MARCELO MIKY MINE

MACHINE LEARNING ENGINEERING

APRENDIZADO SUPERVISIONADO

AULA 04

SUMÁRIO

O QUE VEM POR AÍ?

Vimos na aula anterior a tarefa de Classificação, quando o conjunto de dados possui a classe discreta. A tarefa de Regressão se baseia em um conjunto de dados de treinamento em que cada amostra é composta por um conjunto de variáveis de entrada x e uma variável contínua y de saída. O objetivo principal é treinar um modelo que aprenda a mapear as entradas para as respectivas saídas, permitindo a realização de previsões precisas para novos dados.

Utiliza-se a regressão em diversas indústrias, como previsão de preços, analisando dados históricos de vendas para estimar a demanda futura e otimizar os estoques, análise financeira, para prever o risco de crédito de clientes e identificar oportunidades de investimento, e manutenção preditiva, para prever falhas antes que ocorram, evitando paradas inesperadas e custos adicionais.

HANDS ON

No universo do aprendizado de máquina supervisionado, a regressão se destaca como uma ferramenta poderosa para lidar com dados contínuos. Através da construção de modelos preditivos, ela permite estimar valores futuros com base em dados históricos, abrindo portas para soluções inovadoras em diversas áreas. O problema de Classificação, apresentado anteriormente, pode ser considerado como um tipo de problema de Regressão devido a sua similaridade. Sua principal diferença consiste na saída do modelo: na Classificação o resultado é discreto e na Regressão o resultado é contínuo.

Um exemplo de um problema de Regressão é a predição do valor estimado dos preços de casas. Se esta pergunta for feita em um problema de Classificação, as possíveis classes poderiam ser as categorias A, B e C, representando os preços Alto, Médio e Baixo. Se considerarmos a abordagem de um problema de Regressão, as respostas poderiam ser os valores dos preços. As tarefas de separação em conjuntos de treinamento e teste são feitas da mesma forma para os dois problemas.

Nesta jornada, exploraremos a regressão em detalhes, desde seus fundamentos até a implementação prática em Python com as bibliotecas Pandas e Scikit-learn. Assim, veremos alguns modelos de regressão e métricas próprias para avaliar o desempenho.

SAIBA MAIS

Similar à classificação, a regressão também se baseia no aprendizado supervisionado a partir de dados históricos. No entanto, em vez de prever categorias como "sim" ou "não", a regressão se concentra na estimativa de valores contínuos. A principal diferença está na classe e as respectivas métricas de avaliação. A classificação foca na precisão da classificação em categorias discretas; a regressão mede a distância ou o erro entre a saída do modelo (um valor numérico) e a saída real desejada.

O objetivo da tarefa de regressão é construir um modelo que estime o valor mais provável da variável resposta para um novo padrão (exemplo) com base em um conjunto de dados históricos. O objetivo do treinamento é minimizar o erro observado ajustando os parâmetros do modelo para que as saídas preditas se aproximem ao máximo dos valores reais.

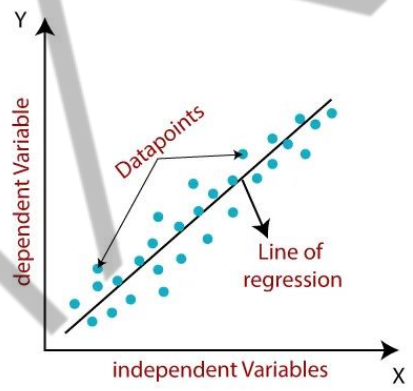

Figura 1 - Dados hipotéticos de um exemplo de dados e uma reta modelada com regressão linear
Fonte: Trehan (2020)

Exemplo Prático: imagine um modelo de regressão que prevê o preço de casas. O modelo é treinado com dados de preços de casas no passado. O conjunto de dados possui informações como número de quartos e banheiros, área em m² do espaço interior, m² do espaço do terreno, vista para o mar e localização, entre outros (figura 2). O objetivo é ajustar o modelo para que ele possa estimar com precisão o preço de novas casas.

