Universidade da Beira Interior
Departamento de Informática

Relatório do Projeto Prático 1

Aprendizagem Supervisionada (Regressão Linear)

Elaborado por:

TASSIA DA SILVA DE CARVALHO

MESTRADO EM ENGENHARIA INFORMÁTICA

UC.14469 APRENDIZAGEM AUTOMÁTICA

Professor Doutor HUGO PEDRO PROENÇA

Conteúdo

Lista de Figuras

Acrónimos

LOO-CV Leave-One-Out Cross-Validation

UBI Universidade da Beira Interior

MSE Erro Quadrático Médio

MAE Erro Absoluto Médio

Capítulo

1

Introdução

1.1 Objetivo do Relatório

O presente relatório foi elaborado no âmbito da Unidade Curricular de Aprendizagem Automática do Mestrado em Engenharia Informática da Universidade da Beira Interior (UBI). O objetivo é aplicar técnicas de *machine learning* ao conjunto de dados fornecido, utilizando um modelo de Regressão Linear com múltiplos parâmetros. Este modelo foi escolhido pela sua simplicidade e eficiência na previsão de valores contínuos a partir de variáveis explicativas. A abordagem é particularmente adequada para esta atividade prática, cujo intuito é verificar se os encargos cobrados pelo seguro de saúde podem ser previstos a partir de variáveis como a idade, o sexo, o índice de massa corporal (IMC), o número de crianças, o estado de fumador e a região de residência.

A aprendizagem automática é uma área de grande relevância atualmente, permitindo a criação de modelos preditivos a partir de dados. De acordo com Bishop (2011), "a aprendizagem supervisionada envolve a construção de uma função de mapeamento a partir de um conjunto de dados rotulado para fazer previsões sobre novos exemplos". A escolha da regressão linear como modelo base fundamenta-se na sua simplicidade matemática e facilidade de interpretação. Como sugerido pelo professor, a regressão linear é um dos primeiros métodos a serem explorados na aprendizagem supervisionada, conforme descrito por Mohri et al. (2018). Este método permite entender a relação entre variáveis de forma direta, o que é valioso no contexto da análise de dados do seguro de saúde.

Além da implementação do modelo de regressão linear, este relatório abrange a análise exploratória do conjunto de dados, que é uma etapa crucial na compreensão das características dos dados. A análise exploratória envolve a vi-

sualização e o resumo das características principais do conjunto de dados, permitindo identificar padrões, tendências e possíveis anomalias. As visualizações ajudam a orientar a escolha dos modelos e as transformações a serem aplicadas aos dados.

Neste estudo, também será aplicada a descida de gradiente, um algoritmo fundamental para a otimização dos parâmetros do modelo. A descida de gradiente é um método iterativo que busca minimizar a função de custo, ajustando os parâmetros do modelo para encontrar os melhores ajustes aos dados. Este algoritmo é amplamente utilizado em regressão e redes neuronais, sendo essencial para o treino de modelos em *machine learning*.

Para validar a eficácia do modelo, serão utilizadas técnicas de validação cruzada, nomeadamente K-fold Cross-Validation e LOO-CV. A escolha dessas técnicas é especialmente relevante dado que o conjunto de dados disponível contém apenas 1338 registos. A LOO-CV, que consiste em usar uma instância como conjunto de teste enquanto utiliza todas as outras para o treino, permite maximizar a utilização dos dados, melhorando a precisão e fiabilidade dos resultados. Como afirmado por Mohri et al. (2018), "a validação cruzada é uma técnica que permite estimar a performance do modelo em dados não vistos, proporcionando uma avaliação mais robusta do seu desempenho".

Este estudo não aborda um caso real, mas sim um exercício académico focado no desenvolvimento de competências de modelagem preditiva. O conjunto de dados utilizado está disponível no site "Kaggle.com" (Medical Cost Personal Dataset) e contém informações genéricas de pessoas, permitindo a análise das variáveis que influenciam os custos dos seguros de saúde.

