Vocabulário de Termos Técnicos

O presente documento tem por objetivo auxiliar na compreensão dos exercícios propostos, principalmente para os participantes iniciantes na área de ciência de dados. Esse documento complementar tem por finalidade reunir algumas definições dos termos técnicos utilizados nos enunciados, porém NÃO é de utilização obrigatória. Cada participante poderá complementar seus estudos da forma que preferir.

Fonte: Chat GPT (https://chat.openai.com)

ABT (*Analytical Base Table*)

ABT (*Analytical Base Table*), que pode ser traduzido como "Tabela de Base Analítica", é uma tabela especializada usada em ciência de dados e análise de dados para armazenar os dados preparados e tratados que servirão como base para a construção e desenvolvimento de modelos preditivos e análises mais complexas.

A principal diferença entre uma ABT e uma tabela de dados bruta é que a ABT é projetada e preparada de forma específica para atender às necessidades de análise e modelagem. A ABT passa por um processo de pré-processamento e tratamento de dados, onde são realizadas as seguintes etapas:

1. **Limpeza de dados:** Eliminação de dados faltantes, correção de erros ou valores inconsistentes e remoção de outliers.

2. **Seleção de variáveis:** Escolha das variáveis relevantes para a análise ou para a tarefa de modelagem. Nem todas as variáveis presentes nos dados brutos são necessárias para a análise ou para o modelo.

3. ***Encoding* de variáveis categóricas:** Conversão de variáveis categóricas em representações numéricas adequadas para uso em modelos de aprendizado de máquina.

4. ***Feature engineering:*** Criação de novas variáveis ou transformações de variáveis existentes para melhorar a capacidade de previsão do modelo.

5. **Balanceamento de classes (opcional):** Em problemas de classificação com classes desbalanceadas, pode-se aplicar técnicas de balanceamento para garantir uma distribuição mais equilibrada das classes na ABT.

6. **Divisão em conjuntos de treinamento e teste:** Separação dos dados em conjuntos de treinamento e teste, conforme discutido anteriormente, para avaliar o desempenho do modelo de forma justa.

7. **Normalização ou padronização (opcional):** Em alguns modelos, é necessário normalizar ou padronizar as variáveis para garantir que elas estejam em escalas comparáveis.

A ABT, após passar por essas etapas, fornece um conjunto de dados preparado e tratado que está pronto para ser usado na construção e treinamento de modelos preditivos ou para análises mais aprofundadas.

Em resumo, uma ABT é uma tabela de dados preparada e tratada, otimizada para uso em análises e modelos preditivos, diferentemente de uma "tabela normal" ou tabela de dados brutos, que pode conter ruído, valores faltantes e outras irregularidades que precisam ser tratadas antes de serem usadas em análises ou modelos.

Acurácia

A acurácia de um modelo é uma métrica que representa a proporção de previsões corretas feitas pelo modelo em relação ao número total de previsões. Em outras palavras, é a medida de quão preciso o modelo é ao fazer previsões em relação ao conjunto de dados de teste.

Matematicamente, a acurácia pode ser calculada pela seguinte fórmula:

Acurácia = (Número de previsões corretas) / (Número total de previsões)

Essa métrica é comumente usada em problemas de classificação, onde o modelo atribui rótulos ou classes a diferentes exemplos. A acurácia é uma medida geral de desempenho do modelo, mostrando o quão bem ele classifica corretamente os exemplos em todas as classes.

A acurácia é uma métrica útil quando o conjunto de dados possui classes balanceadas, ou seja, cada classe tem aproximadamente o mesmo número de exemplos. No entanto, em problemas com classes desbalanceadas, onde uma classe tem muito mais exemplos do que outra, a acurácia pode ser enganosa e não fornecer uma visão completa do desempenho do modelo. Isso ocorre porque um modelo pode alcançar alta acurácia simplesmente classificando a maioria dos exemplos como pertencentes à classe majoritária.

Portanto, em problemas desbalanceados, é importante considerar outras métricas, como a precisão, o *recall* (sensibilidade) ou a *F1-score*, que levam em conta tanto as previsões corretas como as previsões incorretas, especialmente para a classe minoritária.

Em resumo, a acurácia de um modelo é a proporção de previsões corretas em relação ao número total de previsões, sendo uma medida útil para avaliar o desempenho geral do modelo em problemas de classificação com classes balanceadas, mas é importante considerar outras métricas em problemas com classes desbalanceadas.

Balanceamento

Balancear um *dataframe* é importante quando estamos lidando com problemas de aprendizado de máquina onde as classes de interesse não estão representadas de forma equilibrada nos dados. Isso ocorre quando uma ou algumas classes têm um número significativamente maior de exemplos do que outras. Em situações de desequilíbrio de classes, é comum haver uma classe minoritária que possui poucos exemplos, enquanto a classe majoritária tem uma quantidade muito maior.

A seguir estão algumas razões pelas quais é importante balancear um *dataframe*:

1. **Evitar viés no modelo:** Se o conjunto de dados estiver desequilibrado, o modelo pode ser tendencioso em relação à classe majoritária. Isso pode levar o modelo a ter um desempenho ruim na previsão da classe minoritária, que é a classe de interesse.