	id	date	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	lat	long	yr_built	zipcode	price
0	7129300520	20141013T000000	3	1.00	1180	5650	1.0	0	47.5112	-122.257	1955	98178	221900.0
1	6414100192	20141209T000000	3	2.25	2570	7242	2.0	0	47.7210	-122.319	1951	98125	538000.0
2	5631500400	20150225T000000	2	1.00	770	10000	1.0	0	47.7379	-122.233	1933	98028	180000.0
3	2487200875	20141209T000000	4	3.00	1960	5000	1.0	0	47.5208	-122.393	1965	98136	604000.0
4	1954400510	20150218T000000	3	2.00	1680	8080	1.0	0	47.6168	-122.045	1987	98074	510000.0

Figura 2 - Alguns exemplos de um conjunto de dados de casas
Fonte: Elaborado pelo autor (2024)

Modelos de Regressão:

- **Regressão Linear:** estabelece uma relação linear entre as variáveis de entrada e a variável de saída, modelando a dependência através de uma reta. Ideal para relações simples e facilmente interpretáveis.
- **Regressão de Árvore de Decisão:** utiliza uma árvore de decisão como modelo preditivo para partir de observações dos exemplos, representado nos ramos, para conclusões sobre o valor da classe, representado nas folhas.
- **K-NN Regressor**: regressão baseada no algoritmo k-vizinhos mais próximos. O alvo é previsto pela interpolação local dos alvos associados aos vizinhos mais próximos no conjunto de treinamento.
- **Support Vector Machines Regression:** o SVR busca encontrar uma reta ou hiperplano no espaço de dados que minimize o erro entre as previsões do modelo e os valores reais.
- **XGBoost:** implementa o conceito de **gradient boosting** para construir modelos de regressão precisos e eficientes. Combina diversos modelos "fracos", como árvores de decisão, em um único modelo forte, agregando as previsões individuais para melhorar a acurácia geral.

Métricas de Avaliação:

Para avaliar o desempenho dos modelos de regressão, diversas métricas são utilizadas, como: Erro Médio Absoluto (MAE), Raiz Quadrada Média do Erro (RMSE) e Média da Porcentagem do Erro Absoluto (MAPE).

Erro Médio Absoluto, sigla em inglês para **Mean Absolute Error**, mede a diferença média entre os valores preditos e os valores reais. Um valor baixo indica um

bom desempenho do modelo, quantificando a **discrepância média** entre os valores **previstos** pelo modelo e os valores **reais** observados.

Funcionamento Detalhado:

- **Cálculo do Erro Absoluto**: para cada ponto de dados, o erro absoluto é calculado como a diferença absoluta entre o valor previsto e o valor real. Matematicamente, representamos por:
- Erro Absoluto(i) = |Valor Real(i) - Valor Previsto(i)|

Em que:

- **Erro Absoluto(i)** representa o erro absoluto para o i-ésimo ponto de dados.
- **Valor Real(i)** representa o valor real observado para o i-ésimo ponto de dados.
- **Valor Previsto(i)** representa o valor previsto pelo modelo para o i-ésimo ponto de dados.

Cálculo do Erro Médio Absoluto (MAE): após calcular o erro absoluto para cada ponto de dados, o MAE é obtido pela **média** dos erros absolutos individuais:

- MAE = (1/N) * Σ[Erro Absoluto(i)]

Em que:

- **MAE** representa o Erro Médio Absoluto.
- **N** representa o número total de pontos de dados.

Interpretação do MAE:

- **Valor Baixo de MAE:** indica que, em média, o modelo está fazendo previsões próximas dos valores reais, demonstrando bom desempenho.
- **Valor Alto de MAE:** indica que, em média, o modelo está fazendo previsões distantes dos valores reais, o que significa que precisa ser melhorado.

Vantagens do MAE:

- **Fácil interpretação:** o MAE fornece uma medida intuitiva da discrepância média entre os valores previstos e reais, facilitando a compreensão do desempenho do modelo.
- **Robustez a outliers:** o MAE é menos sensível a valores discrepantes (outliers) em comparação com outras métricas como o Erro Médio Quadrático (MSE), pois utiliza o valor absoluto das diferenças.

Desvantagens do MAE:

- **Não leva em conta a magnitude dos erros:** o MAE trata todos os erros com a mesma importância, ignorando a magnitude dos valores reais. Em alguns casos, erros em valores maiores podem ser mais relevantes.
- **Não fornece informações sobre a direção do erro:** o MAE indica apenas a distância média entre os valores previstos e reais, sem informar se as previsões estão acima ou abaixo dos valores reais.

