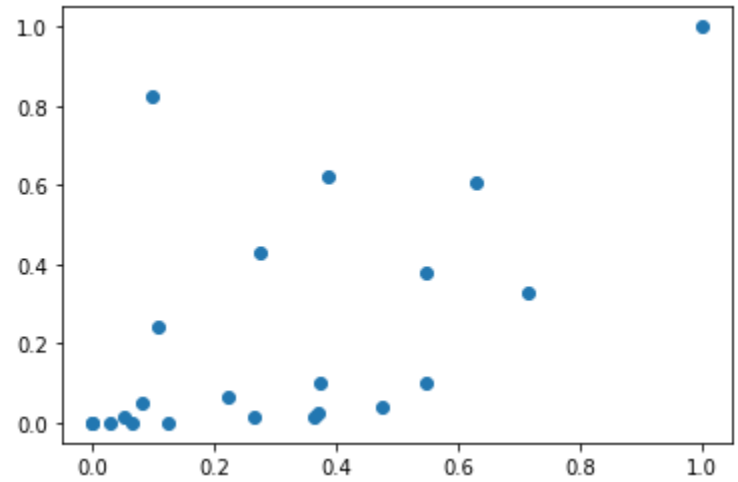
Esse Scaler traduz os registros da escala original em uma nova escala determinada, como por exemplo [0-1], e todos os valores do intervalo original são transferidos proporcionalmente para este novo intervalo.

Este Scaler responde bem se o desvio padrão for pequeno e quando uma distribuição não for Gaussiana. Este Scaler é sensível a Outliers.

```
#Distribuição Original para referência
valores_df = df.iloc[1:23]
valores_df = valores_df[['Distancia(km)','Total de Registros']].values
plt.scatter(valores_df[:,0], valores_df[:,1])
```




```
#Min-Max Scaler
from sklearn.preprocessing import MinMaxScaler
scalar = MinMaxScaler().fit_transform(valores_df)
plt.scatter(scalar[:,0], scalar[:,1])
```

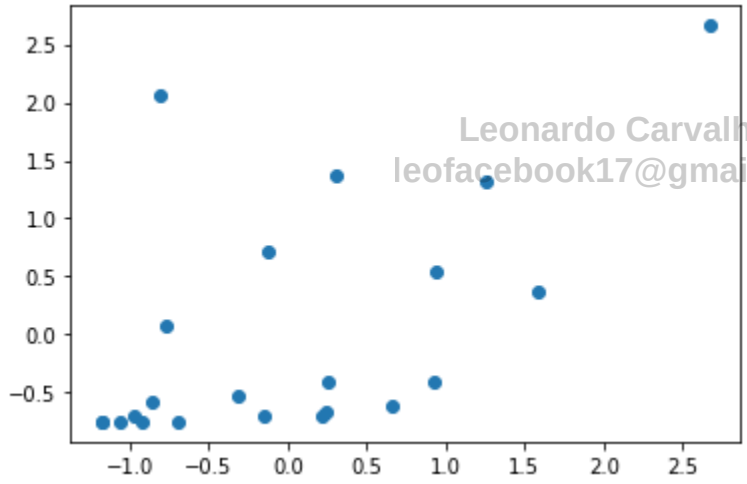


Standard Scaler

O Standard Scaler assume que os dados são normalmente distribuídos dentro desta feature e os dimensiona de forma que a distribuição seja centrada em torno de 0, com um desvio padrão de 1.

Se os dados não forem distribuídos normalmente, este não é o melhor Scaler a ser usado.

```
#Standard Scaler
from sklearn.preprocessing import StandardScaler
scalar = StandardScaler().fit_transform(valores_df)
plt.scatter(scalar[:,0], scalar[:,1])
```

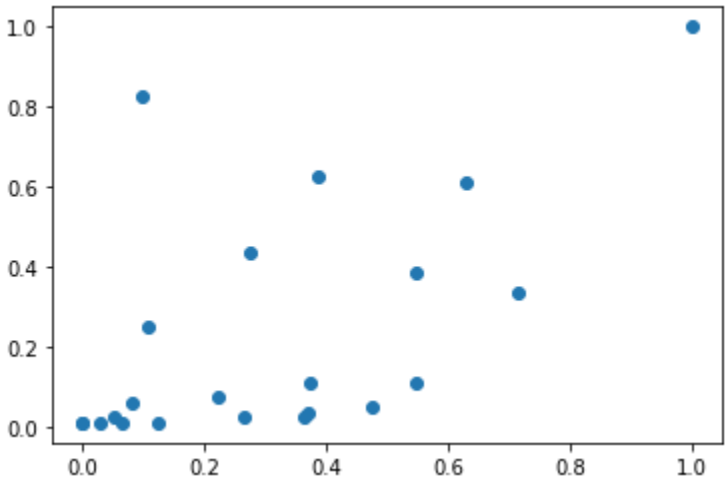


Max Abs Scaler

Redimensiona a escala original de modo que o seu valor máximo absoluto se torne 1 e cada valor inferior seja dividido pelo número máximo, retornando um percentual. Ele não desloca/centraliza os dados e, portanto, não destrói nenhuma dispersão.

Em dados somente positivos, esse Scaler se comporta de maneira semelhante ao Min Max Scaler e, portanto, também sofre com a presença de outliers de forma significante.

```
#MaxAbsScaler
from sklearn.preprocessing import MaxAbsScaler
scalar = MaxAbsScaler().fit_transform(valores_df)
plt.scatter(scalar[:,0], scalar[:,1])
```

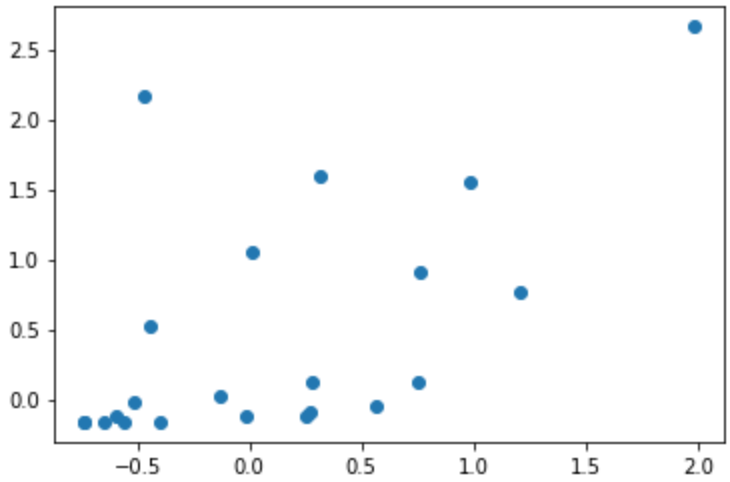


Robust Scaler

Como o nome sugere, este Scaler é resistente à outliers, e funciona da seguinte maneira: Primeiro, a mediana é removida do conjunto de dados, e depois, os dados são escalados pelo IQR (InterQuartile Range).

Observe que os próprios outliers ainda estão presentes nos dados transformados, apesar de não serem tão influentes.

```
#RobustScaler
from sklearn.preprocessing import RobustScaler
scalar = RobustScaler().fit_transform(valores_df)
plt.scatter(scalar[:,0], scalar[:,1])
```

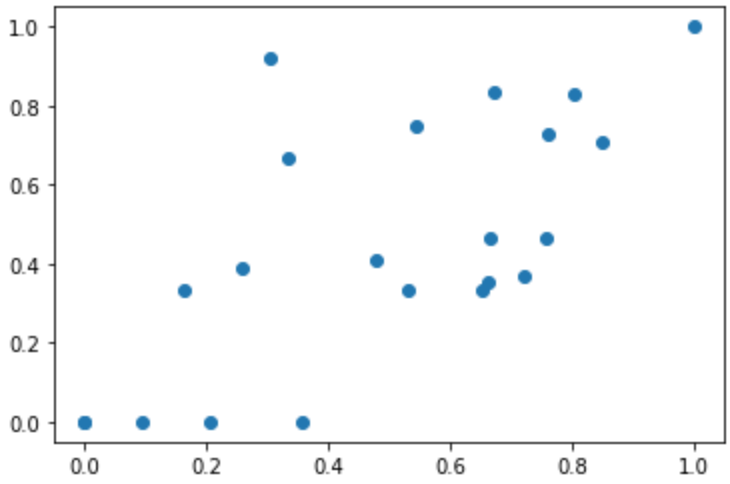


Quantile Transformer Scaler

Este Scaler transforma os dados para seguir uma distribuição normal. Portanto, para uma determinada feature, essa transformação tende a espalhar os valores que estão no centro, por outro lado, ela reduz o impacto de outliers. Logo, podemos entender que este Scaler também é robusto.

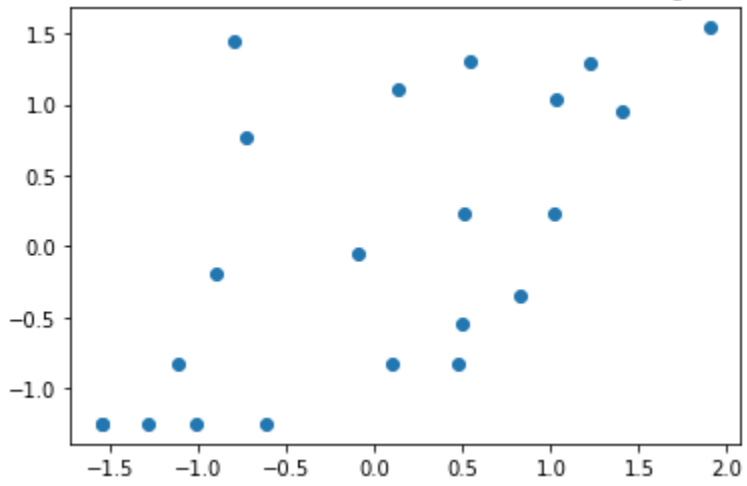
A função de distribuição cumulativa de uma feature é usada para projetar os valores originais. Observe que essa transformação não é linear e pode distorcer as correlações lineares entre as variáveis medidas na mesma escala, mas torna as variáveis medidas em diferentes escalas mais diretamente comparáveis.

```
#QuantileTransformer
from sklearn.preprocessing import QuantileTransformer
scalar = QuantileTransformer(n_quantiles=4).fit_transform(valores_df)
plt.scatter(scalar[:,0], scalar[:,1])
```



Power Transformer Scaler

```
#PowerTransformer
from sklearn.preprocessing import PowerTransformer
scalar = PowerTransformer(method='yeo-johnson').fit_transform(valores_df)
plt.scatter(scalar[:,0], scalar[:,1])
```

Note que na maioria dos casos, as relações entre as duas variáveis se mantêm, mas os seus valores absolutos foram escalados de forma que as features possam ser equivalentes entre si na interpretação do modelo.