2. **Melhorar a precisão e desempenho:** Ao balancear o *dataframe*, garantimos que o modelo tenha exemplos suficientes para aprender a reconhecer padrões e relações nas classes minoritárias. Isso pode melhorar a precisão geral do modelo e seu desempenho em dados não vistos.

3. **Reduzir o impacto de classes dominantes:** Em problemas desbalanceados, o modelo pode ser excessivamente influenciado pela classe majoritária e pode acabar ignorando ou não aprendendo adequadamente as características das classes minoritárias. Balancear o *dataframe* ajuda a mitigar esse problema.

4. **Prevenir *overfitting*:** Em conjuntos de dados desequilibrados, o modelo pode ter dificuldade em generalizar para dados desconhecidos, levando a um *overfitting* nos dados de treinamento. Ao balancear os dados, podemos reduzir a tendência do modelo a se ajustar em excesso aos dados de treinamento.

Existem várias técnicas para balancear um *dataframe*, como *undersampling* (reduzindo a quantidade de exemplos da classe majoritária), *oversampling* (aumentando a quantidade de exemplos da classe minoritária) e o uso de técnicas avançadas, como SMOTE (*Synthetic Minority Over-sampling Technique*), que gera exemplos sintéticos para a classe minoritária.

Em resumo, balancear um *dataframe* é importante para garantir que o modelo aprenda adequadamente as características de todas as classes e evite viés em direção às classes majoritárias, resultando em um modelo mais preciso, justo e confiável para problemas de aprendizado de máquina com classes desbalanceadas.

Dummies ou **Variáveis Fictícias** ou **Variáveis Binárias** *versus* ***One Hot Encoding***

"*Dummies*" e "*One-Hot Encoding*" são técnicas de pré-processamento de dados frequentemente usadas para lidar com variáveis categóricas em modelos de aprendizado de máquina.

1. **Dummies:**

   "*Dummies*" é uma abreviação para "variáveis fictícias" ou "variáveis binárias". Essa técnica envolve a conversão de uma variável categórica em uma representação numérica usando 0s e 1s. Basicamente, cria-se uma nova coluna (ou variável *dummy*) para cada categoria única presente na variável original. Se uma observação tiver uma determinada categoria, a coluna *dummy* correspondente receberá o valor 1, caso contrário, receberá 0.

   Por exemplo, suponha que temos uma coluna categórica chamada "Cor" com três categorias: "Vermelho", "Verde" e "Azul". Usando a técnica de *dummies*, criaríamos três novas colunas: "Cor_Vermelho", "Cor_Verde" e "Cor_Azul". Se um registro tiver a cor "Vermelho", a coluna "Cor_Vermelho" será 1 e as outras duas colunas serão 0.

2. ***One-Hot Encoding*:**

   O "*One-Hot Encoding*" é uma variação do método de dummies. Nessa técnica, também criamos novas colunas (dummies) para cada categoria única, mas a principal diferença é que apenas uma dessas colunas recebe o valor 1 para cada registro, enquanto todas as outras recebem 0s. Isso significa que o valor é codificado de forma "*one-hot*", com uma única posição sendo "quente" (valor 1) e todas as outras "frias" (valor 0).

   Usando o mesmo exemplo da coluna "Cor" com as categorias "Vermelho", "Verde" e "Azul", com o *One-Hot Encoding*, criaríamos as colunas "Cor_Vermelho", "Cor_Verde" e "Cor_Azul". Se um registro tiver a cor "Verde", a coluna "Cor_Verde" terá valor 1 e as outras duas colunas terão valor 0.

3. **Diferença entre *Dummies* e *One-Hot Encoding*:**

   A principal diferença entre essas duas técnicas está na forma como são codificadas as variáveis categóricas com múltiplas categorias. No método de *dummies*, várias colunas são criadas para representar as categorias, e várias colunas podem ter valor 1 para uma única observação. Já no *One-Hot Encoding*, apenas uma coluna é definida como 1 para cada observação, enquanto todas as outras são 0. Isso torna o *One-Hot Encoding* mais eficiente em termos de memória e pode ser preferível em modelos de aprendizado de máquina com muitas categorias únicas.

Ambas as técnicas são amplamente utilizadas na preparação de dados para modelos de aprendizado de máquina, e a escolha entre elas dependerá das características específicas do conjunto de dados e do algoritmo de aprendizado a ser utilizado.

Especificidade

A especificidade é uma métrica de avaliação usada em problemas de classificação, especialmente em casos onde o desbalanceamento das classes é relevante. Essa métrica mede a capacidade de um modelo em identificar corretamente os exemplos pertencentes à classe negativa (classe majoritária, por exemplo), em relação ao total de exemplos reais da classe negativa.

Matematicamente, a especificidade é calculada pela seguinte fórmula:

> Especificidade = Número de verdadeiros negativos (TN) / (Número de verdadeiros negativos (TN) + Número de falsos positivos (FP))

Onde:

- TN (*True Negatives*) é o número de exemplos corretamente classificados como pertencentes à classe negativa pelo modelo.

- FP (*False Positives*) é o número de exemplos que pertencem à classe negativa, mas foram incorretamente classificados como pertencentes à classe positiva pelo modelo.

A especificidade é especialmente útil quando estamos mais interessados em evitar falsos positivos, ou seja, quando queremos que o modelo minimize as previsões errôneas de exemplos negativos como positivos. Isso é comum em problemas onde os custos ou consequências dos falsos positivos são altos.