Raiz quadrada média do erro, sigla em inglês para Root Mean Squared Error (RMSE), é uma métrica amplamente utilizada para avaliar o desempenho de modelos de regressão em Machine Learning. Essa métrica quantifica a discrepância média entre os valores previstos pelo modelo e os valores reais observados, fornecendo uma medida abrangente da precisão do modelo.

Funcionamento Detalhado:

- **Cálculo do Erro Quadrático**: para cada ponto de dados, o erro quadrático é calculado como o quadrado da diferença entre o valor previsto e o valor real.
    - Erro Quadrático(i) = [Valor Real(i) - Valor Previsto(i)]^2

Em que:

- **Erro Quadrático(i)** representa o erro quadrático para o i-ésimo ponto de dados.
- **Valor Real(i)** representa o valor real observado para o i-ésimo ponto de dados.

- **Valor Previsto(i)** representa o valor previsto pelo modelo para o i-ésimo ponto de dados.

Cálculo da Raiz Quadrada Média do Erro (RMSE): após calcular o erro quadrático para cada ponto de dados, o RMSE é obtido pela raiz quadrada da média dos erros quadráticos individuais:

- RMSE = √[(1/N) * Σ[Erro Quadrático(i)]]

Em que:

- **RMSE** representa a Raiz Quadrada Média do Erro.
- **N** representa o número total de pontos de dados.

Interpretação do RMSE:

- Valor **Baixo** de RMSE: indica que, em média, o modelo está fazendo previsões próximas dos valores reais, demonstrando bom desempenho.
- Valor **Alto** de RMSE: indica que, em média, o modelo está fazendo previsões distantes dos valores reais, o que significa que precisa ser melhorado.

Vantagens do RMSE:

- **Unidade de medida**: o RMSE possui a mesma unidade de medida que os valores previstos e reais, facilitando a interpretação da magnitude do erro.
- **Sensibilidade a grandes erros**: o RMSE é mais sensível a grandes erros em comparação com o Erro Médio Absoluto (MAE), pois eleva os erros ao quadrado antes da média. Isso o torna adequado para tarefas em que grandes erros têm maior impacto.
- **Leva em conta a magnitude dos erros**: o RMSE leva em conta a magnitude dos erros, diferentemente do MAE, o que pode ser importante em alguns casos.

Desvantagens do RMSE:

- **Sensibilidade a outliers**: o RMSE é sensível a valores discrepantes (outliers) devido à elevação ao quadrado dos erros. Isso pode distorcer a avaliação do desempenho do modelo em alguns casos.
- **Não fornece informações sobre a direção do erro**: o RMSE indica apenas a distância média entre os valores previstos e reais, sem informar se as previsões estão acima ou abaixo dos valores reais.

Mean Absolute Percentage Error (MAPE): métrica de erro que calcula a média da porcentagem do erro absoluto entre os valores previstos por um modelo de regressão e os valores reais observados. Por expressar o erro médio como uma porcentagem, facilita a interpretação e a comparação entre diferentes modelos.

Funcionamento Detalhado:

- **Cálculo do Erro Absoluto Percentual (MAPE):** para cada ponto de dados, o erro absoluto percentual é calculado como a diferença absoluta entre o valor previsto e o valor real, dividida pelo valor real e multiplicada por 100:
- Erro Absoluto Percentual(i) = |(Valor Real(i) - Valor Previsto(i)) / Valor Real(i)| * 100

Em que:

- **Erro Absoluto Percentual(i)** representa o erro absoluto percentual para o i-ésimo ponto de dados.
- **Valor Real(i)** representa o valor real observado para o i-ésimo ponto de dados.
- **Valor Previsto(i)** representa o valor previsto pelo modelo para o i-ésimo ponto de dados.

Cálculo do Erro Absoluto Percentual Médio (MAPE): após calcular o erro absoluto percentual para cada ponto de dados, o MAPE é obtido pela **média** dos erros absolutos percentuais individuais:

- MAPE = (1/N) * Σ[Erro Absoluto Percentual(i)]

Em que:

- **MAPE** representa o Erro Absoluto Percentual Médio.
- **N** representa o número total de pontos de dados.

