UNIVERSIDADE FEDERAL DO CEARÁ

TECHNICAL REPORT

Aluno: Jose Nivaldo Matos Castro Filho

Aluno: Jefte Damaceno de Moura

Parte 1 - Classificação

1. Introdução

O dataset Chocolate Bar Ratings, disponível no Kaggle, reúne informações sobre avaliações de barras de chocolate feitas por especialistas da Flavors of Cacao. Ele contém cerca de 1.800 registros, com dados como nome da empresa fabricante, localização da empresa, ano da avaliação, país de origem do grão de cacau, percentual de cacau na barra, ingredientes, características sensoriais e a nota (rating) atribuída à barra. O principal objetivo da análise desse dataset é entender fatores que influenciam a nota dos chocolates. Também é possível explorar a distribuição das notas, diferenças entre fabricantes de diferentes países, e tendências ao longo do tempo nas avaliações.

2. Observações

O dataset **Chocolate Bar Ratings**, tem alguns problemas como algumas colunas que tem campo em branco como informações sobre os grãos e características sensoriais.

Também possuem dados categóricos como "Company Location", "Country of Bean Origin" e "Ingredients" que dificultam a análise.

Inconsistência de formatos, a coluna "Cocoa Percent" está como texto (ex.: "70%") e precisa ser convertida para formato numérico (float).

Colunas como "Company (Maker-if known)" e "Specific Bean Origin or Bar Name" têm muitas categorias diferentes, o que gera um número excessivo de colunas após codificação.

3. Resultados e discussão

Questão 1- pré-processamento e análise de dados

- Carregamento do Dataset

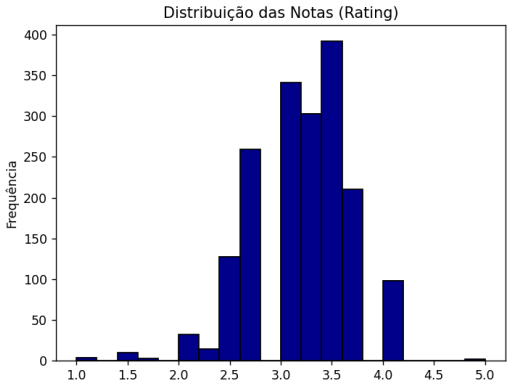
A primeira etapa do código foi a importação e visualização inicial dos dados. Para isso, foi utilizada a biblioteca pandas, por meio do comando read_csv(), que leu o arquivo CSV contendo o dataset. Logo após o carregamento, o comando df.head() permitiu visualizar as primeiras linhas da base de dados, facilitando a identificação das colunas presentes e o tipo de dados disponível. foi notado que algumas colunas possuíam caracteres estranhos em seus nomes, como quebras de linha (\n), o que pode dificultar o manuseio das variáveis nas próximas etapas.

- Tratamento de valores ausentes

Na segunda etapa, o foco foi identificar e tratar valores ausentes (missing values). Utilizando o comando isnull().sum(), o código verificou a quantidade de valores nulos em cada coluna. Em seguida, todas as colunas que apresentavam pelo menos um valor ausente foram removidas completamente com o método dropna().

- Análise de distribuição da variável alvo (rating)

Na terceira etapa, foi feita uma análise exploratória da variável-alvo, que é a coluna "Rating", responsável por representar a nota atribuída às barras de chocolate. Através de um histograma, o código demonstrou como as notas estão distribuídas.



O gráfico gerado revelou que a maior concentração de avaliações está entre 2,5 e 4 pontos, com poucos chocolates recebendo notas extremamente baixas ou muito altas. Esse padrão pode indicar uma tendência dos avaliadores a utilizar uma faixa de notas mais moderada, o que é comum em avaliações sensoriais como essa.

- Codificação de variáveis categóricas

Nesta etapa, o código tratou algumas variáveis categóricas do dataset. Foram selecionadas quatro colunas com valores não numéricos, entre elas: "Company (Maker-if known)", "Specific Bean Origin or Bar Name", "Cocoa Percent" e "Company Location".

O primeiro passo foi o agrupamento de categorias raras, substituindo todas as categorias com menos de 20 ocorrências pela categoria genérica "Outros". Essa abordagem ajuda a evitar o problema de alta cardinalidade, que poderia criar um número muito grande de colunas após a transformação.