Uma alta especificidade indica que o modelo está sendo capaz de identificar corretamente a classe negativa, mas isso pode estar associado a uma maior taxa de falsos negativos (exemplos da classe positiva erroneamente classificados como negativos). Portanto, é importante encontrar um equilíbrio entre a especificidade e outras métricas, como a sensibilidade (*recall*) ou a precisão, dependendo das necessidades e características específicas do problema em questão.

Em resumo, a especificidade é uma métrica que mede a capacidade de um modelo em identificar corretamente os exemplos da classe negativa, sendo especialmente relevante em problemas onde o foco está em evitar falsos positivos e maximizar a detecção correta dos exemplos negativos.

F1-score

O *F1-score* (também conhecido como *F-score* ou *F-measure*) é uma métrica de avaliação de modelos de aprendizado de máquina, especialmente em problemas de classificação, que combina a precisão (*precision*) e o *recall* (sensibilidade) em uma única medida. O *F1-score* é útil quando temos um conjunto de dados desbalanceado, ou seja, quando uma classe tem muito mais exemplos do que a outra.

O *F1-score* é calculado a partir da média harmônica da precisão e do *recall* e é dado pela seguinte fórmula:

F1-score = 2 * (Precisão * Recall) / (Precisão + Recall)

A precisão é a proporção de exemplos positivos corretamente classificados em relação ao total de exemplos classificados como positivos. Ela mede a precisão das previsões positivas do modelo.

O *recall* é a proporção de exemplos positivos corretamente classificados em relação ao total de exemplos reais da classe positiva. Ele mede a capacidade do modelo de detectar corretamente os exemplos positivos.

Ao usar o *F1-score*, estamos considerando tanto a capacidade do modelo de fazer previsões corretas (precisão) quanto sua capacidade de detectar corretamente os exemplos positivos (*recall*). O *F1-score* é uma métrica útil quando queremos equilibrar a importância das duas métricas e quando não podemos privilegiar uma delas sobre a outra.

O *F1-score* varia de 0 a 1, onde 1 representa um modelo perfeito e 0 representa um modelo com desempenho muito ruim. Quanto maior o valor do *F1-score*, melhor o desempenho do modelo em termos de equilíbrio entre precisão e recall.

Em resumo, o *F1-score* é uma métrica que combina a precisão e o *recall* em uma única medida para avaliar o desempenho de modelos de classificação em problemas com classes desbalanceadas, fornecendo uma visão mais completa do desempenho geral do modelo.

Falso positivo / Falso negativo

Falsos positivos e falsos negativos são conceitos importantes em problemas de classificação e são usados para avaliar o desempenho de modelos de aprendizado de máquina em problemas de diagnóstico, detecção ou classificação.

1. **Falsos Positivos:**

   Um falso positivo ocorre quando o modelo prevê incorretamente que um exemplo pertence a uma determinada classe positiva, quando na verdade pertence à classe negativa. Em outras palavras, o modelo identifica erroneamente algo como pertencente à classe positiva quando, na verdade, não é.

   Um exemplo prático seria um teste médico que detecta uma doença em uma pessoa que, na verdade, não possui a doença. O falso positivo ocorre quando o teste erroneamente indica a presença da doença, levando a um resultado incorreto.

2. **Falsos Negativos:**

   Um falso negativo ocorre quando o modelo prevê incorretamente que um exemplo pertence à classe negativa, quando na verdade pertence à classe positiva. Nesse caso, o modelo falha em detectar corretamente a presença de algo que realmente existe.

   Usando o mesmo exemplo médico, um falso negativo ocorre quando o teste não detecta uma doença em uma pessoa que, de fato, tem a doença. Nesse caso, o teste falha em identificar corretamente a presença da doença.

Em resumo, falsos positivos são previsões incorretas de que algo pertence a uma classe positiva quando não pertence, enquanto falsos negativos são previsões incorretas de que algo pertence à classe negativa quando pertence à classe positiva. Ambos são indicadores críticos da capacidade de um modelo de realizar classificações precisas e são utilizados para avaliar o desempenho e ajustar modelos em problemas de classificação.

Filtragem de dados

Filtrar os *dataframes* antes de iniciar a construção de um modelo preditivo é importante por várias razões:

1. **Remoção de dados irrelevantes:** Muitas vezes, os conjuntos de dados contêm informações que não são relevantes para o problema específico de modelagem que estamos abordando. Ao filtrar os dados, podemos remover colunas ou linhas que não têm impacto ou não contribuem para a tarefa de previsão, tornando o conjunto de dados mais focado e mais adequado para o modelo que queremos construir.

2. **Redução de ruído:** Em alguns casos, os conjuntos de dados podem conter ruído ou dados inconsistentes que podem prejudicar a precisão do modelo. Filtrar os dados pode ajudar a remover valores discrepantes ou dados com erros, melhorando a qualidade dos dados e aumentando a eficácia do modelo.

3. **Redução da dimensão:** Alguns conjuntos de dados podem ser muito grandes, com muitas colunas ou atributos. Isso pode tornar a construção do modelo mais lenta e complexa, além de aumentar a probabilidade de *overfitting* (modelo muito ajustado aos dados de treinamento, mas com baixa capacidade de generalização). Ao filtrar e selecionar apenas as colunas relevantes, podemos reduzir a dimensão do conjunto de dados, facilitando o processo de modelagem.