Interpretação do MAPE:

- **Valor Baixo de MAPE (próximo a 0%):** indica que, em média, o modelo está fazendo previsões muito próximas dos valores reais, demonstrando excelente desempenho.
- **Valor Alto de MAPE:** indica que, em média, o modelo está fazendo previsões significativamente distantes dos valores reais, o que significa que precisa ser melhorado.

Vantagens do MAPE:

- **Expressão em porcentagem:** o MAPE expressa o erro como uma porcentagem, facilitando a comparação do desempenho do modelo em diferentes escalas e conjuntos de dados.
- **Adequado para séries temporais:** o MAPE é particularmente útil para séries temporais, pois leva em conta a magnitude dos valores reais ao calcular o erro.
- **Interpretação intuitiva:** o MAPE fornece uma medida intuitiva da discrepância percentual entre os valores previstos e reais.

Desvantagens do MAPE:

- **Sensibilidade a valores reais zero:** o MAPE é indefinido para valores reais zero, o que pode limitar sua aplicabilidade em alguns casos.
- **Não leva em conta a distribuição dos erros:** o MAPE não leva em conta a distribuição dos erros, assumindo que todos os erros têm a mesma importância.
- **Ignora a direção do erro:** o MAPE indica apenas a distância média entre os valores previstos e reais, sem informar se as previsões estão acima ou abaixo dos valores reais.

Implementação Prática com Python e Scikit-learn

Nesta seção, iniciaremos nossa jornada em diversos modelos de regressão utilizando as bibliotecas **Pandas** e **Scikit-learn** em Python. O dataset escolhido para essa demonstração é o **"House Sales in King County, USA"** do Kaggle, que contém informações sobre preços de casas em King County, estado do Washington.

Para iniciar, fazemos a importação das bibliotecas e carregamos o conjunto de dados (código-fonte 1).

```
import numpy as np
import pandas as pd

df = pd.read_csv('kc_house_data.csv')
df.head()
```

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	...
0	7129300520	20141013T000000	221900.0	3	1.00	1180	5650	1.0	0	0	...
1	6414100192	20141209T000000	538000.0	3	2.25	2570	7242	2.0	0	0	...
2	5631500400	20150225T000000	180000.0	2	1.00	770	10000	1.0	0	0	...
3	2487200875	20141209T000000	604000.0	4	3.00	1960	5000	1.0	0	0	...
4	1954400510	20150218T000000	510000.0	3	2.00	1680	8080	1.0	0	0	...

5 rows × 21 columns

Código-fonte 1 – Importação das bibliotecas e load do dataset
Fonte: Elaborado pelo autor (2024)

Neste conjunto de dados há 21 colunas ao todo. Para fins didáticos, vamos trabalhar com estas:

- id - ID único para cada casa.
- date - Data da casa a venda.
- price - Preço da casa.
- bedrooms - Número de quartos.
- bathrooms - Número de banheiros, o valor .5 conta como lavabo.
- sqft_living - Área em m² do espaço interior.
- sqft_lot - Área em m² do espaço do terreno.
- floors - Número de andares.
- waterfront - Se tem vista para o mar (1) ou não (0). (categórico).

Para uma análise mais detalhada, veja a definição das colunas no site do Kaggle. Vemos aqui que faz sentido remover as colunas 'id' e 'date' para o modelo, ficando com o total de 7 colunas e 21613 linhas (código-fonte 2).

```
df = df[['id', 'date', 'price', 'bedrooms', 'bathrooms',
'sqft_living', 'sqft_lot', 'floors', 'waterfront']]

colunas_para_remover = ['id', 'date']
df = df.drop(colunas_para_remover, axis=1)

rows, cols = df.shape
print(f'Linhas: {rows}. Colunas: {cols}')
Linhas: 21613. Colunas: 7
```

Código-fonte 2 – Seleciona-se algumas colunas e remove-se outras para iniciar a modelagem
Fonte: Elaborado pelo autor (2024)

Vamos agora separar o conjunto de features da classe nas variáveis x e y, respectivamente (código-fonte 3).

```
x = df.drop('price', axis=1)
y = df['price']
```

Código-fonte 3 – Separação dos dados de features e classe
Fonte: Elaborado pelo autor (2024)