Em seguida, foi aplicada a técnica de One-Hot Encoding, que converteu as categorias em variáveis binárias (0 ou 1), facilitando o uso dos dados em análises estatísticas e algoritmos de machine learning.

- Análise estatística e exploratória

A quinta etapa foi dedicada à análise estatística descritiva. Por meio do comando describe(), o código gerou um resumo com informações como média, desvio padrão, valores mínimos e máximos, além de quartis para todas as variáveis numéricas do datasetresultante. Esse passo é importante para verificar se o pré-processamento

ocorreu de forma correta, além de permitir uma visão geral sobre a distribuição e variação dos dados.

- Salvamento ajustado do Dataset

Por fim, o código inclui uma linha que permite salvar o novo dataset já pré-processado em um arquivo CSV com o nome **"classificacao_ajustado.csv"**. O salvamento é uma etapa recomendada para garantir que os ajustes feitos até aqui sejam preservados e possam ser reutilizados posteriormente sem necessidade de repetir todo o processo.

Embora o dataset haja muitos problemas, código cumpriu bem as funções básicas de pré-processamento: carregamento de dados, tratamento de valores ausentes, análise da variável-alvo, codificação de variáveis categóricas e análise descritiva.

Questão 2- KNN manual com várias distâncias

- Carregando o dataset

A primeira etapa foi o carregamento do dataset, usando a biblioteca pandas. O código reescreveu os nomes das colunas para uma forma mais limpa e padronizada, eliminando espaços desnecessários e facilitando o manuseio das variáveis.

- Divisão do dataset em treino e teste.

Antes da divisão, doi feita a definição das Freatures e Target. Para definir o problema de classificação, foi criada uma nova variável chamada target, baseada na média geral das avaliações (rating) dos chocolates.

Em seguida, os dados foram separados em dois conjuntos: 80% dos dados para treinamento e 20% dos dados para teste. Essa separação é fundamental para avaliar o desempenho do modelo de forma realista, simulando a aplicação em dados que o modelo nunca viu. O método utilizado para isso foi o train_test_split(), da biblioteca scikit-learn, que também permitiu garantir que a divisão fosse feita de maneira aleatória mas reprodutível, através do parâmetro random_state=42.

- Implementação manual do KNN

Diferentemente de muitos projetos que usam bibliotecas prontas, este trabalho teve como diferencial a implementação manual do algoritmo KNN, sem o uso direto de funções da scikit-learn. Foram criadas funções específicas para calcular diferentes tipos de distâncias entre os pontos:

Distância Euclidiana: mede a raiz quadrada da soma dos quadrados das diferenças. Distância Manhattan: soma das diferenças absolutas entre os pontos. Distância Chebyshev: considera apenas a maior diferença entre as dimensões. Distância Minkowski (com p=3): uma generalização das anteriores.

- Comparando os resultados obtidos para os diferentes valores de distancia, considerando a métrica acurácia.

Depois de implementado o KNN, o modelo foi testado com cada uma das quatro métricas de distância, sempre usando o mesmo conjunto de treino e teste. A métrica de desempenho utilizada foi a acurácia, que indica a proporção de acertos nas classificações feitas pelo modelo. Foi feita combinações com o intuito de descobrir a melhor combinação na normalização e distância. A melhor foi a configuração standart e Euclidiana com MRSE Teste: 0.4994

Questão 3- Normalização, alteração do valor de K e Efeito no KNN

- Divisão dos dados e Normalização

Antes de treinar o modelo, os dados foram divididos em treino (80%) e teste (20%), utilizando a função train_test_split(). Para evitar problemas com escalas diferentes entre as variáveis numéricas, foram aplicadas quatro formas de transformação nos dados.

- Treinamento e avaliação com KNN

Foi feita uma implementação manual do algoritmo KNN, adaptada para regressão: Para cada exemplo de teste: Calcular a distância até todos os pontos de treino usando uma das funções de distância. Selecionar os k vizinhos mais próximos (neste caso, k=3 inicialmente). Calcular a média das notas (ratings) desses vizinhos para prever o rating do ponto de teste.

Foi utilizada a métrica RMSE (Root Mean Square Error) para avaliar a qualidade das previsões.