4. **Melhoria do desempenho do modelo:** Filtrar os dados pode ajudar a eliminar problemas de multicolinearidade (alta correlação entre as variáveis preditoras) e outras situações que podem afetar negativamente o desempenho do modelo. Uma seleção adequada de variáveis pode melhorar a precisão e a interpretabilidade do modelo.

5. **Foco na tarefa:** Filtrar os dados permite que o modelo se concentre nas características mais relevantes para a tarefa específica em questão. Isso pode resultar em um modelo mais eficiente e com melhor capacidade de generalização para novos dados.

Em resumo, filtrar os *dataframes* antes de construir um modelo preditivo é uma etapa importante do pré-processamento de dados. Isso ajuda a melhorar a qualidade do conjunto de dados, a reduzir a dimensão do conjunto de atributos e a eliminar informações irrelevantes ou ruidosas, resultando em um modelo mais focado, preciso e eficiente.

Hierarquia de dados

É importante considerar a hierarquia dos valores ao trabalhar com colunas de código ou qualquer tipo de atributo que possua uma ordem específica ou relacionamento natural entre seus valores.

Quando falamos de "colunas de código", por exemplo, estamos nos referindo a atributos que usam números ou códigos para representar diferentes categorias ou

classes. Esses códigos podem ser usados para representar informações como níveis de importância, graus de satisfação, etapas de um processo, entre outros.

Aqui estão algumas razões pelas quais a hierarquia dos valores em colunas de código pode ser relevante:

1. **Interpretação correta do modelo:** Se os códigos têm uma ordem ou relação específica, é importante que o modelo entenda essa hierarquia corretamente. Caso contrário, o modelo pode fazer previsões incorretas ou não conseguir capturar corretamente a importância relativa das diferentes categorias.

2. **Algoritmos sensíveis à ordem:** Alguns algoritmos de aprendizado de máquina, como regressões lineares ou árvores de decisão, podem ser sensíveis à ordem dos valores em atributos numéricos. Isso significa que a ordem dos códigos pode afetar o desempenho desses algoritmos e a qualidade das previsões.

3. **Representação numérica especial:** Se os códigos são usados como representação numérica especial, como 0 e 1 para indicar "não" e "sim", é importante garantir que o modelo entenda corretamente o significado desses valores e não interprete erroneamente como uma relação de ordem.

4. **Análise de tendências ou padrões:** A hierarquia dos valores em colunas de código pode ser importante para identificar tendências ou padrões em análises exploratórias de dados. Por exemplo, analisar a evolução de códigos ao longo do tempo ou em diferentes contextos pode fornecer insights valiosos.

Para trabalhar corretamente com colunas de código, é fundamental que a hierarquia dos valores seja claramente definida e bem compreendida. É importante garantir que o modelo e os algoritmos utilizados sejam adequados para lidar com dados com hierarquia e que a representação numérica seja tratada de forma apropriada, para que as análises e previsões sejam precisas e confiáveis.

Matriz de Confusão

A matriz de confusão é uma ferramenta essencial para avaliar o desempenho de modelos de aprendizado de máquina em problemas de classificação. Ela é utilizada para visualizar e resumir as previsões feitas pelo modelo em relação aos valores reais das classes.

A matriz de confusão é uma tabela que organiza as previsões do modelo em quatro categorias diferentes:

1. **Verdadeiro Positivo (TP):** Representa os casos em que o modelo previu corretamente que um exemplo pertence à classe positiva e o valor real também é positivo.

2. **Verdadeiro Negativo (TN):** Representa os casos em que o modelo previu corretamente que um exemplo pertence à classe negativa e o valor real também é negativo.

3. **Falso Positivo (FP):** Representa os casos em que o modelo previu incorretamente que um exemplo pertence à classe positiva, mas o valor real é negativo.

4. **Falso Negativo (FN):** Representa os casos em que o modelo previu incorretamente que um exemplo pertence à classe negativa, mas o valor real é positivo.

A matriz de confusão geralmente é organizada em uma tabela com duas linhas (uma para cada classe real) e duas colunas (uma para cada classe prevista). A soma dos valores nas células da matriz representa o número total de exemplos no conjunto de teste.

A partir da matriz de confusão, podemos calcular várias métricas de avaliação do modelo, como:

1. **Precisão (*Precision*):** É a proporção de exemplos positivos corretamente classificados em relação ao total de exemplos classificados como positivos (TP / (TP + FP)).

2. **Sensibilidade ou Recall (*Recall*):** É a proporção de exemplos positivos corretamente classificados em relação ao total de exemplos reais da classe positiva (TP / (TP + FN)).

3. **Especificidade (*Specificity*):** É a proporção de exemplos negativos corretamente classificados em relação ao total de exemplos reais da classe negativa (TN / (TN + FP)).

4. ***F1-score:*** É uma medida que combina precisão e recall para avaliar o desempenho do modelo em ambas as classes.

A matriz de confusão é uma ferramenta poderosa para entender o comportamento do modelo e identificar seus pontos fortes e fracos. Ela nos ajuda a compreender o desempenho em diferentes classes e permite ajustar o modelo para melhorar suas previsões em problemas de classificação.