Capítulo

2

Fundamentação Teórica

2.1 Introdução

Neste capítulo, será realizada uma abordagem dos principais conceitos e técnicas fundamentais para o desenvolvimento de modelos preditivos baseados em regressão linear. Inicialmente, é discutida a regressão linear, destacando como esse método é utilizado para modelar a relação entre variáveis independentes e a variável dependente. Em seguida, explora-se o papel central da função de custo, que mede o erro entre as previsões do modelo e os valores reais, permitindo a otimização dos parâmetros da regressão. A seguir, é introduzida a técnica de descida de gradiente, que minimiza a função de custo ao ajustar iterativamente os parâmetros do modelo. Para garantir a confiabilidade do modelo, são apresentados os métodos de validação cruzada κ-fold e validação (LOO-CV), que avaliam o desempenho do modelo em diferentes divisões do conjunto de dados, buscando evitar o overfitting e proporcionar uma melhor generalização. Por fim, o conceito de regularização é abordado como uma técnica eficaz para controlar o overfitting, garantindo um modelo mais robusto e adequado para predições em dados reais.

2.2 Regressão Linear

A regressão linear é uma técnica que busca modelar a relação entre uma variável dependente y e uma ou mais variáveis independentes X, assumindo que essa relação pode ser expressa como uma combinação linear das variáveis independentes:

$$y = \theta_0 + \theta_1 X_1 + \theta_2 X_2 + ... + \theta_n X_n + \epsilon$$

Onde:

- y representa a variável dependente
- θ_0 é o termo constante (ou intercepto)
- $\theta_1, \theta_2, ..., \theta_n$ são os coeficientes que representam a influência de cada variável independente
- X são variáveis independentes (features) do modelo.
- ϵ é o erro associado à predição, assumido como aleatório

A função, nesse caso, visa minimizar a soma dos erros quadráticos, ou seja, reduzir ao máximo o erro entre o valor predito e o valor real dos encargos de seguro.

Neste projeto prático, o objetivo é verificar se os encargos cobrados pelo seguro saúde podem ser previstos a partir das demais variáveis independentes, obtendo então a função:

$$h_\theta(xi) = \theta_0 + \theta_1 age + \theta_2 sex + \theta_3 bmi + \theta_4 children + \theta_5 smoker + \theta_6 region$$

Durante a pesquisa, surgiu uma dúvida referente à presença do erro ϵ. Foi identificado que o seu uso depende da forma como o modelo de regressão está a abordar o problema, conforme a explicação abaixo:

- A equação teórica da regressão linear inclui ϵ para representar o erro aleatório. Esse termo é crucial para indicar que nem todas as variáveis podem ser explicadas apenas pelas features (variáveis independentes) e que existem outros fatores não observados que influenciam a variável dependente y.
- Em termos práticos, geralmente a concentração está na função de predição ao treinar modelos. Esta função representa a previsão que o modelo faz com base nas variáveis de entrada, sem incluir explicitamente o erro.
- O termo ϵ não aparece na implementação do modelo do presente projeto prático, pois ele é uma representação da variabilidade não explicada, e neste caso o foco está em prever y a partir das features. No entanto, é importante entender que o erro é uma parte essencial da análise

e do ajuste de modelos, pois reflete a capacidade do modelo de generalizar e fazer previsões precisas. Contudo, neste caso, ao ajustar o modelo com descida de gradiente, a busca é para encontrar os parâmetros θ que minimizam a diferença entre as previsões e os valores reais de y. O erro ϵ é implícito neste processo, pois o modelo aprende a ajustar θ para reduzir as previsões incorretas.

De acordo com Bishop (2011), a regressão linear é um modelo probabilístico no qual se assume que os erros ϵ seguem uma distribuição normal, com média zero e variância constante. Isso permite que os coeficientes θ possam ser inferidos através do método de mínimos quadrados ordinários.