Para evitar que alguns modelos sofram com a diferença de escala, vamos fazer uma normalização dos dados (código-fonte 4)

```
# normalizador
from sklearn.preprocessing import StandardScaler

# normalização dos dados
min_max_scaler = StandardScaler()
x = min_max_scaler.fit_transform(x)
```

Código-fonte 4 – Normalização dos dados com a biblioteca Scikit-Learn
Fonte: Elaborado pelo autor (2024)

Agora os dados estão prontos para serem separados no conjunto de treino e teste. Fazemos da mesma forma que na aula de Classificação (código-fonte 5), com 70% dos dados para o treinamento e 30% para teste.

```
from sklearn.model_selection import train_test_split
```

```
x_train, x_test, y_train, y_test = train_test_split(x, y,
test_size=0.3, random_state=23)

print(f'Exemplos para o treinamento: {len(y_train)}. E para o
teste: {len(y_test)}')
Exemplos para o treinamento: 15129. E para o teste: 6484
```

Código-fonte 5 – Separação dos dados em treino e teste
Fonte: Elaborado pelo autor (2024)

Regressão Linear

Um dos métodos mais antigos e fundamentais para mapear as relações entre variáveis contínuas. Através da construção de um modelo matemático simples e intuitivo, a regressão linear permite prever valores futuros de uma variável dependente com base em uma ou mais variáveis independentes (código-fonte 6).

A regressão linear se baseia na seguinte equação:

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \ldots + \beta_n x_n + \varepsilon$$

Em que:

- **y:** é a variável dependente, cujo valor queremos prever.
- **β_0:** é o intercepto, que representa o valor de y quando todas as variáveis independentes são iguais a zero.
- **β_1, β_2,** ..., **β_n:** são os coeficientes de regressão, que indicam a mudança no valor de y para uma unidade de mudança na respectiva variável independente.
- **x_1, x_2,** ..., **x_n:** são as variáveis independentes, que influenciam o valor da variável dependente.
- **ε:** é o termo de erro, que representa a diferença entre o valor previsto de y e o valor real observado.

Estimação dos Parâmetros: o objetivo da regressão linear é estimar os valores dos parâmetros **β_0, β_1,** ..., **β_n** da equação anterior.

```
# métricas
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_percentage_error

from sklearn.linear_model import LinearRegression
```

```
model = LinearRegression()
model.fit(x_train, y_train)

y_pred = model.predict(x_test)
```

Código-fonte 6 – Importação das métricas e modelo, treinamento e previsão da regressão linear
Fonte: Elaborado pelo autor (2024)

Com o valor obtido de y_pred, vamos calcular 3 métricas: MSE, RMSE e MAPE (código-fonte 7).

```
mse = mean_squared_error(y_test,y_pred, squared=True)
rmse = np.sqrt(mse)
mape = mean_absolute_percentage_error(y_test,y_pred)

print(f'MSE: {mse}\nRMSE: {rmse}\nMAPE: {mape}')

MSE: 58290430743.82154
RMSE: 241434.11263494132
MAPE: 0.3468906172169916
```

Código-fonte 7 – Cálculo das métricas de avaliação para a regressão linear
Fonte: Elaborado pelo autor (2024)

Vemos que há espaço para melhoria pelo MAPE, que está com 34,69%. Por ser uma medida de erro, diferente da acurácia, quanto menor, melhor.

K-NN Regressor

Da mesma forma que fizemos para a tarefa de classificação, vamos aplicar aqui o modelo de vizinhos mais próximos, mas voltado para a regressão. O algoritmo também se baseia no conceito de vizinhança local, prevendo o valor de uma nova instância considerando os valores de seus k vizinhos mais próximos no conjunto de treinamento.

O KNN Regressor se baseia na suposição de que instâncias similares tendem a possuir valores de saída (target) similares. Para realizar uma previsão, o algoritmo considera as k instâncias de treinamento mais próximas (vizinhos) da nova instância de entrada e utiliza seus valores de saída para estimar o valor de saída da nova instância.

Existem diferentes estratégias para prever o valor de saída da nova instância. Uma abordagem comum é calcular a **média** dos valores de saída dos k vizinhos mais próximos. Outra opção é realizar uma **votação**, em que o valor de saída mais frequente entre os vizinhos é considerado a previsão.