Multicolinearidade

A multicolinearidade é um conceito importante na análise de regressão e modelagem estatística. Refere-se à presença de alta correlação entre duas ou mais variáveis independentes (também conhecidas como "preditoras" ou "*features*") em um modelo estatístico.

Em outras palavras, a multicolinearidade ocorre quando duas ou mais variáveis independentes estão altamente correlacionadas, o que significa que elas têm uma relação linear forte entre si. Essa correlação forte pode ser problemática em alguns modelos, especialmente em regressão linear, porque pode dificultar a identificação e interpretação das relações individuais entre as variáveis

independentes e a variável dependente (também conhecida como "resposta" ou "*target*").

Existem dois tipos principais de multicolinearidade:

1. **Multicolinearidade perfeita:** Ocorre quando duas ou mais variáveis independentes estão linearmente relacionadas de forma exata. Nesse caso, uma das variáveis pode ser expressa como uma combinação linear das outras, tornando impossível separar seus efeitos individuais no modelo.

2. **Multicolinearidade aproximada:** Ocorre quando duas ou mais variáveis independentes têm alta correlação, mas não são linearmente relacionadas de forma exata. Isso pode ocorrer quando as variáveis têm comportamentos semelhantes ou quando medem aspectos muito próximos da mesma coisa.

A multicolinearidade pode levar a problemas no modelo de regressão, como:

1. **Coeficientes instáveis:** Os coeficientes das variáveis independentes podem variar muito com base em pequenas mudanças nos dados, tornando difícil interpretar seus efeitos.

2. **Baixa significância estatística:** A multicolinearidade pode levar a intervalos de confiança maiores para os coeficientes e, consequentemente, à perda de significância estatística de algumas variáveis que podem ser importantes.

3. **Dificuldade de interpretação:** Quando as variáveis estão altamente correlacionadas, torna-se difícil entender a importância e o efeito individual de cada variável no modelo.

Para lidar com a multicolinearidade, algumas abordagens possíveis incluem a exclusão de variáveis correlacionadas, combinação de variáveis ou o uso de técnicas de regularização, como a regressão Ridge ou a regressão Lasso. Além disso, a realização de análise exploratória dos dados e a compreensão das relações entre as variáveis antes de construir o modelo são fundamentais para evitar problemas de multicolinearidade.

One Hot Encoding

Veja a definição na seção sobre Dummies.

Overfitting *versus* ***Underfitting***

Overfitting (sobreajuste) e *Underfitting* (subajuste) são dois problemas comuns que podem ocorrer ao treinar modelos de aprendizado de máquina. Eles referem-se a situações em que o desempenho do modelo não é ideal para dados desconhecidos, resultando em previsões imprecisas.

1. ***Overfitting*** **(sobreajuste):**

O *overfitting* ocorre quando um modelo se ajusta muito bem aos dados de treinamento, a ponto de memorizar o ruído ou detalhes específicos do conjunto de treinamento. Como resultado, o modelo se torna muito complexo e se adapta perfeitamente aos exemplos de treinamento, mas não generaliza bem para dados não vistos (conjunto de teste). Em outras palavras, o modelo "decora" os dados de treinamento, em vez de aprender os padrões subjacentes que podem ser aplicados a novos dados.

Um modelo com *overfitting* geralmente apresenta um desempenho muito bom nos dados de treinamento, mas apresenta um desempenho inferior em dados de teste, indicando que não está conseguindo generalizar bem. Para evitar *overfitting*, técnicas como a validação cruzada, redução de complexidade do modelo (por exemplo, ajuste dos hiperparâmetros) e o uso de mais dados de treinamento são algumas abordagens possíveis.

2. ***Underfitting*** **(subajuste):**

O *underfitting* ocorre quando um modelo é muito simples para capturar a complexidade dos dados de treinamento. Isso resulta em um desempenho insatisfatório tanto nos dados de treinamento quanto nos dados de teste. Em outras palavras, o modelo não é capaz de capturar os padrões relevantes dos dados, pois é muito rudimentar ou restrito em sua capacidade de aprendizado.

Um modelo com *underfitting* geralmente apresenta um desempenho ruim nos dados de treinamento e de teste, indicando que não está conseguindo aprender os padrões necessários para fazer previsões precisas. Para combater o *underfitting*, podem ser tomadas medidas como aumentar a complexidade do modelo, adicionar mais variáveis relevantes ou ajustar outros hiperparâmetros que possam permitir que o modelo aprenda melhor com os dados.

Em resumo, o *overfitting* ocorre quando o modelo é muito complexo e se ajusta demais aos dados de treinamento, enquanto o *underfitting* acontece quando o modelo é muito simples e não consegue capturar os padrões dos dados. Um bom modelo deve buscar um equilíbrio entre esses dois extremos, de forma a generalizar bem para dados não vistos e fazer previsões precisas.

Oversampling

Oversampling (sobreamostragem) é uma técnica usada em ciência de dados e aprendizado de máquina para lidar com problemas de desbalanceamento de classes em conjuntos de dados. O desbalanceamento de classes ocorre quando uma ou mais classes têm um número significativamente menor de exemplos em relação a outras classes.

A sobreamostragem consiste em aumentar artificialmente o número de exemplos das classes minoritárias, de modo que as classes fiquem mais balanceadas. Isso é feito replicando ou gerando novos exemplos para as classes minoritárias a partir dos exemplos existentes.