Alguns scalers podem dispensar a utilização da transformação log, como o caso do Power Transformer, que naturalmente normaliza nossa amostra de dados no processo de escalonamento.

Tratando Valores Categóricos

Agora que já entendemos como tratar os valores numéricos e seus Scalers, vamos entender como podemos tratar dos valores categóricos. Geralmente os valores categóricos são apresentados como texto, e como boa parte dos algoritmos não conseguem interpretar textos, devemos fazer alguns tratamentos para que o algoritmo consiga processar alguma interpretação numérica destes textos.

Como boa parte dos aspectos no aprendizado de máquina, não existe uma técnica que funcione como uma bala de prata para todos os casos, então é necessário que você interprete o contexto do problema e aplique a melhor técnica em cada caso.

Vou apresentar aqui as principais abordagens para o tratamento de valores categóricos, e que devem cobrir boa parte dos problemas com este tipo de feature no seu dia a dia.

Label Encoding

O Label Encoding nada mais é do que a conversão de cada um dos valores de determinada feature em um equivalente numérico. Este tipo de abordagem pode ser usada quando você precisa codificar valores e não pode quebrar esta feature em várias features binárias subsequentes.

No pandas isto pode ser feito de forma fácil através da seguinte abordagem:

```
df['Cod_Estado'] = df['Estado'].astype('category').cat.codes
df.head()
```

	Cidade	Estado	Distancia(km)	Total de Registros	Log_Distancia(km)	Log_TotalRegistros	Cod_Estado
0	Aaronsburg	PA	0.478779	2	0.000000	0.693147	36
1	Abbeville	LA	0.000000	1	0.000000	0.000000	16
2	Abbeville	SC	0.342226	20	0.000000	2.995732	38
3	Abbotsford	WI	1.536920	4	0.429780	1.386294	46
4	Abbottstown	PA	1.203071	9	0.184878	2.197225	36

LABEL ENCODING

TV show	TV show encoder	Target
Dexter	1	10
Dark	2	9.5
La casa de Papel	3	9
Dark	2	8

Replace each category for a number

Issue: May create a non desirable hierarchy

O uso do Label Encoder requer cautela, especialmente quando você está lidando com variáveis categóricas nominais (ou seja, categorias sem uma ordem inerente). Aqui estão os principais motivos para ter cuidado:

1. Introdução de uma ordem artificial: O Label Encoder converte categorias em números sequenciais (0, 1, 2, 3, ...). Isso pode inadvertidamente introduzir uma relação de ordem ou hierarquia onde não existe, levando modelos de machine learning a interpretar que uma categoria é "maior" ou "mais importante" que outra, o que pode distorcer os resultados em modelos que assumem uma relação ordinal ou numérica entre os valores (como regressão linear, modelos baseados em árvores, etc.).

2. Impacto no desempenho do modelo: Algoritmos de machine learning podem interpretar erroneamente os números codificados como tendo uma relação de magnitude ou ordem (por exemplo, 4 é o dobro de 2, o que não faz sentido em um contexto categórico como "cor: vermelho = 1, azul = 2"). Isso pode afetar negativamente a performance do modelo, pois o modelo pode aprender padrões que não existem de fato nos dados.

One Hot Encoding

A codificação anterior tem a vantagem de ser direta, mas tem a desvantagem de que os valores numéricos podem ser “mal interpretados” pelos algoritmos. Por exemplo, o valor de 0 é obviamente menor que o valor de 4, mas isso realmente corresponde ao conjunto de dados na vida real? Um estado tem “4X” mais peso em nosso cálculo do que outro estado? Neste exemplo, acho que não.

Uma abordagem alternativa comum é chamada de One Hot Encoding. Esta estratégia basicamente converte cada valor de categoria em uma nova coluna e atribui um valor 1 ou 0 (Verdadeiro/Falso) à coluna. Isso tem a vantagem de não ponderar um valor incorretamente, mas tem a desvantagem de adicionar mais colunas ao conjunto de dados.

No Pandas, podemos realizar esta transformação da seguinte maneira:

```
df = pd.get_dummies(df, columns=['Estado'])
df.head()
```


	Cidade	Distancia(km)	Total de Registros	Log_Distancia(km)	Log_TotalRegistros	Estado_AL	Estado_AR	Estado_AZ	Estado_CA	Estado_CO	...
0	Aaronsburg	0.478779	2	0.000000	0.693147	0	0	0	0	0	...
1	Abbeville	0.000000	1	0.000000	0.000000	0	0	0	0	0	...
2	Abbeville	0.342226	20	0.000000	2.995732	0	0	0	0	0	...
3	Abbotsford	1.536920	4	0.429780	1.386294	0	0	0	0	0	...
4	Abbottstown	1.203071	9	0.184878	2.197225	0	0	0	0	0	...

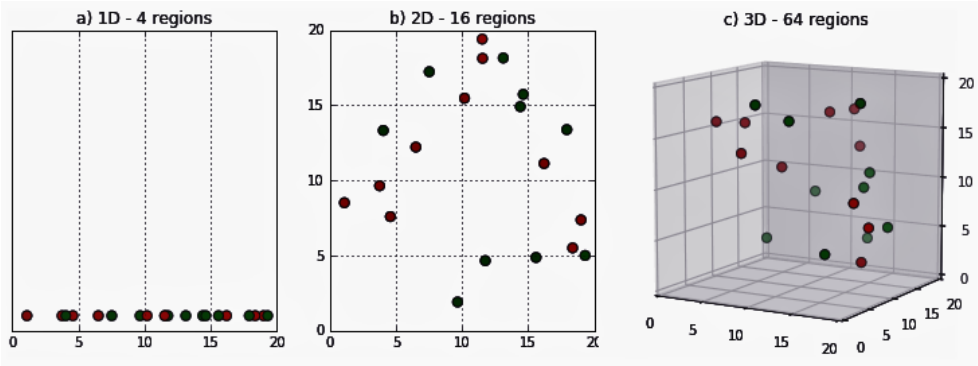
Você pode utilizar estas técnicas em combinação com as técnicas de geração de novas features e criar combinações poderosas para seu algoritmo, de acordo com cada necessidade de negócio.

One hot encoding			
Dexter	**Dark**	**La casa de papel**	**Target**
1	0	0	10
0	1	0	9.5
0	0	1	9
0	1	0	8

Each category becomes a column and we fill it with 0 or 1

Issue: When there're too many categories in a feature

Contudo, como sempre, one hot encoding também pode se tornar um problema. A "maldição da dimensionalidade" é um termo usado para descrever os problemas e desafios que surgem ao trabalhar com dados de alta dimensão, especialmente em contextos de análise de dados e machine learning. À medida que o número de características (dimensões) em um conjunto de dados aumenta, o espaço de dados se expande exponencialmente, o que pode levar a várias complicações.
Note que quando temos 1 só dimensão (1D), os dados estão mais juntinhos. Quando expandimos isso, os dados vão ficando mais esparsos



O One-Hot Encoding transforma variáveis categóricas em um formato binário, criando uma nova coluna para cada categoria única. Por exemplo, se você tem uma coluna categórica com 100 categorias únicas, o One-Hot Encoding resultará em 100 novas colunas binárias. Isso pode aumentar drasticamente a dimensionalidade do conjunto de dados. Com muitas dimensões, a maioria dos dados se torna esparsa (ou seja, preenchida com zeros), o que significa que a distância entre os pontos de dados pode se tornar

enganosamente grande e menos significativa. Isso dificulta para muitos algoritmos de machine learning identificar padrões efetivamente, pois a presença de muitos zeros pode diluir a informação útil.

Em um espaço de alta dimensão, há uma necessidade exponencialmente maior de dados para preencher o espaço e para que os modelos de machine learning aprendam de forma eficaz. Sem dados suficientes, o modelo pode se ajustar demais (overfit) às particularidades do conjunto de treinamento, capturando ruído em vez de sinais reais. Isso prejudica a capacidade do modelo de generalizar bem para novos dados não vistos.

Além disso, aumentar o número de características através do One-Hot Encoding também aumenta a carga computacional para processar os dados e treinar os modelos. Isso exige maior demanda por memória e poder de computação pode tornar o treinamento de modelos mais lento e menos eficiente, especialmente quando se lida com grandes volumes de dados.

Target Encoding

O Target Encoder faz o encoding de uma maneira especial: ele olha para o target (a coisa que você está tentando prever) e calcula a média dela para cada categoria. Por exemplo, se você está prevendo o preço de casas e a cor da casa é a variável categórica, o Target Encoder vai calcular a média do preço de todas as casas vermelhas, todas as casas azuis, etc.

Depois, ele substitui cada valor categórico pela média do alvo correspondente. Assim, em vez de "vermelho", "azul", "verde", você terá números que representam a média do preço para essas cores. Isso ajuda o modelo de machine learning a entender melhor a relação entre a cor da casa e seu preço.

Ao contrário do Label Encoder, o Target Encoder cria uma representação numérica que está diretamente relacionada ao valor alvo, ajudando o modelo a captar melhor as nuances da variável categórica.

Ele é particularmente útil quando temos muitas categorias únicas, onde métodos como one-hot encoding podem levar a uma alta dimensionalidade.

Target encoding		
TV show	**TV show encoder**	**Target**
Dexter	10	10
Dark	8.75	9.5
La casa de Papel	9	9
Dark	8.75	8

Encode categorical variables by their ratio of target. Do it in a cross-validation manner (subsets from the same dataset)

Issue: May give collisions , but when applied properly it's the best encoding for both linear and nonlinear

O problema é que o target encoding pode levar a overfitting e viés no modelo, especialmente ao lidar com variáveis categóricas de alta cardinalidade. Isso ocorre porque a média do target para uma categoria com apenas alguns exemplos pode não ser representativa da verdadeira distribuição subjacente.

