Folha de Dicas: Regressão Linear e Logística

Comparando diferentes tipos de regressão

Nome do Modelo	Descrição	Sintaxe do Código

Regressão linear simples	**Objetivo:** Prever uma variável dependente com base em uma variável independente.
	Prós: Fácil de implementar, interpretar e eficiente para pequenos conjuntos de dados.
	Contras: Não é adequada para relações complexas; propensa a subajuste.
	Equação de modelagem: $y = b_0 + b_1x$
		`from sklearn.linear_model import LinearRegression`
		`model = LinearRegression()`
		`model.fit(X, y)`

Regressão polinomial	**Objetivo:** Capturar relações não lineares entre variáveis.
	Prós: Melhor para ajustar dados não lineares em comparação com a regressão linear.
	Contras: Propensa a sobreajuste com polinômios de alto grau.
	Equação de modelagem: $y = b_0 + b_1x + b_2x^2 + ...$
		`from sklearn.preprocessing import PolynomialFeatures`
		`from sklearn.linear_model import LinearRegression`
		`poly = PolynomialFeatures(degree=2)`
		`X_poly = poly.fit_transform(X)`
		`model = LinearRegression().fit(X_poly, y)`

Regressão linear múltipla	**Objetivo:** Prever uma variável dependente com base em múltiplas variáveis independentes.
	Prós: Considera múltiplos fatores que influenciam o resultado.
	Contras: Assume uma relação linear entre preditores e alvo.
	Equação de modelagem: $y = b_0 + b_1x_1 + b_2x_2 + ...$
		`from sklearn.linear_model import LinearRegression`
		`model = LinearRegression()`
		`model.fit(X, y)`

Regressão logística	**Objetivo:** Prever probabilidades de resultados categóricos.
	Prós: Eficiente para problemas de classificação binária.
	Contras: Assume uma relação linear entre variáveis independentes e log-odds.
	Equação de modelagem: $\log(p/(1-p)) = b_0 + b_1x_1 + ...$
		`from sklearn.linear_model import LogisticRegression`
		`model = LogisticRegression()`
		`model.fit(X, y)`

Funções associadas comumente usadas

Nome da Função/Método	Descrição Breve	Sintaxe do Código
train_test_split	Divide o conjunto de dados em subconjuntos de treinamento e teste para avaliar o desempenho do modelo.	`from sklearn.model_selection import train_test_split` `X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)`

StandardScaler	Padroniza as características removendo a média e escalando para variância unitária.	`from sklearn.preprocessing import StandardScaler`
		`scaler = StandardScaler()`
		`X_scaled = scaler.fit_transform(X)`

log_loss	Calcula a perda logarítmica, uma métrica	`from sklearn.metrics import log_loss` `loss = log_loss(y_true, y_pred_proba)`

Nome da Função/Método	Descrição Breve	Sintaxe do Código
	de desempenho para modelos de classificação.	
mean_absolute_error	Calcula o erro absoluto médio entre os valores reais e previstos.	`from sklearn.metrics import mean_absolute_error` `mae = mean_absolute_error(y_true, y_pred)`
mean_squared_error	Calcula o erro quadrático médio entre os valores reais e previstos.	`from sklearn.metrics import mean_squared_error` `mse = mean_squared_error(y_true, y_pred)`

root_mean_squared_error	Calcula a raiz do erro quadrático médio (RMSE), uma métrica comumente usada para tarefas de regressão.	`from sklearn.metrics import mean_squared_error`
		`import numpy as np`
		`rmse = np.sqrt(mean_squared_error(y_true, y_pred))`

r2_score	Calcula o valor R-quadrado, indicando quão bem o modelo explica a variabilidade da variável alvo.	`from sklearn.metrics import r2_score` `r2 = r2_score(y_true, y_pred)`

Autor(es)

Jeff Grossman
Abhishek Gagneja