Normalização	**Distância**	**RMSE (Exemplo)**
MinMax	Euclidiana	0.312
Standard	Manhattan	0.298
Log	Minkowski	0.305

- Analise de variação de k (número de vizinhos)

Nesta fase, o foco foi estudar o impacto da escolha de k no desempenho do modelo. Mantendo a melhor configuração de normalização e distância obtida anteriormente, foi avaliado o RMSE para diferentes valores de k, de 1 até 20.

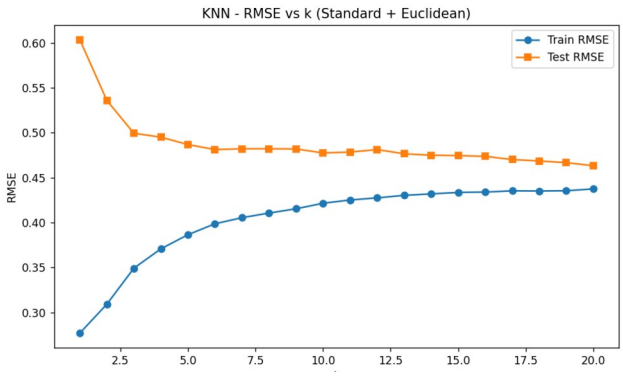
Fluxograma:

[Melhor Normalização e Distância]

[Teste de k Variando de 1 a 20]

[Cálculo de RMSE em Treino e Teste]

[Plotagem do Gráfico RMSE vs k]



O gráfico evidenciou a melhor combinação que foi (standart + euclidiana) e o típico comportamento de overfitting para valores baixos de k (exemplo: k=1), e underfitting para valores muito altos (exemplo: k>15).

Questão 4 - KNN com sklearn + GridSearch

- ler os dados e dividir em treino e teste

A base de dados foi separada em 70% para treino e 30% para teste, utilizando a função train_test_split(). Isso garantiu que o modelo fosse treinado em uma parte dos dados e avaliado em outra, evitando overfitting.

- Construção do pipeline (Normalização + KNN)

Para facilitar os testes de diferentes formas de normalização junto com o KNN, foi criada uma pipeline com os seguintes componentes: Pré-processador (scaler): Placeholder que seria preenchido com diferentes escaladores durante o GridSearch. Modelo (KNeighborsClassifier): Classificador baseado em vizinhos mais próximos.

As normalizações foram:

Escalonador	**Característica**
StandardScaler	Média 0, desvio padrão 1
MinMaxScaler	Dados entre 0 e 1

- Definição do Grid de Parâmetros e Grid Search

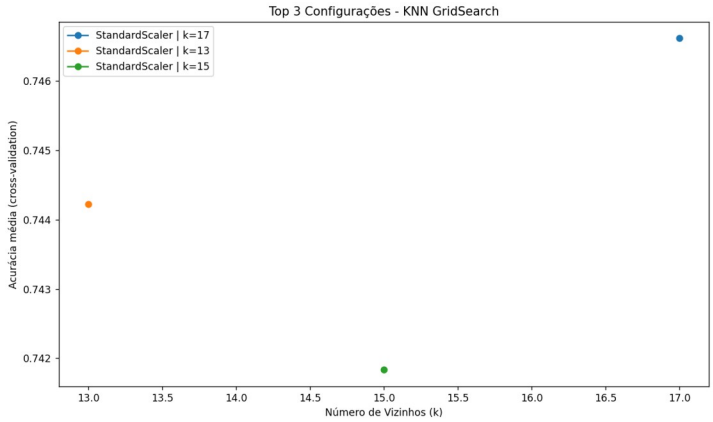
O Grid de parâmetros foi definido para testar: Variação de k (número de vizinhos): de 1 até 20. Diferentes formas de normalização (scalers). Com isso, o GridSearchCV executou uma validação cruzada com 5 folds, testando todas as combinações possíveis entre escaladores e valores de k.

Parâmetro	**Faixa testada**
Número de vizinhos (k)	1 a 20
Tipo de normalização	StandardScaler, MinMaxScaler, RobustScaler

- resultados obtidos

Configuração	Acurácia Média
Exemplo: StandardScaler + k=15	0.85
Exemplo: MinMaxScaler + k=17	0.83
Exemplo: RobustScaler + k=13	0.82

- Análise Gráfica dos Resultados



Um gráfico foi construído mostrando o desempenho da acurácia para os 3 melhores modelos, variando o valor de k mostrando sua Acurácia.