Citando diretamente Bishop (2011), "a regressão linear é particularmente útil em problemas onde as relações entre as variáveis preditoras e a variável resposta podem ser aproximadas por funções lineares, sendo possível expandir seu poder preditivo com a inclusão de termos polinomiais ou interações entre as variáveis".

A regressão linear, além de sua simplicidade e utilidade em modelar relações lineares entre variáveis preditoras e a variável resposta, depende de uma abordagem robusta para ajustar os seus parâmetros de forma eficiente. Nesse contexto, a função de custo desempenha um papel central, sendo a responsável por quantificar o erro entre as previsões do modelo e os valores reais. É por meio da minimização dessa função, que mede o desvio entre as previsões do modelo e os dados observados, que a regressão linear busca encontrar os melhores parâmetros. Assim, a função de custo orienta o ajuste do modelo, tornando-o mais preciso ao longo do processo de treinamento. A seguir, serão apresentados os conceitos que envolvem a função de custo e a sua importância no ajuste dos parâmetros.

2.3 Função de Custo na Regressão Linear

Como base na bibliografia sugerida, Bishop (2011) afirma: "A função de custo é usada para medir o erro do modelo ao prever os dados de treino, sendo um elemento essencial para guiar o processo de ajuste dos parâmetros do modelo". Ou seja, a função de custo é uma métrica que quantifica quão bem um modelo se ajusta aos dados.

O objetivo principal ao usar uma função de custo na regressão linear é encontrar os parâmetros do modelo (os coeficientes θ) que minimizam essa função de custo. Isso é realizado por meio de algoritmos de otimização, como a descida de gradiente, que será explicado a seguir. Durante o processo de treino, o algoritmo ajusta os parâmetros do modelo iterativamente, movendo-

se na direção oposta ao gradiente da função de custo até que a minimização seja alcançada.

No caso da regressão linear, a função de custo frequentemente utiliza o MSE, que é a média dos quadrados das diferenças entre os valores reais e os valores preditos pelo modelo:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

Onde:

- n é o número total de observações no conjunto de dados.
- y_i representa o valor real observado para a i-ésima observação.
- $\hat{y}_i$ é o valor predito pelo modelo para a i-ésima observação.

2.3.1 Interpretação do MSE

O MSE calcula a média das diferenças quadráticas entre os valores reais e os valores preditos. O uso do quadrado das diferenças tem algumas implicações importantes:

- Penalização de Erros Maiores: Ao elevar as diferenças ao quadrado, o MSE penaliza erros maiores de forma mais severa do que erros menores. Isso significa que um modelo que comete grandes erros terá um MSE mais alto, indicando que ele não está a performar bem.
- Sensibilidade a Outliers: O MSE é sensível a outliers, pois valores extremos influenciam a média quadrática, o que pode distorcer a avaliação do modelo. Em casos onde a presença de outliers é significativa, pode ser mais apropriado usar outras métricas de custo, como o Erro Absoluto Médio Erro Absoluto Médio (MAE).

Ele fornece uma medida de quão longe as previsões do modelo estão dos valores reais. O objetivo é minimizar essa função de custo durante o treinamento do modelo, o que implica que o modelo está a aprender a prever melhor os valores reais.

A função de custo é um elemento central na aprendizagem de máquinas, permitindo a avaliação e a otimização de modelos preditivos. Na regressão linear, o MSE serve como uma métrica eficaz para guiar o ajuste dos parâmetros do modelo, buscando a melhor aproximação dos valores reais através

da minimização das diferenças quadráticas. Essa abordagem não apenas melhora a precisão do modelo, mas também permite uma análise mais profunda do relacionamento entre as variáveis preditoras e a variável dependente.