Existem várias abordagens de oversampling, incluindo:

1. **Replicação de exemplos:** Nesta abordagem, os exemplos existentes da classe minoritária são replicados várias vezes para aumentar sua representatividade no conjunto de dados.

2. **Técnicas de geração sintética:** Essas técnicas criam novos exemplos sintéticos da classe minoritária, utilizando algoritmos como SMOTE (*Synthetic Minority Over-sampling Technique*) ou ADASYN (*Adaptive Synthetic Sampling*). Esses algoritmos geram novos exemplos sintéticos com base em uma combinação de exemplos existentes da classe minoritária.

O objetivo da sobreamostragem é melhorar o desempenho de modelos de aprendizado de máquina ao equilibrar as classes e permitir que o modelo aprenda corretamente a representação da classe minoritária. Dessa forma, o modelo se torna mais capaz de fazer previsões precisas para ambas as classes, incluindo a classe minoritária.

No entanto, é importante tomar cuidado com a sobreamostragem excessiva, pois pode levar a um ajuste excessivo do modelo aos dados de treinamento e prejudicar o desempenho em dados desconhecidos. É fundamental encontrar um equilíbrio entre a sobreamostragem e outras técnicas de tratamento de desbalanceamento, como a subamostragem (*Undersampling*) ou o uso de métricas de avaliação apropriadas para medir o desempenho do modelo em classes desbalanceadas.

Precisão

A precisão de um modelo é uma métrica usada para avaliar o quão bem o modelo está realizando previsões corretas em relação ao número total de previsões feitas. Em outras palavras, é uma medida da proporção de exemplos corretamente classificados pelo modelo em relação ao número total de exemplos.

Matematicamente, a precisão pode ser calculada pela seguinte fórmula:

Precisão = (Número de previsões corretas) / (Número total de previsões)

Essa métrica é comumente usada em problemas de classificação, onde o modelo atribui rótulos ou classes a diferentes exemplos. A precisão é particularmente útil quando o conjunto de dados possui classes balanceadas, ou seja, cada classe tem aproximadamente o mesmo número de exemplos.

No entanto, é importante notar que a precisão pode ser enganosa em problemas com classes desbalanceadas, onde uma classe tem muitos mais exemplos do que outra. Nesses casos, o modelo pode alcançar uma alta precisão simplesmente classificando a maioria dos exemplos como pertencentes à classe majoritária. Isso pode resultar em uma precisão alta, mas o modelo pode estar falhando em prever corretamente a classe minoritária, que pode ser a classe de interesse.

Portanto, em problemas desbalanceados, é importante considerar outras métricas, como a sensibilidade (*recall*) ou a *F1-score*, que levam em conta tanto as previsões corretas como as previsões incorretas, especialmente para a classe minoritária.

Em resumo, a precisão é uma métrica importante para avaliar o desempenho de um modelo em problemas de classificação, mas é crucial considerar o contexto do problema e outras métricas relevantes, especialmente em casos de classes desbalanceadas.

Semente (*Train/Test Split*)

A semente (*seed*) no *train-test split* é um parâmetro usado em alguns algoritmos de divisão de dados em treinamento e teste, como o train_test_split da biblioteca *Scikit-learn* do Python. A semente é um número inteiro que determina como os dados serão aleatoriamente divididos em conjuntos de treinamento e teste.

Quando dividimos os dados em treinamento e teste, geralmente queremos que essa divisão seja aleatória. No entanto, a aleatoriedade pode variar a cada vez que executamos a divisão, o que pode resultar em resultados diferentes a cada execução do código.

Ao usar uma semente, podemos controlar a aleatoriedade e garantir que a divisão seja consistente em diferentes execuções. Isso é especialmente importante para fins de reprodução de resultados, quando queremos obter os mesmos resultados sempre que executarmos o código com a mesma semente.

Por exemplo, se definirmos a semente como 42 (random_state=42), cada vez que dividirmos os dados em treinamento e teste usando essa semente, obteremos exatamente a mesma divisão. Isso é útil para testar e ajustar o modelo, pois podemos comparar resultados consistentes ao modificar parâmetros ou algoritmos.

Em resumo, a semente no *train-test split* é um parâmetro que controla a aleatoriedade na divisão dos dados em treinamento e teste. Usar uma semente específica permite obter divisões consistentes e reprodutíveis, o que é útil em situações em que precisamos comparar resultados ou reproduzir experimentos.

Sensibilidade (*recall*)

Sensibilidade, também conhecida como *recall*, é uma métrica de avaliação usada em problemas de classificação, especialmente em casos onde o desbalanceamento das classes é relevante. Essa métrica mede a capacidade de um modelo em identificar corretamente os exemplos pertencentes à classe positiva (classe minoritária, por exemplo), em relação ao total de exemplos reais da classe positiva.

Matematicamente, a sensibilidade (*recall*) é calculada pela seguinte fórmula:

> Sensibilidade (*Recall*) = Número de verdadeiros positivos (TP) / (Número de verdadeiros positivos (TP) + Número de falsos negativos (FN))

Onde:

- TP (*True Positives*) é o número de exemplos corretamente classificados como pertencentes à classe positiva pelo modelo.
- FN (*False Negatives*) é o número de exemplos que pertencem à classe positiva, mas foram incorretamente classificados como pertencentes à classe negativa pelo modelo.