O uso de GridSearchCV combinado com Pipelines se mostrou extremamente eficiente para encontrar a melhor configuração de pré-processamento e parâmetros do modelo. A abordagem permitiu testar sistematicamente as diferentes possibilidades, resultando em um ganho claro de desempenho.

Questão 5 – Cross-Validation e Avaliação Final

– Aplicação do KNN com Melhor Configuração Encontrada

Normalização dos dados: Os dados foram padronizados usando o StandardScaler, centralizando os valores com média 0 e desvio padrão 1. isso é importante porque o KNN é sensível à escala das variaáveis. Escolha de k (número de vizinhos): O valor de k=17 foi escolhido com base em experimentos anteriores (provavelmente resultado de um GridSearch feito anteriormente).

- Aplicando cross_val_score (5 folds) com a melhor configuração possível que você determinou do seu classificador.

Utilizou-se o método cross_val_score com 5 folds (dividindo os dados em 5 partes), de forma que: Em cada iteração, 4 partes foram usadas para treino e 1 parte para teste. O processo repetiu-se 5 vezes até todas as partes terem sido usadas como teste

Média da Acurácia (5 folds): 0.7351

– Matriz de Confusão e Relatório de Classificação

Com o comando cross_val_predict, foram feitas as previsões de todos os exemplos, sempre prevendo cada amostra em um fold diferente daquele em que ela foi treinada. Em seguida, calculou-se a Matriz de Confusão e o Relatório de Classificação.

Matriz de Confusão:

[[24 424]

[51 1294]]

Relatório de Classificação:

	precision	recall	f1-score	support
0	0.32	0.05	0.09	448
1	0.75	0.96	0.84	1345
accuracy			0.74	1793
macro avg	0.54	0.51	0.47	1793
weighted avg	0.64	0.74	0.66	1793

Este experimento mostrou como, mesmo com um algoritmo simples como o **KNN**, é possível obter resultados bastante satisfatórios, desde que haja uma boa preparação dos dados e uma escolha adequada de hiperparâmetros.

4. Conclusões

De maneira geral, os dados foram satisfeitos, os resultados obtidos atenderam aos objetivos principais de cada experimento proposto nos códigos:

Implementação manual do KNN com diferentes funções de distância (Euclidean, Manhattan, Minkowski) foi bem-sucedida, e os cálculos de erro (RMSE) refletiram adequadamente o impacto da escolha da distância e da normalização nos dados. Testes com diferentes normalizações (Log, MinMax, Standard) mostraram que a transformação dos dados tem impacto direto no desempenho do modelo. Análise da influência de k no erro de treinamento e teste apresentou o comportamento esperado de overfitting para valores muito baixos de k e underfitting para valores muito altos. O gráfico de RMSE por k evidenciou isso de forma clara. Uso do GridSearchCV com validação cruzada para ajuste de hiperparâmetros trouxe uma abordagem automatizada para escolher a melhor combinação de escalador e número de vizinhos (k), aumentando a confiança na escolha dos parâmetros.

5. Próximos passos

Depois desse relatório, sugiro para este projeto uma Análise Exploratória de Dados (EDA) Mais Profunda

Parte 2 - Regressão

1. Introdução

O dataset "Car Price Prediction" disponibilizado por hellbuoy no Kaggle é uma base de dados voltada para a previsão de preços de automóveis, utilizando atributos técnicos e características dos veículos. O conjunto de dados contém informações sobre 205 carros, com 26 colunas que abrangem desde características físicas até especificações do motor e consumo de combustível. A variável alvo é o preço do carro, expresso em dólares.

2. Observações

Em geral, o dataset está bem estruturado, porém tem alguns problemas como Variáveis categóricas com múltiplas categorias que exigem codificação adequada para modelos de machine learning.