Uma opção é fazer o Kfold do target encoding. Isso funciona dividindo os dados em K partes, onde cada parte é usada como um conjunto de validação, enquanto as outras K-1 partes são usadas como conjunto de treinamento. Para cada parte, a média do target para cada categoria é calculada no conjunto de treinamento e usada para codificar as categorias no conjunto de validação. Isso garante que a codificação seja baseada apenas no conjunto de treinamento e seja menos propensa a overfitting e viés.

Os passos para realizar a Codificação Alvo K-fold são os seguintes:

Dividir os dados em K partes.

Para cada parte:
a. Dividir os dados em conjuntos de treinamento e validação.
b. Calcular a média do target para cada categoria no conjunto de treinamento.
c. Usar a média do target para codificar as categorias no conjunto de validação.

d. Combinar os conjuntos de validação codificados de todas as K partes para criar feature final codificada.

Veja o kfold target encoder em detalhes aqui

Material complementar

Resumo sobre Feature Engineering

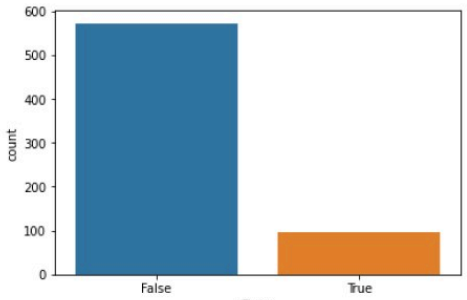
Transformação de Dados Categóricos

Normalização de Variáveis

Porque e quando normalizar os dados

Dados desbalanceados

O desbalanceamento de dados é um problema crítico no contexto de aprendizado de máquina e mineração de dados. Ele surge quando há uma quantidade significativamente menor de amostras em uma classe em comparação com outra (no target), o que pode prejudicar o desempenho da maioria dos algoritmos de machine learning que são projetados para trabalhar com conjuntos de dados balanceados.



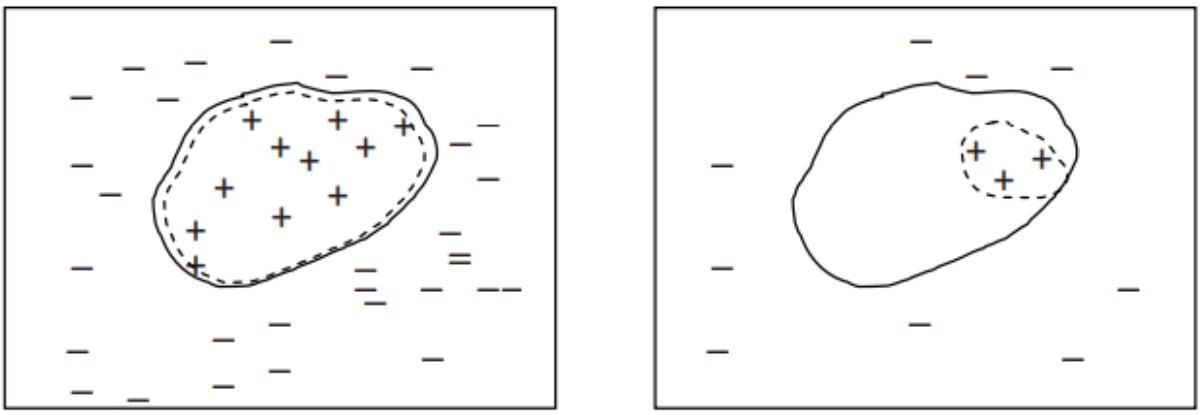
Acima você pode ver que nosso target (churn) tem muito mais amostras False do que True. Isso significa que menos clientes deram churn (desistiram do serviço), o que é totalmente esperado nesse caso! Afinal, se a maior parte dos clientes desistissem dos nossos serviços, nosso business estaria muito mal das pernas.

Como os dados tornam-se desbalanceados, e qual a consequência?

O desbalanceamento de dados pode ser uma característica intrínseca de alguns problemas, como em análises de fraudes de cartão de crédito, em que é esperado que haja uma quantidade maior de transações legítimas do que indevidas. No entanto, em outros casos, a desigualdade na distribuição dos dados pode estar relacionada a problemas na coleta de dados, como custo, privacidade, entre outros fatores.

Independentemente da causa do desbalanceamento, ele pode introduzir desafios significativos na avaliação e treinamento de modelos de machine learning. Por exemplo, a métrica de avaliação pode estar enviesada em favor da classe majoritária, levando a conclusões equivocadas e desfavorecendo a classe minoritária.

Além disso, algoritmos de machine learning, como árvores de decisão, análise discriminante e redes neurais, são projetados para trabalhar com conjuntos de dados balanceados, o que pode resultar em um limite de decisão enviesado em favor da classe majoritária e aumentar a probabilidade de classificação incorreta das instâncias de classes minoritárias. Um exemplo disto é a imagem abaixo, onde o limite de decisão na figura b não representa a realidade por causa do desbalanceamento.



(a) (b)

Outro problema relacionado ao desbalanceamento é o impacto do ruído nas classes minoritárias, uma vez que os algoritmos de machine learning podem ter a tendência de tratar essas amostras como ruído e, assim, subestimá-las ou descartá-las durante o treinamento do modelo. Ademais, o tamanho do conjunto de dados é um fator importante a ser considerado na construção de um bom classificador, pois a falta de exemplos pode tornar difícil descobrir regularidades em classes pequenas. Em geral, quanto mais dados de treinamento estiverem disponíveis, menos sensíveis serão os classificadores em relação às diferenças entre as classes.

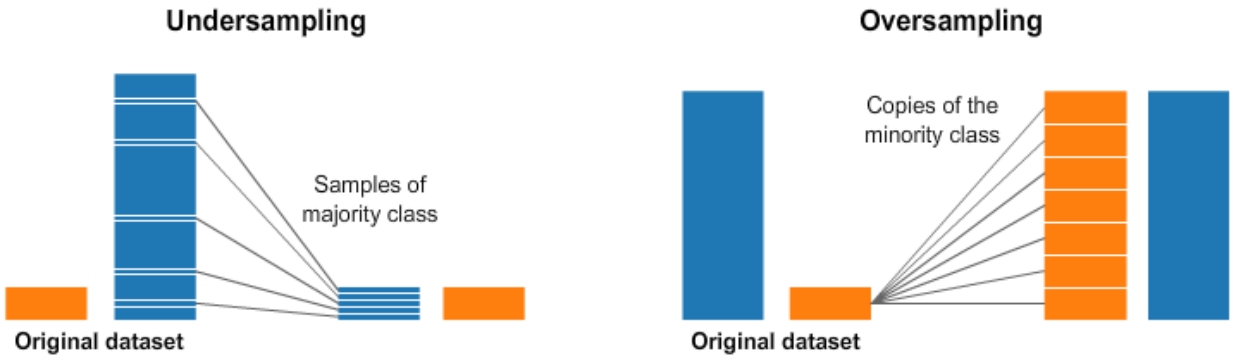
Para lidar com o desbalanceamento de dados, diversas abordagens têm sido propostas, como a reamostragem dos dados, o ajuste de pesos, a utilização de algoritmos especializados em lidar com dados desbalanceados, entre outras técnicas. Essas abordagens visam melhorar a qualidade do modelo, aumentando a precisão na classificação de instâncias de classes minoritárias e, consequentemente, o desempenho geral do modelo em tarefas de aprendizado de máquina, e serão discutidas a seguir.

Amostragem

A amostragem é uma técnica comumente utilizada para lidar com o desbalanceamento de dados em problemas de aprendizado de máquina. Essa técnica tem como objetivo pré-processar os dados de treinamento para minimizar a discrepância entre as classes, modificando as distribuições no conjunto de treinamento.

A amostragem pode ser dividida em três categorias: subamostragem (undersampling), superamostragem (oversampling) e técnica híbrida. Na subamostragem, o objetivo é reduzir a presença da classe majoritária, selecionando um subconjunto dessa classe para compor o conjunto de treinamento. Esta estratégia procura equilibrar a proporção entre as classes, diminuindo o risco de o modelo ser enviesado pela classe majoritária. Entretanto, uma desvantagem dessa abordagem é a possibilidade de perder dados úteis, uma vez que uma parcela considerável da classe majoritária é descartada

Por outro lado, a superamostragem aumenta a quantidade de instâncias da classe minoritária, replicando as amostras existentes no conjunto de treinamento. Esse método tem o objetivo de reforçar a presença da classe minoritária, para que ela não seja negligenciada durante o treinamento do modelo. Entretanto, essa técnica pode levar ao sobreajuste, uma vez que a mesma instância é repetida várias vezes, fazendo com que o modelo se ajuste demasiadamente a esses exemplos específicos.



A técnica híbrida, como sugere o nome, combina a subamostragem e a superamostragem em um único método. Essa abordagem busca obter o equilíbrio ideal entre as classes, minimizando as desvantagens de ambas as

técnicas, preservando a diversidade da classe majoritária e reforçando a representatividade da classe minoritária.

É importante ressaltar que a escolha da melhor distribuição de classes irá depender das medidas de desempenho específicas do problema e pode variar de um conjunto de dados para outro. Alguns exemplos de técnicas a serem utilizados são:

Na minha experiência, o undersampling costuma funcionar muito bem quando temos um conjunto grande de dados!

Class Weight

A ideia por trás do uso do class weight é dar mais importância às classes menos frequentes durante o treinamento do modelo. Isso é feito atribuindo pesos maiores às classes menos frequentes e pesos menores às classes mais frequentes. Dessa forma, o modelo é incentivado a prestar mais atenção às classes menos frequentes, melhorando seu desempenho na classificação dessas classes sem comprometer significativamente o desempenho geral.

Os pesos das classes podem ser definidos manualmente ou calculados automaticamente com base na frequência das classes. Um método comum para calcular os pesos automaticamente é usar a inversa da frequência das classes, de modo que as classes menos frequentes recebam pesos maiores e as mais frequentes recebam pesos menores. Essa abordagem ajuda a equilibrar o impacto de cada classe no custo total da função de perda durante o treinamento, incentivando o modelo a aprender a classificar corretamente as amostras de todas as classes.