A sensibilidade é especialmente útil quando estamos mais interessados em evitar falsos negativos, ou seja, quando queremos que o modelo minimize as previsões errôneas de exemplos positivos como negativos. Isso é comum em problemas de diagnóstico médico, detecção de fraudes e outras situações onde o custo de um falso negativo é alto.

Uma alta sensibilidade indica que o modelo está sendo capaz de detectar corretamente a classe positiva, mas isso pode ser acompanhado de uma maior taxa de falsos positivos (exemplos da classe negativa erroneamente classificados como positivos). Portanto, é importante encontrar um equilíbrio entre a sensibilidade e a precisão (ou outras métricas), dependendo das necessidades e características específicas do problema em questão.

Em resumo, a sensibilidade (*recall*) é uma métrica que mede a capacidade de um modelo em identificar corretamente os exemplos da classe positiva, sendo especialmente relevante em problemas onde o foco está em evitar falsos negativos e maximizar a detecção correta dos exemplos positivos.

Train/Test Split

Trata-se de uma técnica de divisão dos dataframes em treino e teste. É uma estratégia comum usada em ciência de dados e aprendizado de máquina para avaliar o desempenho de modelos preditivos de forma mais realista e evitar o *overfitting* (ajuste excessivo).

Essa técnica envolve dividir o conjunto de dados original em dois subconjuntos mutuamente exclusivos:

1. **Conjunto de Treinamento (*Training Set*):** É o subconjunto dos dados usado para treinar o modelo. Nele, o modelo aprende a capturar os padrões e relações presentes nos dados.

2. **Conjunto de Teste (*Test Set*):** É o subconjunto dos dados usado para avaliar o desempenho do modelo após o treinamento. Ele contém exemplos que o modelo nunca viu durante o treinamento e serve para testar a capacidade do modelo de generalizar e fazer previsões precisas em dados não vistos anteriormente.

A divisão típica é fazer uma alocação de 70-80% dos dados para o conjunto de treinamento e 20-30% para o conjunto de teste. No entanto, essa proporção pode variar dependendo do tamanho do conjunto de dados e da complexidade do modelo.

Após treinar o modelo no conjunto de treinamento, ele é avaliado usando o conjunto de teste. A métrica de avaliação (por exemplo, precisão, acurácia, F1-score) nos dá uma medida de quão bem o modelo generaliza em dados não vistos.

É importante realizar a divisão de forma aleatória para evitar que haja viés nos conjuntos de treinamento e teste. Além disso, pode ser necessário repetir a divisão várias vezes e calcular a média das métricas de desempenho para obter uma estimativa mais robusta da capacidade do modelo.

Essa técnica é fundamental para garantir que o modelo seja avaliado de forma justa e realista e que seu desempenho seja confiável ao ser usado em dados desconhecidos, o que é essencial para o desenvolvimento de modelos precisos e úteis em problemas de aprendizado de máquina.

Underfitting

Veja a definição na seção sobre *Overfitting*.

Undersampling

O *undersampling* é uma técnica de balanceamento de dados em que reduzimos a quantidade de exemplos da classe majoritária para tornar o conjunto de dados mais equilibrado em termos de distribuição de classes. Essa técnica é frequentemente usada quando temos um conjunto de dados desequilibrado, ou seja, uma classe tem muito mais exemplos do que outras.

Ao aplicar o *undersampling*, eliminamos aleatoriamente exemplos da classe majoritária para que o número de exemplos em todas as classes seja mais parecido. Por exemplo, se tivermos um conjunto de dados com duas classes, A e B, e a classe A tiver 1000 exemplos e a classe B tiver apenas 100 exemplos, podemos usar o undersampling para reduzir o número de exemplos da classe A para, por exemplo, 200, para que ambas as classes tenham 200 exemplos.

A diferença do *undersampling* em relação a outros métodos de balanceamento mais comuns, como *oversampling* e SMOTE (*Synthetic Minority Over-sampling Technique*), é que o *undersampling* atua diretamente na classe majoritária, removendo exemplos para equilibrar o conjunto de dados. Por outro lado, o *oversampling* e o SMOTE atuam na classe minoritária, gerando novos exemplos sintéticos ou duplicando exemplos existentes para aumentar o número de exemplos dessa classe.

Comparativamente, o *undersampling* pode resultar na perda de informações importantes presentes nos exemplos removidos, uma vez que descartamos dados da classe majoritária. Além disso, se o número de exemplos da classe majoritária for significativamente maior que o da classe minoritária, o *undersampling* pode levar a um conjunto de dados muito pequeno, o que pode prejudicar a capacidade do modelo de aprender padrões complexos.

Em contrapartida, o *oversampling* e o SMOTE geram exemplos sintéticos para a classe minoritária, permitindo que o conjunto de dados fique mais balanceado sem perder informações. No entanto, essas técnicas também têm suas limitações,

como a introdução de ruído ou *overfitting* em casos onde a geração de exemplos sintéticos é feita de forma inadequada.

A escolha entre as técnicas de balanceamento dependerá do contexto específico do problema e dos dados disponíveis. Em alguns casos, pode ser necessário testar várias abordagens para determinar a que melhor se adequa aos dados e ao modelo de aprendizado de máquina a ser utilizado.