Um dos principais métodos utilizados para ajustar os modelos de regressão linear é a descida de gradiente, uma técnica de otimização iterativa que minimiza a função de custo ao ajustar os parâmetros do modelo. A descida de gradiente funciona calculando o gradiente da função de custo e atualizando os pesos na direção oposta do gradiente, com o objetivo de encontrar o mínimo global (ou local) da função.

Esta técnica é amplamente utilizada em vários algoritmos de aprendizagem supervisionada, e a sua eficácia depende de parâmetros como a taxa de aprendizagem e o número de iterações. Ao aplicar a descida de gradiente, garantimos que o modelo converge para um conjunto ideal de parâmetros que minimizam o erro entre as previsões e os valores reais. Este processo é particularmente relevante no contexto de regressão linear, que foi o foco principal da análise neste trabalho. Abaixo é conceituado o método da descida de gradiente e o seu funcionamento.

2.4 Descida de Gradiente

O método de descida de gradiente é uma técnica iterativa utilizada para minimizar a função de custo da regressão linear. De acordo com Mohri et al. (2018, pp. 98-102), trata-se de um algoritmo de otimização que ajusta os coeficientes θ na direção do gradiente negativo da função de custo, atualizando os parâmetros de acordo com a seguinte fórmula:

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta)$$

Aqui, α representa a taxa de aprendizagem, ou seja, o tamanho do passo que damos na direção do gradiente em cada iteração, e $J(\theta)$ é a função de custo, definida como o erro quadrático médio entre as predições do modelo e os valores reais.

A eficácia do método de descida de gradiente depende da escolha apropriada da taxa de aprendizagem α, como descrito por Mohri et al. (2018, p. 99): "Escolher um valor de α muito pequeno resulta em uma convergência muito lenta, enquanto valores grandes demais podem fazer com que o algoritmo não converja para o mínimo global."

Quanto à taxa de aprendizagem, foi observado que ela pode ser representada tanto por α quanto por λ, dependendo do contexto em que se trabalha. Abaixo segue um resumo de quando cada símbolo deve ser usado:

- α: Quando se refere à curva de aprendizagem, muitas vezes é analisado como a performance do modelo melhora em relação ao número de iterações, e a taxa de aprendizagem α pode afetar essa performance.

- λ: Embora não seja diretamente associado à curva de aprendizagem, λ pode influenciar a forma como o modelo aprende, especialmente em relação à complexidade do modelo e à performance em dados não vistos.

Algumas referências fazem a seguinte divisão:

- Taxa de Aprendizagem: Na literatura de machine learning, a notação mais comum para a taxa de aprendizagem é α. A maioria dos algoritmos de otimização, como a descida de gradiente, geralmente usa α para se referir a esse parâmetro. Exemplos incluem:

  - Bishop, C. M. (2011) em "Pattern Recognition and Machine Learning".
  - Goodfellow, I., Bengio, Y., Courville, A. (2016) em "Deep Learning".

- Regularização: O uso de λ é comum em algoritmos que implementam regularização, como Lasso e Ridge regression. Referências de exemplo que mencionam o uso de λ incluem:

  - Hastie, T., Tibshirani, R., Friedman, J. (2009) em "The Elements of Statistical Learning".

Se a referência é especificamente sobre a taxa de aprendizagem na sua curva de aprendizagem, use α. Se está sendo discutida a regularização, então use λ.

Para finalizar, a descida de gradiente se mostrou uma técnica poderosa para encontrar os parâmetros ótimos em problemas de regressão linear, ajustando continuamente os valores de θ para minimizar a função de custo. Com a sua capacidade de lidar com conjuntos de dados de alta dimensionalidade, essa abordagem se tornou um dos métodos mais utilizados no aprendizado de máquina supervisionado. Agora que os conceitos de regressão linear, função de custo e descida de gradiente foram abordados, é fundamental compreender e apresentar os métodos de validação do modelo, como a validação cruzada κ-fold e o LOO-CV, para avaliar a performance e a generalização do modelo.