Imagine que você tem 100 emails, onde 90 são emails normais (não-spam) e 10 são spam. Você tem um programa que tenta identificar quais emails são spam e quais não são. Inicialmente, sem ajustes especiais, o programa faz alguns erros:

- Erra ao marcar 2 emails normais como spam.

- Não consegue identificar 3 spams, tratando-os como emails normais.

Então, na situação inicial, o programa é muito bom em identificar emails normais, acertando quase sempre (97,8% das vezes), mas não tão bom em encontrar spams, acertando apenas 70% das vezes.

Para melhorar a habilidade do programa de identificar spam, decidimos dizer ao programa que encontrar spam é mais importante. Fazemos isso dando mais "pontos" sempre que ele acerta um spam, de forma que errar em spams se torne mais "caro" para ele do que errar em emails normais.

Após essa mudança, o programa passa a errar um pouco mais nos emails normais, marcando 4 como spam, mas melhora muito em identificar spams, errando apenas em 1.

Agora, mesmo o programa errando um pouco mais nos emails normais, sua habilidade de encontrar spam melhorou muito, indo de 70% para 90% de acerto. Isso significa que, de cada 10 spams, ele agora acerta 9, em vez de apenas 7, como antes.

Esse ajuste é como se disséssemos ao programa: "Olha, nós realmente não queremos perder esses spams, então preste mais atenção neles, mesmo que isso signifique cometer alguns erros a mais com os emails normais." Esse tipo de ajuste ajuda muito quando é mais importante não perder certos tipos de erros, como não deixar passar um spam.

A implementação de pesos de classe varia dependendo do framework de aprendizado de máquina utilizado (como scikit-learn, TensorFlow, PyTorch, etc.), mas a maioria desses frameworks oferece alguma forma de suporte para essa técnica. Ao usar pesos de classe, é importante experimentar com diferentes valores de pesos para encontrar o equilíbrio certo que melhore o desempenho do modelo nas classes de interesse sem prejudicar demais o desempenho nas outras classes.

Se quiser ver um exemplo em Python, acesse:

https://www.educative.io/answers/how-to-set-weights-for-imbalanced-classes

Outras técnicas

- **SMOTE (Synthetic Minority Over-sampling Technique)**: Criação de amostras sintéticas da classe minoritária para equilibrar a distribuição. Por criar amostras sintéticas, não costuma ser um bom approach. Caso

queira ler mais sobre o assunto, recomendo esse artigo

- **ADASYN (Adaptive Synthetic Sampling)**: Uma variação do SMOTE que ajusta o número de amostras sintéticas geradas conforme a necessidade de cada classe minoritária.
- **SVM SMOTE**: Integração do SMOTE com Support Vector Machines para uma amostragem mais eficiente e precisa.
- **RUS (Random Under Sampling)**: Redução do número de amostras na classe majoritária para equilibrar o conjunto de dados.
- **NEAR MISS**: Técnicas de under-sampling que selecionam amostras da classe majoritária com base na proximidade com a classe minoritária.
- **TOMEK LINKS**: Identificação e eliminação de pares de amostras de classes opostas que estão próximas (links Tomek), refinando o conjunto de dados.
- **EDITED NEAREST NEIGHBOURS**: Remoção de amostras da classe majoritária que têm vizinhos mais próximos da classe minoritária, aprimorando a qualidade do conjunto de dados.

Importante: Apesar de o SMOTE ser validado na academia, nós não indicamos a sua utilização porque há um consenso no mercado de trabalho que o mesmo não generaliza bem. Indicamos a utilização do class weight que geralmente possui resultados melhores. Veja o artigo indicado no SMOTE

Material complementar

https://pdfs.semanticscholar.org/b85b/5b7c8aeed21eaeaebebc47400a1c009e7a1b.pdf
https://imbalanced-learn.org/stable/references/under_sampling.html
https://imbalanced-learn.org/stable/references/over_sampling.html
https://medium.com/@daniele.santiago/aprenda-a-balancear-seus-dados-com-undersampling-e-oversampling-em-python-6fd87095d717

Overfit e underfit

Os fenômenos de underfit e overfit representam desafios frequentes na criação de modelos preditivos

Overfit: Acontece quando um modelo é tão detalhadamente ajustado aos dados de treinamento que falha em generalizar para novos conjuntos de dados. Isso significa que, apesar de apresentar excelente desempenho nos dados de treino, o modelo não tem a mesma eficácia ao ser aplicado a dados novos. É como se o modelo tivesse memorizado os dados específicos, em vez de entender os padrões subjacentes. Como resultado, o modelo acaba sendo excessivamente complicado, capturando até o ruído nos dados de treinamento.

É semelhante ao estudante que memoriza as respostas dos exercícios, acerta todas na prática, mas falha no exame. Este estudante "overfitou"!

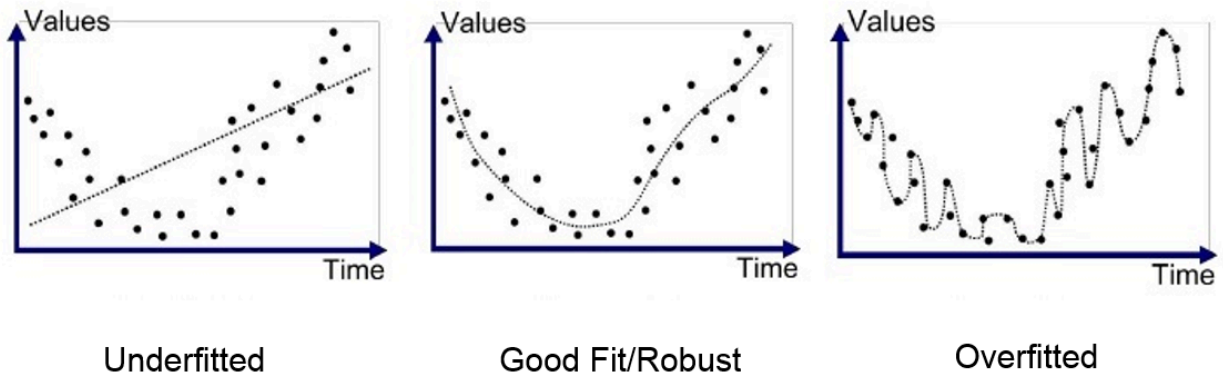
Detectar o overfit geralmente ocorre durante a etapa de validação. Por exemplo, se o modelo tem um desempenho excepcional nos dados de treinamento mas falha no teste, isso indica fortemente overfit. Existem estratégias para mitigar esse problema, como a regularização, que serão exploradas em conteúdos futuros.

Underfit: É o oposto do overfit. Em vez de capturar excessivamente a variação dos dados (incluindo os erros), o underfit ocorre quando o modelo não consegue identificar adequadamente os padrões nos dados. O modelo é tão simplista que não aprende de maneira efetiva as relações nos dados de treinamento.

Geralmente, o underfit é evidente já nos resultados de treinamento, onde o modelo não apresenta bom desempenho nem durante o treinamento nem na validação. Para combater o underfitting, pode-se escolher um modelo mais complexo, aumentar o número de características (features) ou melhorar o pré-processamento dos dados.

Encontrar um ponto de equilíbrio entre o sobreajuste e o underfitting é crucial, exigindo ajustes no modelo e na estratégia de validação. Esse equilíbrio é essencial para a modelagem preditiva em aprendizado de

máquina, destacando o trade-off entre viés e variância.



Underfitted | Good Fit/Robust | Overfitted

Viés (bias): Refere-se à diferença entre a previsão média feita pelo modelo e o valor real que se deseja prever. Modelos com **alto viés** são simplificados demais, fazendo suposições fortes sobre os dados, o que leva a um desempenho ruim tanto nos dados de treinamento quanto de teste, caracterizando o **underfit**.

Variância (variance): Representa quão variáveis são as previsões do modelo para um ponto de dados específico. Modelos com **alta variância** são excessivamente detalhados em relação aos dados de treinamento, não conseguindo generalizar para novos dados, o que é um indicativo de **overfit**.

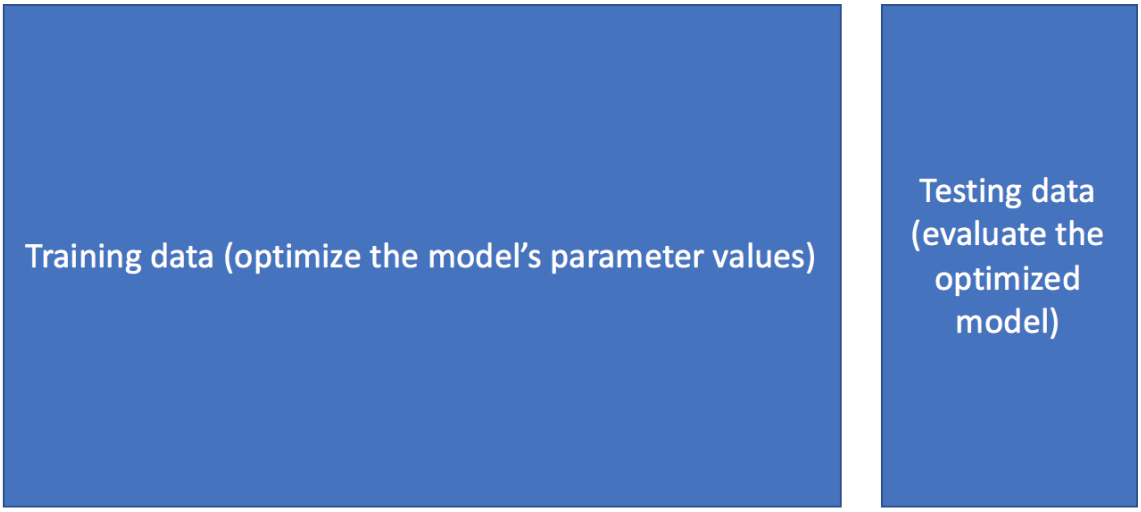
O ideal é desenvolver um modelo que apresente baixo viés e baixa variância, o que é desafiador. Aumentar a complexidade do modelo pode fazer com que ele aprenda ruídos, considerando até variações que não são relevantes. Por outro lado, modelos demasiadamente simples não capturam bem a complexidade dos fenômenos estudados. Por isso, busca-se um equilíbrio entre viés e variância.