Implementação em Python: para implementar o KNN Regressor em Python, utilizaremos a biblioteca scikit-learn (código-fonte 8).

```
from sklearn.neighbors import KNeighborsRegressor

model = KNeighborsRegressor(n_neighbors=7, metric='euclidean')
model.fit(x_train, y_train)
y_pred = model.predict(x_test)

mape = mean_absolute_percentage_error(y_test,y_pred)
print(f'MAPE: {mape}')
MAPE: 0.324655444199613
```

Código-fonte 8 – Implementação para o KNN Regressor
Fonte: Elaborado pelo autor (2024)

Aqui utilizamos 7 vizinhos e a métrica de distância a euclidiana. Observamos que este modelo ficou um pouco melhor que a regressão linear. Faça você alguns testes com outras quantidades de vizinhos!

Support Vector Machines para Regressão (SVR)

Este modelo é uma opção poderosa e versátil para lidar com problemas complexos e alcançar alta precisão. O SVR para regressão busca encontrar uma reta ou hiperplano no espaço de dados que minimize o erro entre as previsões do modelo e os valores reais observados. Em contraste com a regressão linear, que assume uma relação linear entre as variáveis, o SVR permite capturar relações não lineares complexas nos dados.

O SVR utiliza a função de perda de margem máxima para penalizar os pontos de dados que se desviam demais do hiperplano. Essa função busca encontrar um hiperplano que maximize a margem entre os pontos de dados e o hiperplano, garantindo que a maioria dos pontos se encontre em uma região de erro mínimo. Para lidar com dados não lineares, o SVR utiliza variáveis kernels, que transformam os dados em um espaço de dimensão mais alta, em que a relação linear se torna evidente. Diversas variáveis kernels podem ser utilizadas, como o kernel linear, o kernel polinomial e o kernel RBF (Radial Basis Function).

O SVR possui dois parâmetros de regularização importantes: **C** e **ε**. O parâmetro **C** controla a força da penalização por erros, enquanto o parâmetro **ε** define a largura da margem em torno do hiperplano. A escolha adequada desses parâmetros é crucial para o bom desempenho do modelo.

Vejamos o desempenho deste modelo (código-fonte 9).

```
from sklearn.svm import SVR

model = SVR()
model.fit(x_train, y_train)

y_pred = model.predict(x_test)

mape = mean_absolute_percentage_error(y_test,y_pred)

print(f'MAPE: {mape}')
MAPE: 0.4214532788664194
```

Código-fonte 9 – Support Vector Machines para Regressão no conjunto de dados de casas
Fonte: Elaborado pelo autor (2024)

Fazendo alguns testes nos parâmetros do modelo, podemos melhorar um pouco o resultado com estes parâmetros (código-fonte 10). Iremos comentar melhor a técnica utilizada para isso na próxima aula.

```
from sklearn.svm import SVR

model = SVR(kernel='linear', C=100)
model.fit(x_train, y_train)

y_pred = model.predict(x_test)

mape = mean_absolute_percentage_error(y_test,y_pred)

print(f'MAPE: {mape}')
MAPE: 0.32089126628819453
```

Código-fonte 10 – SVR com alguns parâmetros otimizados
Fonte: Elaborado pelo autor (2024)

Regressão de Árvore de Decisão

Também conhecido como árvore de regressão, é um modelo composto por uma série de **nós** e **ramos**, organizados de forma hierárquica. Cada nó representa uma decisão sobre uma variável independente e cada ramo representa o resultado dessa decisão. O processo de construção da árvore de regressão se dá através de um algoritmo recursivo que busca minimizar o erro de previsão a cada passo. O algoritmo divide os dados em subconjuntos com base em valores específicos das

variáveis independentes, utilizando critérios como o ganho de informação ou a variância reduzida.

Para realizar uma previsão, um novo ponto de dados é guiado pela árvore, passando pelos nós e ramos de acordo com seus valores nas variáveis independentes. O valor previsto é determinado pelo valor médio das observações presentes no nó folha em que o novo ponto de dados termina.

A estrutura da árvore de regressão pode ser facilmente visualizada, permitindo uma interpretação intuitiva do processo de decisão e das variáveis mais importantes para o modelo. É possível extrair regras de decisão a partir da árvore de regressão, que indicam as condições que um novo ponto de dados deve atender para pertencer a uma determinada classe ou obter um determinado valor de previsão.