2.5 Validação Cruzada κ-fold

A validação cruzada κ-fold é uma técnica robusta para avaliar o desempenho de modelos de aprendizado supervisionado. Como descrito por Bishop (2011), esse método divide o conjunto de dados em κ subconjuntos, ou "folds". O modelo é treinado em $\kappa-1$ subconjuntos e testado no subconjunto restante. Esse processo é repetido κ vezes, de modo que cada subconjunto funcione como conjunto de teste exatamente uma vez.

A principal vantagem desse método é sua capacidade de mitigar o problema de overfitting, já que ele permite que o modelo seja testado em diferentes subconjuntos do conjunto de dados. Como observa Bishop (2011), "a validação cruzada κ-fold fornece uma estimativa mais confiável do desempenho do modelo em dados não vistos, uma vez que o processo de validação é feito em vários subconjuntos distintos".

2.6 Validação LOO-CV

A validação leave-one-out (LOO-CV) é uma abordagem robusta para testar modelos preditivos, especialmente em conjuntos de dados mais pequenos, como o dataset usado, que possui 1338 registos.

A validação LOO-CV divide o conjunto de dados em n subconjuntos, onde n corresponde ao número total de registos no dataset. Para cada iteração, uma única instância do conjunto de dados é separada para ser usada como conjunto de validação, enquanto as restantes são usadas para treinar o modelo. Este processo é repetido para todas as instâncias, garantindo que cada registo seja usado para validação uma vez. Esta técnica é especialmente útil em datasets pequenos, pois todas as observações são testadas, uma a uma, garantindo uma avaliação mais precisa da capacidade de generalização do modelo.

A vantagem da validação leave-one-out está na fidelidade e fiabilidade dos resultados obtidos, uma vez que cada instância do conjunto de dados contribui para o processo de validação. No entanto, o custo computacional desta técnica é elevado, uma vez que o modelo precisa ser treinado n vezes (1338 vezes, no caso deste dataset).

2.7 Regularização e Overfitting

Um dos principais desafios em aprendizado supervisionado é evitar o overfitting, ou seja, quando o modelo se ajusta demasiadamente aos dados de treinamento, capturando até mesmo o ruído e perdendo sua capacidade de ge-

neralização. Uma das formas mais comuns de mitigar o overfitting é através da regularização, que introduz um termo de penalização na função de custo para reduzir a complexidade do modelo.

Como discutido por Mohri et al. (2018, p. 146), "a regularização é uma técnica eficaz para evitar overfitting, especialmente quando o número de variáveis preditoras é alto em relação ao número de amostras disponíveis".

Com base nos conceitos discutidos ao longo deste capítulo, foi possível estabelecer uma sólida compreensão da regressão linear e dos mecanismos que garantem a otimização e validação dos modelos preditivos. A função de custo, juntamente com a descida de gradiente, formam a base para o ajuste adequado dos parâmetros, enquanto as técnicas de validação cruzada e regularização se mostraram essenciais para garantir que o modelo tenha um bom desempenho em dados não vistos e evite problemas de overfitting. Essas ferramentas proporcionam um framework robusto para a criação de modelos preditivos eficientes, permitindo maior confiança na capacidade de generalização e precisão das previsões geradas. A partir dessas fundações, é possível avançar no desenvolvimento de modelos mais complexos e na implementação de soluções em problemas práticos.

Capítulo

3

Metodologia

3.1 Introdução

Neste capítulo, é apresentada a metodologia adotada para o desenvolvimento deste estudo, com foco na análise preditiva. A seção inicial descreve o conjunto de dados utilizado, seguido por uma análise exploratória. Em seguida, apresenta-se a implementação do modelo de regressão linear, abordando a técnica de descida de gradiente. Por fim, são apresentadas as abordagens de validação.