A validação cruzada é uma técnica comumente utilizada para manejar o trade-off entre viés e variância, permitindo estimar o erro de teste e selecionar o modelo que otimiza esse equilíbrio.

Split de dados - Método holdout

Todas as formas de avaliação, seja da regressão quanto da classificação, devem ser feitas na etapa de treino, teste e validação. Mas o que é isso?
Vamos ver rapidamente a finalidade de **dividir nossos dados** em dados de treinamento, validação e teste. O objetivo final de qualquer modelo de aprendizado de máquina é aprender com exemplos de tal maneira que o modelo seja capaz de generalizar o aprendizado para novas instâncias que ainda não foram vistas.

Em um nível muito básico, você deve treinar em um subconjunto de seu conjunto de dados total, mantendo os dados restantes para avaliação para avaliar a capacidade de generalização do modelo - em outras palavras, "quão bem meu modelo se sairá em dados que não aprendidas diretamente durante o treinamento?"



Isso significa que você verificará a performance do modelo out-of-sample (fora da amostra de treino)

- É usual usarmos 70% dos dados para treino, 30% para teste. Mas não é uma regra!

Problema dessa abordagem

Quando você treina vários modelos em um conjunto de dados, você também vai precisar avaliar a capacidade de cada modelo de generalizar para dados não vistos. No entanto, se você usar os dados de teste para essa avaliação e ficar ajustando o seu modelo sempre com base na resposta do teste, acabará "ajustando" a arquitetura do modelo aos dados de teste - perdendo a capacidade de avaliar verdadeiramente o desempenho do modelo em dados não vistos. Isso às vezes é chamado de "vazamento de dados" - vamos ver mais sobre isso a seguir.

Outro approach: Holdout

Para mitigar isso, podemos **dividir o conjunto de dados total em três subconjuntos**: dados de treinamento, dados de validação e dados de teste. A introdução de um conjunto de dados de validação nos permite avaliar o modelo em dados diferentes dos que foram treinados e selecionar a melhor arquitetura de modelo, enquanto ainda mantém um subconjunto dos dados para a avaliação final no final do desenvolvimento do modelo. Esse método é chamado de **Holdout.**

Aqui, você vai sempre ajustar seu modelo com base no conjunto de validação. Quando achar que seu modelo está bom, vai medir no teste apenas para confirmar - ou seja, a verificação de quão bom está seu modelo no conjunto de teste será feita 1 única vez.

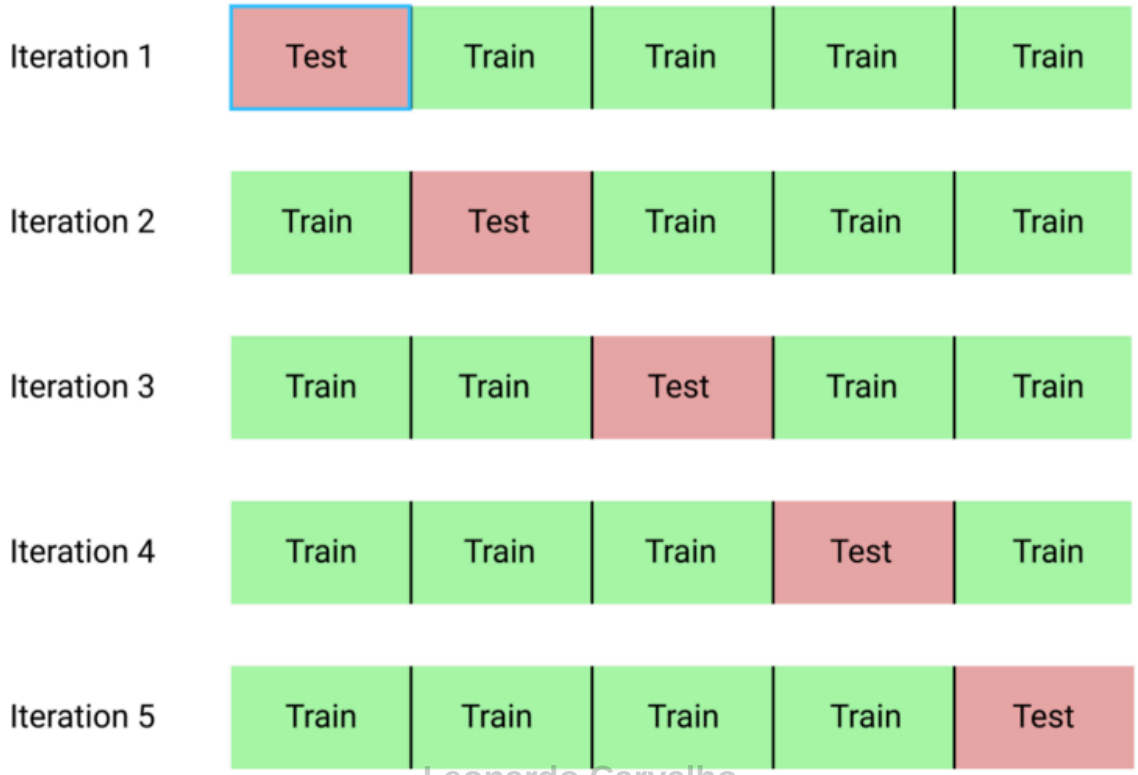
- É usual usarmos 70% dos dados para treino, 15% para teste e 15% para

validação
- Essa abordagem só vai te servir se tiver uma boa quantidade de dados. Caso contrário, 15% pode não representar muito na sua base.

Cross validation (K-fold)

Cross Validation (CV) é uma técnica muito utilizada para avaliação de desempenho de modelos de aprendizado de máquina. O CV consiste em particionar os dados em conjuntos(partes), onde um conjunto é utilizado para treino e outro conjunto é utilizado para teste e avaliação do desempenho do modelo. A utilização do CV tem altas chances de detectar se o seu modelo está sobreajustado aos seus dados de treinamento, ou seja, sofrendo overfitting. Existem mais de um método de aplicação de CV, mas por hora vamos focar no **K-fold.**

K-fold consiste em dividir a base de dados de forma aleatória em K subconjuntos (em que K é definido previamente) com aproximadamente a mesma quantidade de amostras em cada um deles. A cada iteração, treino e teste, um conjunto formado por K-1 subconjuntos são utilizados para treinamento e o subconjunto restante será utilizado para teste gerando um resultado de métrica para avaliação (ex: acurácia). Esse processo garante que cada subconjunto será utilizado para teste em algum momento da avaliação do modelo.



Vamos levar em consideração um cenário de 5-fold cross validation. Os dados serão divididos em 5 partes. Na primeira iteração a primeira parte será utilizada para teste e as partes restantes serão utilizadas para treinamento do modelo gerando uma métrica de avaliação. Na segunda iteração, a segunda parte será utilizada para teste enquanto as demais para treino. Esse processo será repetido 5 vezes até que toda a base passe pelo processo de treino e teste gerando uma métrica de avaliação média para o modelo, conforme a imagem acima.

O valor de K deve ser escolhido cuidadosamente de forma que cada grupo de dados para treino e teste sejam grandes o suficiente para representarem estatisticamente o dataset original. É possível encontrar na literatura a orientação para a utilização de k=5 ou k=10, valores encontrados através de experimentos, mas não é uma regra formal.

Um valor de K mal escolhido pode resultar em uma interpretação errada do desempenho do seu modelo como medidas de avaliação do modelo com alta variância (as medidas podem variar bastante de acordo com os dados

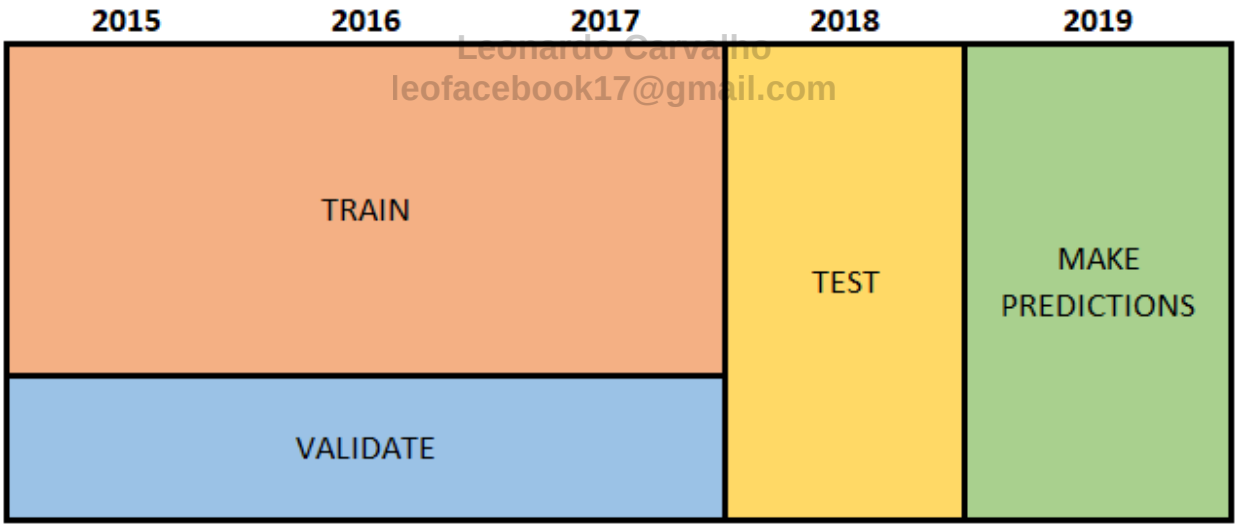
utilizados para treinamento).

A biblioteca sklearn possui uma função chamada **cross_val_score** que podemos utilizar para aplicar o método K-fold cross validation. Veja a documentação aqui.

- Essa é a abordagem que eu mais recomendo!