Vejamos uma implementação deste modelo com a biblioteca Scikit-Learn (código-fonte 11).

```
from sklearn.tree import DecisionTreeRegressor

model = DecisionTreeRegressor()
model.fit(x_train, y_train)

y_pred = model.predict(x_test)

mape = mean_absolute_percentage_error(y_test,y_pred)

print(f'MAPE: {mape}')
MAPE: 0.4132546879741096
```

Código-fonte 11 – Implementação da Árvore de Regressão
Fonte: Elaborado pelo autor (2024)

XGBoost

Este modelo é dos mais utilizados na indústria e academia. Ele se baseia no conceito de impulsionamento de árvores, em que diversas árvores de decisão fracas são sequencialmente construídas e combinadas para melhorar o desempenho geral do modelo. Cada árvore aprende com os erros das árvores anteriores, corrigindo suas falhas e refinando a previsão final.

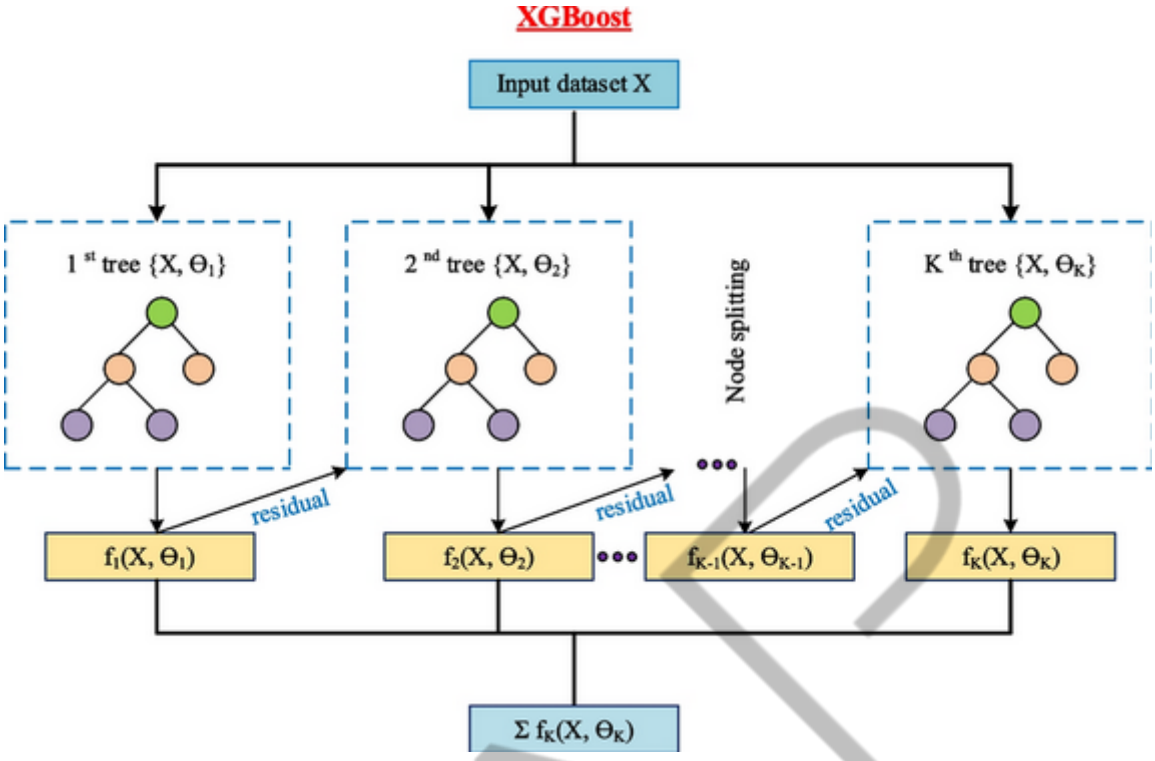

Figura 3 – Representação da Arquitetura do XGBoost
Fonte: Researchgate [s.d.]

O XGBoost utiliza uma **função objetivo** personalizada que combina o erro de previsão com uma **penalização de complexidade**. Essa penalização visa evitar o overfitting, controlando o crescimento das árvores e impedindo que se tornem muito complexas e memorizem os dados de treinamento. O XGBoost utiliza o **algoritmo gradiente** para otimizar os parâmetros do modelo de forma eficiente. Através de cálculos iterativos, o algoritmo busca minimizar a função objetivo, ajustando os parâmetros das árvores de decisão para alcançar a melhor performance.