3.2 Descrição do Conjunto de Dados

O conjunto de dados utilizado contém informações de 1338 indivíduos com as seguintes variáveis:

- Idade (Age): Idade do beneficiário do seguro;
- Sexo (Sex): Sexo do beneficiário;
- IMC (BMI): Índice de Massa Corporal;
- Número de crianças (Children): Número de dependentes;
- Fumador (Smoker): Se o beneficiário é fumador ou não;
- Região (Region): Localização geográfica do beneficiário;
- Encargos (Charges): Valor cobrado pelo seguro de saúde (variável a prever).

A variável dependente, Encargos, é contínua e será modelada utilizando a técnica de regressão linear. As demais são as variáveis independentes.

3.3 Análise Exploratória de Dados

Este tópico será dividido conforme a ficha do projeto prático, com a apresentação do código construído para cada solicitação. Nas figuras, as instruções são comentadas, dando a finalidade de cada comando.

a. Conversão de todos os dados categóricos em valores numéricos

b. Checagem de nulos

Neste trecho de código, foram utilizadas as bibliotecas ‘csv‘ e ‘numpy‘ para a leitura e manipulação dos dados. A biblioteca ‘csv‘ permite que o modelo manipule arquivos no formato CSV (Comma-Separated Values), facilitando a extração de informações de maneira estruturada. Já a biblioteca ‘numpy‘ é utilizada para a criação de arrays multidimensionais, o que proporciona uma manipulação eficiente e rápida dos dados. O objetivo da função *load dataset* é carregar o conjunto de dados a partir de um arquivo CSV, realizando o tratamento de colunas categóricas, como sexo, fumador e região, por meio da conversão em valores numéricos, utilizando a técnica de one-hot encoding, adequando-os para análises estatísticas. Além disso, a função inclui uma verificação de valores nulos, que ignora linhas com dados ausentes, garantindo a integridade dos dados utilizados nos modelos de regressão subsequentes.

```
##############################################################################################
#   Função para Carregar Dados com Tratamento das Colunas Categoricas e verificação de Nulos
##############################################################################################
def load_dataset(path):
    # Mapeamento para transformar a coluna 'region' em valores numéricos
    region_mapping = {'northwest': 0, 'northeast': 1, 'southeast': 2, 'southwest': 3}

    with open(path) as csv_file:
        csv_reader = csv.reader(csv_file, delimiter=',')
        next(csv_reader)  # Pula o cabeçalho
        X = []  # Lista de features (idade, sexo, etc.)
        y = []  # Lista de valores dependentes (encargos)

        for row in csv_reader:
            # Verificação de valores nulos
            if any(value == '' for value in row):
                continue  # Ignora a linha se algum valor estiver nulo

            # Conversão para números: por ex., 1 para masculino, 0 para feminino
            sexo = 1 if row[1] == "male" else 0
            fumante = 1 if row[4] == "yes" else 0

            # Converte a região usando o dicionário de mapeamento
            regiao = region_mapping.get(row[5], -1)  # Usando -1 como valor padrão se a região não for válida

            # Adiciona as variáveis (features) e o viés (termo constante 1 para 00)
            X.append([float(row[0]), sexo, float(row[2]), float(row[3]), fumante, regiao, 1])
            y.append(float(row[6]))  # O alvo (encargos)

    X = np.asarray(X)  # Converte para numpy array
    y = np.asarray(y)  # Converte para numpy array
```