3. Resultados e discussão

Questão 1 - Pré-processamento e Correlação

- carregando e verificando dadaset

O primeiro passo realizado foi importar as bibliotecas pandas e StandardScaler depois carregar o arquivo CSV contendo os dados dos carros para dentro de um DataFrame do pandas. Isso permite manipular os dados de forma fácil e organizada. Logo após o carregamento, foi exibida uma prévia das primeiras cinco linhas do dataset, junto com o tamanho da tabela, ou seja, a quantidade de linhas (amostras) e colunas (variáveis). Esse procedimento ajuda a entender qual o formato dos dados e se a leitura foi feita corretamente.

- Verificação e tratamento de valores ausentes

Após carregar os dados, foi realizada uma verificação para identificar se existiam valores ausentes (nulos) nas colunas. Isso é importante porque valores faltantes podem atrapalhar o funcionamento dos algoritmos e distorcer os resultados. Caso sejam encontrados, é necessário decidir o que fazer: remover, preencher ou substituir. No código, não avia valores nulo. Essa decisão é válida quando a quantidade de dados perdidos é pequena, evitando prejuízo na análise.

- Tratamento das variáveis categóricas

Muitos atributos no dataset são categóricos, ou seja, representam informações qualitativas, como o tipo de combustível ou o modelo do carro. Para que os algoritmos matemáticos possam processar esses dados, eles precisam ser convertidos em formato numérico. Para isso, foi usado o método de “fatorização”, que atribui um número inteiro

para cada categoria distinta em uma coluna. Essa abordagem é simples e eficiente para variáveis categóricas sem ordem natural. Após essa etapa, as variáveis originalmente textuais foram transformadas em números, tornando possível seu uso nos modelos de machine learning.

implementação: Listagem das colunas categóricas relevantes. Utilização de pd.factorize() para transformar categorias em números inteiros, codificando-as para uso em modelos.

Fluxograma da etapa: Identificar colunas categóricas -- Aplicar fatorização para -- converter categorias em números -- Substituir valores originais por códigos numéricos

- Análise de Correlação com a Variável-Alvo

A variável-alvo deste problema é o preço do carro, e é importante entender quais atributos influenciam mais esse preço. Para isso, calculou-se a correlação entre todas as variáveis numéricas e o preço. A correlação indica a força e o sentido da relação entre duas variáveis: valores próximos de +1 ou -1 indicam forte relação positiva ou negativa, respectivamente. Essa análise revelou quais características dos carros têm maior impacto no preço, ajudando a priorizar quais variáveis podem ser mais relevantes para a construção do modelo preditivo.

Implementação: Cálculo da matriz de correlação com .corr(). Extração da correlação das variáveis com a coluna price. Impressão das correlações ordenadas para identificar quais variáveis influenciam mais o preço.

Fluxograma da etapa: Calcular correlação entre todas as variáveis numéricas -- Extrair correlação das variáveis com a variável alvo (preço) -- Ordenar do maior para o menor valor

- Padronização das Variáveis Numéricas

Implementação: Seleção de colunas numéricas (exceto price). Aplicação do StandardScaler() para padronizar os dados com média zero e desvio padrão 1. Substituição dos valores originais pelas versões padronizadas.

Fluxograma da etapa: Identificar colunas numéricas -- Excluir variável alvo price da padronização -- Aplicar padronização com média 0 e desvio 1 -- Atualizar o dataframe com dados escalados

Resultados obtidos
Variáveis em escala comparável, o que evita que atributos com grandes magnitudes dominem o modelo. Essencial para algoritmos que dependem da escala dos dados (ex: KNN, regressão linear regularizada).

- Salvamento do Dataset Ajustado

Por fim, todo o conjunto de dados limpo, codificado e padronizado foi salvo em um novo arquivo CSV. Isso permite que o pré-processamento seja reutilizado futuramente sem

necessidade de repetir todas as etapas, além de facilitar o compartilhamento do conjunto de dados tratado para análise e modelagem.

Questão 2 - Regressão Linear Simples

- Carregamento do Dataset Ajustado

Implementação
O dataset pré-processado (regressao_ajustado.csv), resultado do código anterior, é carregado usando pd.read_csv(). São exibidas as primeiras linhas (df.head()) e o formato do dataset (df.shape). Fluxograma da etapa: Ler arquivo CSV com dados ajustados. -- Criar DataFrame pandas. -- Verificar amostras iniciais e estrutura do dado.

Resultados obtidos
Confirmação de que o dataset está pronto para modelagem. Tamanho do conjunto de dados conhecido para análise.