Variáveis Categóricas

Variáveis categóricas são um tipo de variável em análise de dados e ciência de dados que representam categorias ou grupos distintos. Essas variáveis têm valores que pertencem a um conjunto finito de categorias, onde cada valor está associado a uma classe ou grupo específico. Exemplos comuns de variáveis categóricas incluem: gênero (masculino/feminino), estado civil (solteiro/casado/divorciado), cor dos olhos (azul/verde/castanho) e categorias de produtos (eletrônicos, roupas, alimentos).

As principais vantagens das variáveis categóricas para a aplicação de modelos são:

1. **Representação de informações qualitativas:** As variáveis categóricas permitem representar informações qualitativas ou características não numéricas em um conjunto de dados. Essas informações podem ser valiosas para a análise de dados e para entender as características distintas dos dados.

2. **Segmentação e agrupamento:** Variáveis categóricas são úteis para segmentar e agrupar dados com base em diferentes categorias. Isso pode ajudar a identificar padrões, tendências ou diferenças entre grupos, facilitando a tomada de decisões em diferentes cenários.

3. **Interpretação e comunicação:** As variáveis categóricas podem tornar os resultados de modelos mais fáceis de interpretar e comunicar para um público não técnico. Por exemplo, em um modelo de classificação que prevê se um cliente fará uma compra ou não, é mais claro dizer que o cliente pertence à categoria "compra" ou "não compra" do que apresentar uma probabilidade numérica.

4. ***Encoding* para modelos de aprendizado de máquina:** Embora a maioria dos algoritmos de aprendizado de máquina exija dados numéricos, as variáveis categóricas podem ser convertidas em representações numéricas por meio de técnicas de *encoding*, como o *one-hot encoding*. Isso permite que as variáveis categóricas sejam usadas como recursos nos modelos de aprendizado de máquina.

5. **Enriquecimento de modelos:** As variáveis categóricas podem fornecer informações adicionais e complementares aos dados numéricos. Ao incorporar essas variáveis nos modelos, podemos melhorar a capacidade de previsão e obter insights mais ricos sobre os dados.

Em resumo, as variáveis categóricas são importantes para representar informações qualitativas e permitem uma melhor compreensão e segmentação

dos dados. Além disso, elas podem ser convertidas em representações numéricas adequadas para serem usadas em modelos de aprendizado de máquina, enriquecendo as análises e previsões.

Variáveis Preditoras ou **Variáveis Independentes** ou ***Features***

Variáveis preditoras, também conhecidas como variáveis independentes ou *features*, são as variáveis de entrada ou características utilizadas em modelos de aprendizado de máquina ou análise estatística para fazer previsões, classificações ou inferências sobre uma variável de interesse, também chamada de variável dependente ou variável resposta.

Em outras palavras, as variáveis preditoras são os atributos que usamos para explicar ou prever o comportamento ou valor de uma outra variável. Por exemplo, se estivermos construindo um modelo para prever o preço de uma casa, as variáveis preditoras podem ser o tamanho da casa, o número de quartos, a localização, a idade da propriedade, entre outras características relevantes.

No contexto da análise estatística e do aprendizado de máquina, as variáveis preditoras são consideradas como a entrada do modelo, e o objetivo é encontrar relações ou padrões entre essas variáveis e a variável resposta para fazer previsões ou tomar decisões.

A seleção adequada das variáveis preditoras é uma etapa crítica no desenvolvimento de modelos eficazes. As variáveis escolhidas devem ser relevantes e informativas para a tarefa em questão, evitando variáveis redundantes ou irrelevantes que possam prejudicar a precisão do modelo ou causar *overfitting*.

Em resumo, as variáveis preditoras são as características, atributos ou informações de entrada que utilizamos para prever, explicar ou analisar uma variável de interesse em modelos de aprendizado de máquina e análise estatística.

Variável *Targuet* ou Variável Alvo

A variável "target" (ou "variável-alvo") em um modelo preditivo é a variável que estamos tentando prever ou estimar com base em outras variáveis (também chamadas de "variáveis preditoras" ou "features"). Em outras palavras, é o valor que o modelo tenta acertar ou predizer.

Em um cenário típico de modelo preditivo, temos um conjunto de dados que contém várias observações (exemplos) onde cada observação possui várias características (variáveis preditoras) e um resultado conhecido (variável-alvo). O objetivo do modelo é aprender com esses dados para fazer previsões precisas sobre o valor da variável-alvo para novos exemplos que não estão no conjunto de dados de treinamento.

Por exemplo, suponha que queremos criar um modelo preditivo para prever o preço de venda de uma casa com base em várias características, como o tamanho da casa, o número de quartos, a localização, etc. Neste caso, o preço de venda seria a variável-alvo, e as características (tamanho da casa, número de quartos, localização, etc.) seriam as variáveis preditoras.

Ao treinar o modelo com um conjunto de dados de exemplos de casas onde conhecemos o preço de venda e as características, o modelo tentará aprender padrões e relações entre as variáveis preditoras e a variável-alvo. Posteriormente, poderemos usar esse modelo treinado para fazer previsões sobre o preço de venda de novas casas, com base em suas características.

A variável-alvo é essencial em modelos preditivos, pois é o que queremos prever ou estimar. A qualidade e precisão das previsões do modelo dependem em grande parte de como as variáveis preditoras estão relacionadas com a variável-alvo nos dados de treinamento e da capacidade do modelo de aprender e generalizar essas relações para novos dados.