Figura 3.1: Pré-processamento dos dados

c. Obtenção dos histogramas de cada feição

- Gráficos de barras

- Estimativas de densidade

```
###############################################################################
#   Função para obtenção do Gráficos de barras e Densidade
###############################################################################
def exploratory_data_analysis_histogram(X, y):
    feature_mapping = {0: 'age', 1: 'sex', 2: 'bmi', 3: 'children', 4: 'smoker', 5: 'region', 6: 'charges'}
    # Histograma de cada feature
    # Loop através de cada feature no conjunto de dados
    for i in range(X.shape[1]):
        # Criação de uma nova figura para os gráficos
        plt.figure(figsize=(12, 5)) # Tamanho da figura


        # Gráfico de barras - Histograma
        plt.subplot(1, 2, 1)  # Cria um subplot na primeira coluna
        plt.hist(X[:, i], bins=20, alpha=0.6, color='g', edgecolor='black')
        plt.title(f'Histograma da Feature: {feature_mapping[i]}')  # Título do histograma
        plt.xlabel(feature_mapping[i])  # Rótulo do eixo X
        plt.ylabel('Frequência')  # Rótulo do eixo Y

        # Estimativa de densidade - Gráfico de densidade
        plt.subplot(1, 2, 2)  # Cria um subplot na segunda coluna
        sns.kdeplot(X[:, i], color='b', linewidth=2)  # Plota a curva de densidade
        plt.title(f'Estimativa de Densidade da Feature: {feature_mapping[i]}')  # Título da densidade
        plt.xlabel(feature_mapping[i])  # Rótulo do eixo X
        plt.ylabel('Densidade')  # Rótulo do eixo Y

        # Ajusta o layout para que os gráficos não se sobreponham
        plt.tight_layout()
        plt.show()  # Exibe os gráficos gerados
```

Figura 3.2: Função para obtenção dos gráficos

A obtenção do histograma de cada característica é uma etapa crucial na análise exploratória de dados, pois permite visualizar a distribuição de cada variável do conjunto de dados.

Os gráficos de barras ajudam a compreender como os dados estão distribuídos em termos de frequência, facilitando a identificação de tendências, como a centralização dos valores ou a presença de assimetrias, o que pode sugerir dados fora do padrão ou distribuídos de forma não uniforme.

As estimativas de densidade, por sua vez, oferecem uma visualização suavizada das distribuições de probabilidade das características, auxiliando na percepção de padrões que poderiam passar despercebidos nos gráficos de barras, fornecendo uma visão mais clara sobre a forma e o comportamento dos dados. Juntos, estes métodos proporcionam uma visão abrangente sobre as características de cada variável, contribuindo para a correta preparação dos dados antes de aplicar modelos de aprendizagem de máquina.

d. Análise da correlação entre as características:

- Observando os gráficos de dispersão entre pares de características.
- Observando os gráficos de dispersão entre cada característica e a variável dependente.

A análise de correlação entre características é uma etapa fundamental para entender as relações entre as variáveis de um conjunto de dados. Ao observar os gráficos de dispersão entre pares de características, é possível identificar como as variáveis independentes se relacionam entre si, o que ajuda a detectar possíveis colinearidades — situações em que duas ou mais variáveis estão altamente correlacionadas. A colinearidade pode influenciar negativamente o desempenho de modelos preditivos, especialmente em algoritmos de regressão, pois pode levar à redundância de informação. Já ao observar a relação entre cada característica e a variável dependente, é possível identificar o impacto direto de cada variável sobre o alvo (no caso, os encargos). Essa visualização ajuda a identificar padrões de dependência, como tendências lineares ou não lineares, e também a verificar a relevância de cada característica para o modelo, auxiliando no processo de seleção de variáveis e no aprimoramento do modelo preditivo.

```
###############################################################################
#   Função para obtenção do Correlação
###############################################################################
def exploratory_data_analysis_correlation(X, y):
    feature_mapping = {0: 'age', 1: 'sex', 2: 'bmi', 3: 'children', 4: 'smoker', 5: 'region', 6: 'charges'}

    # a) Observando os gráficos de dispersão entre pares de características (features independentes)
    # O objetivo aqui é ver como as características se relacionam umas com as outras
    for i in range(X.shape[1]):
        for j in range(i+1, X.shape[1]):  # Compara cada par de características sem repetição
            # Plota o gráfico de dispersão entre a característica 'i' e 'j'
            plt.scatter(X[:, i], X[:, j], alpha=0.6)
            plt.title(f'Feature {feature_mapping[i]} vs Feature {feature_mapping[j]}')  # Título do gráfico
            plt.xlabel(f'Feature {feature_mapping[i]}')  # Nome da característica no eixo X
            plt.ylabel(f'Feature {feature_mapping[j]}')  # Nome da característica no eixo Y
            plt.show()  # Mostra o gráfico de dispersão

    # b) Observando os gráficos de dispersão entre cada característica e a variável dependente 'charges' (encargos)
    # Isso ajuda a identificar como cada característica influencia os encargos
    for i in range(X.shape[1]):
        # Plota o gráfico de dispersão entre a característica 'i' e a variável dependente 'y' (encargos)
        plt.scatter(X[:, i], y, alpha=0.6)
        plt.title(f'Feature {feature_mapping[i]} vs Encargos')  # Título do gráfico
        plt.xlabel(f'Feature {feature_mapping[i]}')  # Nome da característica no eixo X
        plt.ylabel('Encargos')  # A variável dependente no eixo Y (encargos)
        plt.show()  # Mostra o gráfico de dispersão
```