Out-of-time

Todas as técnicas que falamos acima são técnicas de amostragem out-of-sample. Entretanto, caso você tenha uma coluna de tempo e tenha vários meses ou anos de amostra, também é importante fazer uma validação out-of-time. Ou seja, você vai treinar seu modelo com os primeiros Y meses e vai validar com os últimos X meses. Isso é importante para que o modelo



Leia mais sobre essas técnicas aqui:
https://neptune.ai/blog/cross-validation-in-machine-learning-how-to-do-it-right

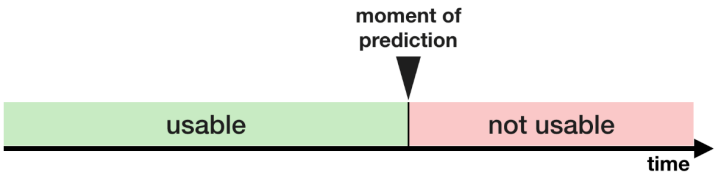
Data Leakage

O que é data leakage?

Data leakage em machine learning refere-se à exposição não intencional ou inadequada de informações dos dados de treinamento para o modelo durante o processo de aprendizagem. Isso pode impactar significativamente o desempenho e a capacidade de generalização do modelo, levando a resultados imprecisos e não confiáveis.

Notavelmente, isso ocorrer pode invalidar completamente uma predição. Geralmente, ele ocorre em datasets mais complexos, como time series, onde a divisão treino/teste é mais difícil. Alguns dos motivos mais comuns para o data leakage acontecer são:

1. **Contaminação Treino-Teste**: Isso acontece quando informações do conjunto de teste (ou validação) vazam para o conjunto de treinamento. Por exemplo, se dados do conjunto de teste são usados por engano durante o feature engineering ou o treinamento do modelo, o modelo pode aprender a memorizar padrões específicos dos dados de teste, resultando em um desempenho excessivamente otimista no conjunto de teste, mas com uma má generalização para dados não vistos.
2. **Vazamento da Target**: O vazamento da target ocorre quando dados diretamente relacionados ao target (a variável que o modelo está tentando prever) são incluídos nos dados de treinamento. Esses dados podem não estar disponíveis durante a previsão no mundo real, e incluí-los pode levar a uma precisão do modelo alta, que não irá corresponder à realidade. Esse problema geralmente surge quando os dados são gerados usando informações que não estariam disponíveis durante a previsão, fazendo com que o modelo "trapaceie" durante o treinamento.
3. **Usar informações futuras para prever o passado**: Por exemplo, usar dados relacionados ao target coletados após o valor do target ter sido registrado.



Exemplo de um caso de data leakage para um case de fraude

Um conjunto de dados é coletado ao longo do tempo, incluindo informações sobre a variável alvo (fraudulenta ou não) e features sobre as transações.

Ao fazer a modelagem, um cientista de dados verifica que há uma feature chamada "data da transação". Para melhorar o modelo, ele resolve criar uma nova feature chamada "dias desde a última fraude". Essa feature calcula o tempo (em dias) desde a última transação fraudulenta de um usuário.

O problema aqui é que essa nova feature, "dias desde a última fraude", não estaria disponível durante a transação. Ela efetivamente vaza informações sobre transações fraudulentas futuras para o passado.

Quando esse modelo é implantado no mundo real e usado para prever transações futuras, ele terá um desempenho ruim porque a feature "dias desde a última fraude" não estará disponível durante a previsão. Ela foi construída usando informações futuras durante o processo de treinamento.

Como detectar se há data leakage no modelo?

Para detectar um data leakage, é necessário verificar alguns pontos importantes, tais como:

1. **Análise Exploratória de Dados Completa**: Explore seu conjunto de dados completamente e entenda as relações entre as diferentes variáveis. Procure por padrões ou features suspeitas que possam indicar um possível data leakage. Visualizações, análises de correlação e resumos estatísticos podem ser úteis nesse sentido.
2. **Conhecimento do Domínio e do Negócio**: Com conhecimento do negócio, é possível identificar features que possam introduzir vazamento de dado e que não devem ser incluídas no modelo.

3. **Discrepâncias de Desempenho na Validação Cruzada**: Treine seu modelo usando diferentes estratégias de validação cruzada e compare as métricas de desempenho. Se houver uma discrepância significativa entre o desempenho em diferentes folds de validação cruzada, isso pode indicar a presença de data leakage.
4. **Análise da Importância das Features**: Analise a importância ou contribuição das features do seu modelo. Se features indisponíveis durante a previsão mostrarem alta significância, isso indica data leakage.
5. **Validação Out of Time**: Se você estiver lidando com dados de séries temporais, realize uma validação out of time. Treine seu modelo em dados de um período específico e valide-o em dados de um período diferente. Isso pode ajudá-lo a identificar data leakage devido a fatores relacionados ao tempo.
6. **Inspeção do Processo de Coleta de Dados**: Revise o processo de coleta de dados para garantir que não houve erros ou inclusão não intencional de dados que não deveriam estar disponíveis durante a previsão. Verifique se há vazamentos potenciais de informações futuras nos dados passados.

7. **Correlação com Vazamento de Alvo**: Procure por features que se correlacionam altamente com a variável alvo, mas que não deveriam estar disponíveis durante a previsão. Tais features podem indicar vazamento da variável alvo..
8. **Identificação de Informações Sensíveis**: Verifique se há informações sensíveis no conjunto de dados que poderiam levar a overfitting ou vazamento de dados não intencional.
9. **Comportamento do Modelo no Conjunto de Teste/Validação**: Analise as previsões do modelo no conjunto de teste ou validação. Se ele tiver um desempenho significativamente melhor do que o esperado com base na complexidade do problema, pode ser um sinal de data leakage.
10. **Feedbacks e Revisão**: Busque feedback de colegas. Perspectivas novas podem frequentemente ajudar a identificar possíveis problemas de data leakage que podem ter sido ignorados.

Como pode prevenir o data leakage?

A prevenção do data leakage é crucial para garantir a confiabilidade e a precisão dos modelos de machine learning. Uma boa conduta para evitá-la é a utilização de validação cruzada e utilização de um conjunto de validação.
Nesta seção, veremos algumas técnicas e práticas recomendadas para minimizar o risco de data leakage, garantindo uma avaliação mais precisa do desempenho do modelo.

1. **Dividir Dados Antes de alguns Pré-processamento:** Certifique-se de dividir seus dados em conjuntos de treinamento e validação antes de realizar algumas etapas de pré-processamento. Por exemplo, quando fazemos scaling ou input de missing, é super importante que o split seja feito antes! Dessa forma, não há chance de informações do conjunto de validação vazarem para o conjunto de treinamento durante a engenharia de features ou a limpeza de dados.
2. **Evitar Vazamento de Dados na Engenharia de Features**: Tenha cuidado ao criar features para garantir que nenhuma informação do conjunto de validação seja usada na geração de features durante o treinamento. Qualquer informação usada na engenharia de features deve ser baseada exclusivamente nos dados de treinamento.
3. **Inspeção de Etapas de Pré-processamento**: Inspeccione cuidadosamente todas as etapas de pré-processamento e transformações para verificar que os dados de validação não os influenciem. Garanta que qualquer escalonamento, normalização ou imputação seja realizado com base apenas nos dados de treinamento.
4. **Realize a Preparação de Dados Dentro das Folds de Validação Cruzada**: Você pode facilmente vazar informações ao preparar seus dados para machine learning. O efeito é o overfitting dos seus dados de treinamento e uma avaliação excessivamente otimista do desempenho do seu modelo em dados não vistos. Por exemplo, se você normalizar ou padronizar todo o seu conjunto de dados e, em seguida, estimar o desempenho do seu modelo usando validação cruzada, você cometeu data leakage. Uma avaliação não vazada de algoritmos de machine learning nessa situação calcularia os parâmetros para redimensionar os dados dentro de cada fold da validação cruzada e usaria esses parâmetros para preparar os dados no conjunto de teste retido em cada ciclo. Para isso, uma boa dica é usar pipelines.

5. **Reserve um Conjunto de Dados de Validação:** Outra abordagem, talvez mais simples, é dividir seu conjunto de dados de treinamento em conjuntos de treino e validação, e guardar o conjunto de validação. Uma vez que você tenha concluído seu processo de modelagem e criado seu modelo final, avalie-o no conjunto de dados de validação. Isso pode fornecer uma verificação de sanidade para ver se sua estimativa de desempenho foi excessivamente otimista e houve vazamento de dados.
6. **Validação Cruzada Temporal**: Para dados de séries temporais ou quaisquer dados com uma ordenação temporal, use técnicas como Validação Cruzada de Séries Temporais. Nessa abordagem, você divide os dados com base no tempo, garantindo que o conjunto de validação contenha dados de um período posterior ao do conjunto de treinamento. Isso impede que o modelo aprenda com dados futuros durante o treinamento.
7. **Validação Cruzada Consciente de Grupos**: Se o seu conjunto de dados contém grupos ou clusters, você deve usar Validação Cruzada Consciente de Grupos, como GroupKFold ou StratifiedGroupKFold. Isso garante que todas as amostras do mesmo grupo permaneçam juntas no conjunto de treinamento ou validação, evitando o vazamento de dados entre os grupos.
8. **Validação Cruzada Aninhada**: Ao realizar ajustes de hiperparâmetros ou seleção de modelos, use Validação Cruzada Aninhada. Essa abordagem adiciona um loop externo de validação cruzada para lidar com o processo de seleção de modelos, enquanto o loop interno lida com o ajuste de hiperparâmetros. Isso garante que não haja vazamento de dados na seleção do melhor modelo ou hiperparâmetros.
9. **Corte Temporal**: Remova todos os dados imediatamente antes do evento de interesse, focando no momento em que você soube de um fato ou observação, em vez do momento em que a observação ocorreu.

Lista Conceitual

Questões:

[Cientista de dados] [Júnior em diante]

1. O que é aprendizado de máquina supervisionado?
2. Mencione três maneiras de tornar seu modelo robusto a outliers.
3. Mencione três maneiras de lidar com dados ausentes ou corrompidos em um conjunto de dados.
4. Você está construindo um classificador binário e descobriu que os dados estão desbalanceados, o que você deve fazer para lidar com essa situação?

5. Como verificamos se uma variável segue a distribuição normal?
6. O que é overfitting (sobreajuste)?
7. Como validar seus modelos?
8. Por que precisamos dividir nossos dados em três partes: treino, validação e teste?
9. Defina o processo de validação cruzada e a motivação por trás do uso dela.
10. Como escolhemos K na validação cruzada K-fold?
11. Devemos fazer o scaler das nossas features antes ou depois do split? Por quê?