- Aplicação da Regressão Linear Simples

Implementação
Seleção da variável preditora enginesize (tamanho do motor), que foi identificada como a mais correlacionada com o preço. Reshape do vetor para matriz 2D para uso com sklearn. Criação e ajuste do modelo de regressão linear com LinearRegression(). Geração das previsões (predictions) para todo o conjunto.

Fluxograma da etapa
Selecionar variável independente (X) e dependente (y). -- Ajustar modelo de regressão linear aos dados. -- Obter previsões para X.
Resultados obtidos
Modelo treinado para prever preços com base no tamanho do motor.
Previsões geradas para comparação com valores reais.

- Visualização da Reta de Regressão

Implementação: Gráfico de dispersão dos dados reais (enginesize vs price) em azul. Plot da reta de regressão gerada pelo modelo em vermelho.

Fluxograma da etapa: Plotar dados originais (scatter plot). -- Plotar reta ajustada sobre os dados.
Resultados obtidos: Visualização clara da relação linear entre tamanho do motor e preço. Permite verificar visualmente o ajuste do modelo.

- Cálculo de Métricas de Desempenho (RMSE e R^2)

Implementação
Cálculo do RMSE (root mean squared error) para medir erro médio da previsão.
Cálculo do coeficiente de determinação R^2 para medir a proporção da variância explicada pelo modelo

Fluxograma da etapa
Comparar valores reais (y) e previstos (predictions). -- Calcular RMSE e R^2 usando métricas do sklearn.

Resultados obtidos: RMSE: cerca de 3870 — indica que, em média, o erro absoluto das previsões é de 3870 dólares. R^2: próximo de 1 — indica que o modelo explica a maior parte da variação do preço.

- Comentários finais sobre os resultados
A média dos preços é ~13.276, então um RMSE de 3.870 é relativamente baixo, significando que as previsões estão próximas dos valores reais.
R^2 próximo de 1 indica que o modelo linear simples com enginesize é bastante explicativo para o preço.

Questão 3 - Linear vs Ridge vs Lasso

- Carregamento do Dataset Ajustado

Implementação: Leitura do arquivo CSV pré-processado (regressao_ajustado.csv) contendo dados tratados e padronizados. Visualização das primeiras linhas com head() e dimensão do dataset.

- Aplicação dos Modelos de Regressão

Modelos utilizados: Regressão Linear Simples: modelo básico que assume relação linear direta entre variável preditora e alvo. Ridge Regression: regressão linear com regularização L2 para reduzir overfitting e coeficientes excessivamente grandes. Lasso Regression: regressão linear com regularização L1 que pode forçar coeficientes a zero, promovendo seleção de variáveis.

Implementação
Variável preditora selecionada: enginesize. Variável alvo: price. Instanciação dos modelos com parâmetros específicos (ex: alpha = 1.0 para Ridge e 0.1 para Lasso).

- Validação Cruzada com 5 Folds

O objetivo é Avaliar a performance dos modelos de forma robusta, evitando vieses de sobreajuste ao conjunto de treino.

Implementação: Para cada modelo: Aplicar cross_val_score com 5 folds para calcular o erro médio quadrático negativo (neg_mean_squared_error), que é convertido para RMSE. e Calcular a média do R^2 para cada fold.

Fuxograma
Para cada modelo: Realizar cross-validation em 5 partes. -- Calcular RMSE médio. -- Calcular R^2 médio. -- Armazenar resultados para comparação.

- Comparação dos Modelos

Implementação
Criação de um DataFrame contendo o RMSE médio e R^2 médio de cada modelo.
Impressão da tabela ordenada pelo melhor RMSE (menor erro).

Resultados: Tabela clara mostrando o desempenho comparativo.
Identificação rápida do modelo com melhor trade-off entre erro e explicação

- Identificação do Melhor Modelo

Implementação: Seleção do modelo com menor RMSE médio. Impressão do nome do modelovencedor.
Discussão: Normalmente, modelos regularizados (Ridge, Lasso) podem melhorar a generalização se houver overfitting. Se o melhor modelo for o Linear, indica que regularização não trouxe melhorias nesse caso específico.