Vejamos uma implementação em Python do modelo (código-fonte 12).

```
from xgboost import XGBRegressor
model = XGBRegressor()
model.fit(x_train, y_train)

y_pred = model.predict(x_test)

mape = mean_absolute_percentage_error(y_test,y_pred)

print(f'MAPE: {mape}')
MAPE: 0.3115478188030054
```

Código-fonte 12 – Implementação do XGBoost em Python
Fonte: Elaborado pelo autor (2024)

Notamos um resultado muito bom comparado aos outros modelos. Utilizando alguns parâmetros em seu treinamento, podemos ter um resultado um pouco melhor (código-fonte 13).

```
params = {"n_estimators": 100,
          "max_depth": 6,
          "learning_rate": 0.1}

# Treinando o modelo
model = XGBRegressor(**params)
model.fit(x_train, y_train)

y_pred = model.predict(x_test)

mape = mean_absolute_percentage_error(y_test,y_pred)

print(f'MAPE: {mape}')
MAPE: 0.30671574201583884
```

Código-fonte 13 – Otimização de alguns parâmetros no XGBoost
Fonte: Elaborado pelo autor (2024)

MERCADO, CASES E TENDÊNCIAS

O Rabbit R1 é um aparelho que promete ser uma inteligência artificial de bolso: o dispositivo é capaz de realizar compras online, reservar passagens aéreas e solicitar um carro por aplicativo utilizando apenas comando de voz. Ele promete até sugerir uma receita baseando-se em uma foto dos itens da geladeira. Saiba mais aqui.

A IA está em um ponto que é possível criar rostos de pessoas que não existem; esta tecnologia se chama deepfake. Ela permite essa manipulação, principalmente em vídeos, de uma forma bastante convincente, trocando o rosto de pessoas em vídeos e sincronizando movimentos labiais. Veja sobre aqui.

O QUE VOCÊ VIU NESTA AULA?

Nesta aula vimos alguns modelos de regressão, suas definições, conceitos e implementações. Também utilizamos a métrica MAPE, dado seu valor percentual de referência. Caso decida utilizar outra métrica, é importante definir qual será o valor de referência (baseline) para saber se o desenvolvimento dos modelos está melhorando ou não.

Estes modelos são muito usados na indústria e vimos que há possibilidade de melhoria dentro de um mesmo modelo. Na próxima aula, veremos técnicas para aplicar este aprimoramento, além de outras importantes para cientistas de dados aplicarem em seus modelos.

REFERÊNCIAS

CHEN, T.; GUESTRIN, C. **XGBoost:** a scalable tree boosting system. 2016. Disponível em: <http://arxiv.org/abs/1603.02754>. Acesso em: 05 jul. 2024.

CRISTIANINI, N.; RICCI, E. Support Vector Machines: 1992; Boser, Guyon, Vapnik. Em: **Encyclopedia of Algorithms**. Boston: Springer US, 2008. p. 928–932.

GÉRON, A. **Understanding support vector machines**. [s.l.]: O'Reilly Media, 2017.

RESEARCHGATE. **Figure** - available from: Earth Science Informatics. [s.d.] Disponível em: <https://www.researchgate.net/figure/The-XGBoost-algorithm-structure_fig4_378851394>. Acesso em: 05 jul. 2024.

TREHAN, D. **Linear Regression explained**. 2020. Disponível em: <https://pub.towardsai.net/linear-regression-explained-f5cc85ae2c5c>. Acesso em: 04 jul. 2024.

XGBOOST. **XGBoost Documentation — xgboost 2.1.0-dev documentation**. 2024. Disponível em: <https://xgboost.readthedocs.io/en/latest/>. Acesso em: 24 abr. 2024.

PALAVRAS-CHAVE

Palavras-chave: Regressão. Regressão Linear. Regressão de Árvore de Decisão. K-NN Regressor. Support Vector Machine Regression. XGBoost. Pandas. Numpy. Scikit-Learn. Python. Erro Médio Absoluto. Raiz quadrada média do erro. Mean Absolute Percentage Error.