Responda as perguntas abaixo em até 20 minutos e tente simular uma situação de entrevista real.

12. Suponha que você tenha um conjunto de dados com 30 dias de endereços IP, IDs de dispositivos e endereços de e-mail. Como você classificaria um usuário da Amazon como fraudador ou não, dado que você não tem rótulos?

Dicas

Para abordar esta questão do caso de produto, estruture a resposta da seguinte maneira:

- Faça perguntas de esclarecimento - Entenda que tipo de fraudadores a Amazon está tentando prever.
- Sobre os dados - Como você aproveitaria as variáveis de identificação - endereço IP, ID do dispositivo e endereço de e-mail - para resolver o problema de fraude?
- Metodologia de aprendizado de máquina - Como você projetaria um sistema que detecta fraudadores?

13. Como você estimaria o salário de uma ocupação profissional no LinkedIn?

Observação: Neste caso, estamos projetando o sistema do zero. Então, você tem que descobrir como coletar os dados e estruturá-los.

Respostas Lista Conceitual

1. **O que é aprendizado de máquina supervisionado?**

A aprendizagem supervisionada é um tipo de aprendizagem de máquina em que nossos algoritmos são treinados usando dados de treinamento bem rotulados, e as máquinas preveem a saída com base nesses dados. Dados rotulados indicam que os dados de entrada já foram marcados com a saída apropriada. Basicamente, é a tarefa de aprender uma função que mapeia o conjunto de entrada e retorna uma saída. Alguns exemplos são: Regressão Linear, Regressão Logística, KNN, etc.

2. **Mencione três maneiras de tornar seu modelo robusto a outliers.**

Abaixo veremos algumas condutas importantes:

- Investigar os outliers é sempre o primeiro passo para entender como tratá-los. Após compreender a natureza do motivo pelo qual os outliers ocorreram, você pode aplicar um dos vários métodos mencionados abaixo.
- Adicionar regularização que reduzirá a variância, por exemplo, regularização L1 ou L2.
- Utilizar modelos baseados em árvores (random forest, gradient boosting) que geralmente são menos afetados por outliers.
- Winsorizar os dados. Winsorização é a transformação de estatísticas limitando valores extremos nos dados estatísticos para reduzir o efeito de outliers possivelmente espúrios. Em dados numéricos, se a distribuição for quase normal, usando o escore Z, podemos detectar os outliers e tratá-los removendo-os ou limitando-os com algum valor. Se a distribuição for assimétrica, usando o IQR, podemos detectar e tratar removendo-os ou limitando-os com algum valor. Em dados categóricos, verifique a contagem de valores em porcentagem; se tivermos muito poucos registros de alguma categoria, podemos removê-los ou limitá-los com algum valor categórico como "outros".
- Transformar os dados, por exemplo, fazer uma transformação logarítmica quando a variável de resposta segue uma distribuição exponencial ou é assimétrica à direita.

- Utilizar métricas de erro mais robustas, como MAE ou perda de Huber, em vez de MSE.
- Remover os outliers, somente faça isso se você tiver certeza de que os outliers são verdadeiras anomalias que não valem a pena adicionar ao seu modelo. Isso deve ser considerado por último, pois removê-los significa perder informações.

3. **Mencione três maneiras de lidar com dados ausentes ou corrompidos em um conjunto de dados.**

Em geral, os dados do mundo real frequentemente apresentam muitos valores ausentes. A causa dos valores ausentes pode ser corrupção de dados ou falha ao registrar os dados. O tratamento de dados ausentes é muito importante durante a pré-processamento do conjunto de dados, pois muitos algoritmos de aprendizado de máquina não suportam valores ausentes. No entanto, você deve começar perguntando ao proprietário/stakeholder dos dados sobre os dados ausentes ou corrompidos. Pode ser no nível de entrada de dados, devido à codificação de arquivo, etc., que, se alinhada, pode ser tratada sem a necessidade de usar técnicas avançadas.

Existem diferentes maneiras de lidar com dados ausentes, e discutiremos apenas três delas:

Excluindo a linha com valores ausentes

O primeiro método para lidar com valores ausentes é excluir as linhas ou colunas que têm valores nulos. Este é um método fácil e rápido e leva a um modelo robusto, no entanto, levará à perda de muitas informações dependendo da quantidade de dados ausentes e só pode ser aplicado se os dados ausentes representarem uma pequena porcentagem do conjunto de dados inteiro.

Usando algoritmos de aprendizado que suportam valores ausentes

Alguns algoritmos de aprendizado de máquina são robustos a valores ausentes no conjunto de dados. O algoritmo K-NN pode ignorar uma coluna de uma medida de distância quando há valores ausentes. O Naive Bayes

também pode suportar valores ausentes ao fazer uma previsão. Outro algoritmo que pode lidar com um conjunto de dados com valores ausentes ou nulos é o modelo de floresta aleatória e Xgboost, pois pode trabalhar com dados não lineares e categóricos. O problema com este método é que a implementação desses modelos na biblioteca scikit-learn não suporta o tratamento de valores ausentes, então você terá que implementá-lo você mesmo.

Imputação de valores ausentes

Imputação de dados significa a substituição de valores estimados para dados ausentes ou inconsistentes em seu conjunto de dados. Existem diferentes maneiras de estimar os valores que substituirão o valor ausente. O mais simples é substituir o valor ausente pelo valor mais repetido na linha ou na coluna. Outra maneira simples é substituí-lo pela média, mediana ou moda do restante da linha ou coluna.

A vantagem disso é que é uma maneira fácil e rápida de lidar com os dados ausentes, mas pode levar a vazamento de dados e não considera a covariância entre os recursos. Uma maneira melhor é usar um modelo de aprendizado de máquina para aprender o padrão entre os dados e prever os valores ausentes, este é um método muito bom para estimar os valores ausentes que não levarão a vazamento de dados e levarão em consideração a covariância entre os recursos, a desvantagem deste método é a complexidade computacional, especialmente se seu conjunto de dados for grande.

4. **Você está construindo um classificador binário e descobriu que os dados estão desbalanceados, o que você deve fazer para lidar com essa situação?**

Se houver um desequilíbrio nos dados, existem várias medidas que podemos tomar para treinar um classificador binário mais justo:

1. Pré-processamento:

- Verifique se é possível obter mais dados.

- Use técnicas de amostragem (sobre-amostragem da classe minoritária e subamostragem da classe majoritária, também pode-se adotar uma abordagem híbrida). Também podemos usar técnicas de aumento de dados para adicionar mais pontos de dados para a classe minoritária, mas com pequenas variações/mudanças, resultando em novos pontos de dados que são semelhantes aos que foram derivados. A técnica mais comum/popular é o SMOTE (Synthetic Minority Oversampling Technique).
- Supressão: Embora não seja recomendado, podemos descartar algumas características diretamente responsáveis pelo desequilíbrio.
- Aprendizado de Representação Justa: Projetar os exemplos de treinamento para um subespaço ou plano minimiza o desequilíbrio nos dados.
- Reponderação: Podemos atribuir pesos a cada exemplo de treinamento para reduzir o desequilíbrio nos dados.

2. Processamento em andamento:

- Regularização: Podemos adicionar termos de pontuação que medem o desequilíbrio nos dados na função de perda e, portanto, minimizar a função de perda também minimizará o grau de desequilíbrio em relação à pontuação escolhida, o que também minimiza indiretamente outras métricas que medem o grau de desequilíbrio nos dados.
- Debiasing adversarial: Aqui usamos a noção adversarial para treinar o modelo, onde o discriminador tenta detectar se há sinais de desequilíbrio nos dados previstos pelo gerador e, portanto, o gerador aprende a gerar dados que são menos propensos ao desequilíbrio.

3. Pós-processamento:

- Equalização de odds: Aqui tentamos igualar as probabilidades para as classes em relação ao desequilíbrio nos dados para corrigir o desequilíbrio no modelo treinado. Normalmente, o escore F1 é uma boa escolha, se tanto a precisão quanto a recuperação forem importantes.
- Escolha de métricas de desempenho adequadas: Por exemplo, a precisão não é uma métrica correta a ser usada quando as classes estão

desequilibradas. Em vez disso, use precisão, recall, escore F1 e curva ROC.

5. **Como verificamos se uma variável segue a distribuição normal?**

Trace um histograma a partir dos dados amostrados. Se você conseguir ajustar a curva em forma de sino "normal" ao histograma, então a hipótese de que a variável aleatória subjacente segue a distribuição normal não pode ser rejeitada.

Verifique a assimetria (skewness) e a curtose (kurtosis) dos dados amostrados. Assimetria = 0 e curtose = 3 são típicos de uma distribuição normal, então quanto mais distantes esses valores estiverem de 0 e 3, respectivamente, mais não normal será a distribuição.

Use os testes de Kolmogorov-Smirnov e/ou Shapiro-Wilk para normalidade. Eles levam em conta tanto a assimetria quanto a curtose simultaneamente.

Verifique o gráfico Quantil-Quantil. É um gráfico de dispersão criado plotando dois conjuntos de quantis um contra o outro. O gráfico Q-Q normal coloca os pontos de dados em uma linha aproximadamente reta.

6. **O que é overfitting (sobreajuste)?**

Quando o seu modelo apresenta um desempenho muito bom no conjunto de treinamento, mas não consegue generalizar para o conjunto de teste, isso geralmente indica um problema de sobreajuste (overfitting). O sobreajuste ocorre quando o modelo se ajusta excessivamente aos detalhes e ao ruído nos dados de treinamento, em vez de aprender os padrões subjacentes que podem ser generalizados para novos dados.

7. **Como validar seus modelos?**

Uma das abordagens mais comuns é dividir os dados em partes de treinamento, validação e teste. Os modelos são treinados nos dados de treinamento, os hiperparâmetros (por exemplo, parada precoce) são selecionados com base nos dados de validação e a medição final é feita no

conjunto de dados de teste. Outra abordagem é a validação cruzada: dividir o conjunto de dados em K partes e, a cada vez, treinar modelos nas partes de treinamento e medir o desempenho nas partes de validação. Você também pode combinar essas abordagens: criar um conjunto de dados de teste/holdout e fazer validação cruzada com o restante dos dados. A qualidade final é medida no conjunto de dados de teste.