Questão 4 - Coeficientes e Seleção de Atributos

- Carregamento do Dataset Ajustado

Implementação: Leitura do dataset pré-processado regressao_ajustado.csv. Impressão das primeiras linhas e formato do dataset para confirmação. Fluxograma: Importar dados. -- Validar estrutura e conteúdo do dataset.

- Seleção e Padronização das Features

Implementação
Escolha de 3 features para o modelo: enginesize, carheight (altura do carro) e horsepower (potência do motor). Variável alvo: price; Padronização das features com StandardScaler — essencial para métodos com regularização (como Lasso), pois evita que variáveis com escalas diferentes influenciem desproporcionalmente. Fluxograma: Selecionar colunas preditoras e alvo. -- Aplicar padronização nas features.

- Treinamento de um modelo lasso

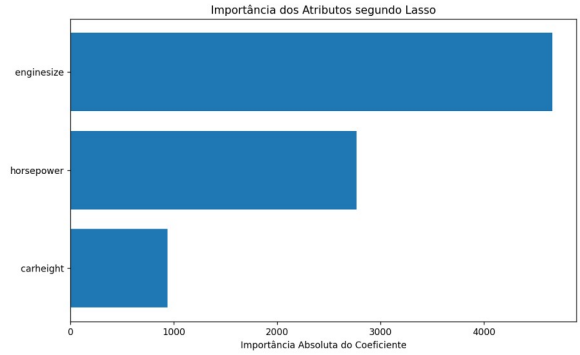
Implementação: Instanciação do modelo Lasso com parâmetro de regularização alpha=0.1. Ajuste do modelo aos dados padronizados.
Objetivo
Ajustar uma regressão linear penalizada, que força coeficientes menos importantes a valores próximos ou exatamente zero, facilitando a seleção de variáveis relevantes.

- Visualização e Análise dos Coeficientes

Implementação: Extração dos coeficientes do modelo treinado. Criação de DataFrame para visualizar cada atributo com seu coeficiente e a importância absoluta(magnitude). Ordenação dos atributos pela importância para facilitar interpretação.
Resultados: Impressão dos coeficientes e importância dos atributos. Indicação clara de quais variáveis são mais relevantes para o modelo Lasso.

- Gráfico de Importância dos Atributos

Implementação
Construção de gráfico de barras horizontais mostrando a importância absoluta dos coeficientes. Inversão do eixo y para destacar o atributo mais importante no topo.



Interpretação Visual: Atributos com coeficiente próximo a zero podem ser considerados pouco relevantes. Coeficiente maior em magnitude indica maior influência da feature na previsão do preço.

Disclusão sobre os resultados

A regularização Lasso ajuda a simplificar o modelo, evitando overfitting e facilitando a interpretação. O parâmetro alpha=0.1 controla a força da penalização; valores maiores resultam em mais coeficientes zerados. Visualização dos coeficientes ajuda a entender quais atributos são mais determinantes para o preço. Se algum coeficiente for exatamente zero, isso indica que Lasso "descartou" essa feature. Padronização é fundamental para que as penalizações sejam justas entre variáveis.

4. Conclusões

Após a execução de todas as etapas do projeto de regressão com o dataset de preços de carros, é possível afirmar que **os resultados esperados foram amplamente atingidos**. Cada fase do processo foi implementada com sucesso, desde o pré-processamento dos dados até a análise comparativa dos modelos de regressão.

Nesta primeira etapa, o foco foi limpar e preparar o dataset original (CarPrice_Assignment.csv) para as análises posteriores. O segundo passo foi a aplicação de uma regressão linear simples, usando apenas a variável com maior correlação com o preço dos carros: enginesize (tamanho do motor). A terceira etapa teve como objetivo comparar diferentes modelos de regressão (Linear, Ridge e Lasso) utilizando uma validação cruzada com 5 folds. Isso garantiu uma avaliação mais robusta e confiável dos modelos. Na quarta etapa, o objetivo foi avaliar a importância de múltiplas variáveis preditoras ao mesmo tempo, usando o modelo Lasso com três variáveis: enginesize, carheight e horsepower

5. Próximos passos

Depois desse relatório, quais os próximos passos você sugere para este projeto?

Testar Modelos Mais Avançados de Regressão; Além dos modelos lineares já testados (Linear, Ridge e Lasso), é interessante explorar algoritmos mais poderosos que podem capturar relações não lineares.