8. **Por que precisamos dividir nossos dados em três partes: treino, validação e teste?**

O conjunto de treinamento é usado para ajustar o modelo, ou seja, treinar o modelo com os dados. O conjunto de validação é então usado para fornecer uma avaliação imparcial de um modelo enquanto se ajustam os hiperparâmetros. Isso melhora a generalização do modelo. Finalmente, um conjunto de dados de teste, que o modelo nunca "viu" antes, deve ser usado para a avaliação final do modelo. Isso permite uma avaliação imparcial do modelo. A avaliação nunca deve ser realizada nos mesmos dados usados para treinamento. Caso contrário, o desempenho do modelo não seria representativo.

9. **Defina o processo de validação cruzada e a motivação por trás do uso dela.**

A validação cruzada é uma técnica usada para avaliar o desempenho de um modelo de aprendizado em vários subconjuntos dos dados de treinamento. Em geral, dividimos os dados em conjuntos de treinamento e teste, onde usamos os dados de treinamento para treinar nosso modelo e os dados de teste para avaliar o desempenho do modelo em dados não vistos, e o conjunto de validação para escolher os melhores hiperparâmetros.

Agora, uma divisão aleatória na maioria dos casos (para conjuntos de dados grandes) é suficiente. No entanto, para conjuntos de dados menores, ela é suscetível à perda de informações importantes presentes nos dados nos quais não foi treinada. Portanto, a validação cruzada, apesar de ser computacionalmente cara, combate esse problema.

O processo de validação cruzada é o seguinte:

- Definir k ou o número de dobras.
- Embaralhar aleatoriamente os dados em k blocos de tamanhos iguais (dobras).
- Para cada i na dobra 1 a k, treinar os dados usando todas as dobras, exceto a dobra i, e testar na dobra i.
- Média dos erros de validação/teste K do passo anterior para obter uma estimativa do erro.

Esse processo tem como objetivo alcançar o seguinte:

- Prevenir o sobreajuste durante o treinamento, evitando treinar e testar no mesmo subconjunto de pontos de dados.
- Evitar a perda de informações usando apenas um subconjunto dos dados para validação. Isso é importante para conjuntos de dados pequenos.

A validação cruzada é sempre boa de ser usada para conjuntos de dados pequenos e, se usada para conjuntos de dados grandes, a complexidade computacional aumentará dependendo do número de dobras.

10. Como escolhemos K na validação cruzada K-fold?

Existem duas coisas a considerar ao decidir sobre K: o número de modelos que obtemos e o tamanho do conjunto de validação. Não queremos que o número de modelos seja muito pequeno, como 2 ou 3. Pelo menos 4 modelos fornecem uma decisão menos tendenciosa sobre as métricas. Por outro lado, queremos que o conjunto de dados seja pelo menos 20-25% do total. Para que seja mantida pelo menos uma proporção de 3:1 entre conjunto de treinamento e validação.

11. Devemos fazer alguns pre-processamentos depois do split, para evitar data leakage. No caso do scaler, se não fizermos o split antes, dados do teste farão parte da estatística do scaler - portanto, poderá haver data leakage.

12. **Suponha que você tenha um conjunto de dados com 30 dias de endereços IP, IDs de dispositivos e endereços de e-mail. Como você classificaria um usuário da Amazon como fraudador ou não, dado que você não tem rótulos?**

Comentário: Outras empresas FAANG, como o Facebook, adoram fazer perguntas sobre a detecção de fraudadores e spammers. Espere uma pergunta semelhante sobre a previsão de spammers ou qualquer outro usuário que viole a política de um site em geral. A estrutura fornecida na solução pode ser replicada em outros problemas.

Candidato: Posso começar perguntando se os fraudadores que precisam ser detectados estão entre os vendedores ou consumidores? É a minha experiência que os comportamentos são bastante diferentes, dependendo se o fraudador está do lado do vendedor ou do consumidor.

Entrevistador: Boa pergunta. Vamos dizer que o tipo de fraudadores que queremos detectar são consumidores.

Candidato: Ótimo. Tenho uma pergunta de acompanhamento. Pode haver vários tipos de atividades fraudulentas do lado do consumidor. Posso assumir que são fraudadores que cometem estornos? Um exemplo poderia ser

fraudadores que compraram e receberam mercadorias e depois alegam que não fizeram a transação para receber um reembolso?

Comentário: O candidato demonstra conhecimento de domínio ao entender que nem todos os fraudadores são iguais. Ela reconhece que existem diferentes tipos de fraudadores e, dependendo do tipo de fraudador que está sendo alvo, a estratégia de aprendizado de máquina seria diferente.

Entrevistador: Sim, vamos supor que seja o caso.

Candidato: Ótimo, obrigado pelo esclarecimento. Tenho outra pergunta de acompanhamento, que é sobre os próprios dados. É justo assumir que um usuário pode ter vários endereços IP, IDs de dispositivos e endereços de e-mail?

Entrevistador: Sim.

Candidato: Ótimo. Acho que posso começar a idealizar uma solução potencial. É meu entendimento que um fraudador criaria várias contas usando uma combinação de diferentes endereços IP, IDs de dispositivos e endereços de e-mail.

Um usuário normal provavelmente teria um endereço de e-mail vinculado a um único endereço IP, que geralmente é sua internet doméstica. No entanto, um fraudador pode ter vários endereços de e-mail vinculados a um endereço IP porque está tentando estabelecer várias contas.

Ter várias contas permitiria que ele continuasse cometendo fraudes de estorno mesmo se uma das contas fosse banida.

Entrevistador: Entendo, qual é a sua sugestão então?

Candidato: Bem, precisamos derivar um conjunto de features que conte os endereços IP distintos por endereço de e-mail nos últimos 30 dias. Podemos usar essa estratégia para outras combinações de variáveis de identificação. Aqui está uma amostra de features que podem ser criadas:

- Número de endereços de e-mail distintos por endereço IP nos últimos 7 dias

- Número de endereços de e-mail distintos por endereço IP nos últimos 14 dias
- Número de endereços de e-mail distintos por endereço IP nos últimos 30 dias
- Número de IDs de dispositivos distintos por endereço de e-mail nos últimos 7 dias
- ...

Comentário: Observe que derivar features baseadas em contagem usando variáveis de identificação é a melhor prática ao criar um modelo que detecta fraudadores.

Entrevistador: Ótimo, agora como você usaria essas features para detectar fraudadores?

Candidato: Posso pensar em duas abordagens potenciais. A primeira abordagem é baseada em regras, onde analisamos a distribuição de probabilidade por feature. Aplicamos um limite no percentil 99. Um usuário com, digamos, 7 endereços de e-mail distintos vinculados a um único endereço IP nos últimos 7 dias é considerado suspeito.

Aplicamos o limite em todas as variáveis para derivar um subconjunto de usuários que são os mais suspeitos. Obviamente, isso não é baseado em modelo, mas deve fornecer um bom ponto de partida para rotular potenciais fraudadores.

Agora, precisamos verificar esses rótulos, então enviamos o subconjunto para a equipe de revisão para verificação do rótulo. Isso nos ajudará a identificar a precisão do sistema.

Entrevistador: Ótima sugestão. Qual é a outra ideia?

Candidato: A outra ideia usa um modelo de aprendizado profundo. Um autoencoder é um algoritmo não supervisionado que ajuda a sinalizar outliers, que é o que queremos fazer no conjunto de features. Podemos usar o erro de reconstrução (medido em MSE) para identificar usuários com altas pontuações de outliers. Podemos repetir o procedimento de verificação manual desses usuários, enviando-os para a equipe de revisão.

13. Como você estimaria o salário de uma ocupação profissional no LinkedIn?

Solução

Candidato: Primeiro, gostaria de delimitar o problema, pois ele não segue o problema de regressão ou classificação tradicional. Posso assumir que já tenho acesso a dados salariais da plataforma?

Entrevistador: Neste caso, estamos projetando o sistema do zero. Então, você tem que descobrir como coletar os dados e estruturá-los.

Candidato: Entendi. Preciso adquirir as informações salariais em algum lugar. Uma possível fonte são os dados econômicos do governo. Tenho certeza de que o Bureau of Labor Statistics (BLS) conteria essas informações salariais que poderiam ser fornecidas aos usuários.

Entrevistador: Ok, e se a ocupação não estiver na lista, como você derivaria a estimativa?

Candidato: Acredito que este é o momento em que uma pesquisa poderia ser usada. É comum coletar essas informações diretamente dos usuários quando não podem ser inferidas indiretamente. Mas, geralmente, as pesquisas são ignoradas pelos usuários. Portanto, um incentivo é necessário para que respondam a pesquisas sobre sua ocupação e informações salariais.

Entrevistador: Entendi. Como você forneceria a estimativa salarial de uma ocupação no nível geográfico?

Candidato: Eu diria que isso pode ser inferido com base na localização da empresa em que o respondente trabalha ou em um campo de texto na pesquisa.

Entrevistador: Suponha que o salário de um cientista de dados seja conhecido em Nova York, mas não em Londres, como você faria a estimativa então?

Candidato: Hmm... Eu diria que esta é uma situação em que um modelo de regressão é necessário. Basicamente, prever a distribuição salarial

desconhecida com base nas distribuições conhecidas. Eu poderia usar features provenientes de dados econômicos regionais (por exemplo, custo de vida, população, salário médio), dados de ocupação (por exemplo, estimativas salariais dos conhecidos) e dados da empresa (por exemplo, indústria, setor, indicadores econômicos). Os dados de entrada podem ser treinados em um modelo de regressão, seja um modelo linear ou floresta aleatória, para prever o desconhecido.

Entrevistador: Ok, como você lidaria com outliers nas respostas da pesquisa?

Candidato: Uma abordagem simples para lidar com outliers deve ser suficiente. Eu usaria o método IQR para remover outliers que são maiores que 1,5 ou 2 vezes o percentil 75. Isso deve remover informações de estimativa salarial incomuns, como $1M para uma posição de cientista de dados de nível inicial.

Entrevistador: Ótimo, então como você forneceria as estimativas ao usuário?

Candidato: Acredito que estatísticas simples, como a mediana, o percentil 10 e o percentil 90, devem ser suficientes.