Figura 3.3: Função para correlacionar as features

3.4 Implementação do Algoritmo de Regressão Linear com a descida gradiente

O modelo de regressão linear foi implementado utilizando o algoritmo de descida de gradiente para a otimização dos parâmetros. O script Python utilizado realiza a predição e exibe gráficos de evolução da função de custo ao longo das iterações. Até ao momento, são conhecidas as definições previamente es-

tabelecidas no capítulo de fundamentação teórica, onde foram detalhados os conceitos de Regressão Linear e o papel fundamental da descida do gradiente na otimização dos parâmetros do modelo e a relação da função de custo.

A seguir, descreve-se a aplicação da descida de gradiente para o modelo, focando na interação entre a função de custo e o método de descida de gradiente por etapas:

1. **Inicialização dos Parâmetros**: Os parâmetros do modelo θ são inicializados aleatoriamente;

2. **Cálculo do Gradiente**: Em cada iteração, o gradiente da função de custo em relação aos parâmetros é calculado:

$$\frac{\partial J(\theta)}{\partial \theta_j} = \frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i) x_{ij}$$

   Onde x_{ij} é a j-ésima feature da i-ésima amostra

3. **Atualização dos Parâmetros**: Os parâmetros são actualizados na direcção oposta ao gradiente,para minimizar a função de custo. A atualização é feita usando a fórmula:

$$\theta_j := \theta_j - \alpha \frac{\partial J(\theta)}{\partial \theta_j}$$

   Onde: α é a taxa de aprendizado, que determina o tamanho do passo que damos na direção oposta ao gradiente.

4. **Iteração**: Estas etapas são repetidas até atingir um critério de paragem, como uma mudança mínima na função de custo ou um número máximo de iterações.

O script abaixo utiliza uma abordagem iterativa, onde, a cada iteração, ajusta os parâmetros da regressão linear (os coeficientes θ) com base no erro calculado, movendo-se na direção do gradiente da função de custo que se baseou no MSE.

3.4.1 Visualização da Implementação do Modelo de Regressão Linear

As figuras a seguir ilustram componentes essenciais do modelo de regressão linear implementado com o algoritmo de descida de gradiente.

Função de Custo: A primeira imagem apresenta a função de custo, que quantifica o desempenho do modelo ao comparar as previsões feitas com os valores reais. A função de custo escolhida, o MSE, é fundamental para a otimização dos parâmetros, pois o algoritmo de descida de gradiente busca minimizá-la durante o treino.

Descida Gradiente para Regressão Linear: A segunda imagem descreve o processo de descida de gradiente, destacando como os parâmetros do modelo são ajustados iterativamente com base no gradiente da função de custo. Este processo é crucial para encontrar os valores ideais de θ, que representam os coeficientes da regressão.

Parâmetros de Entrada: A terceira imagem detalha os parâmetros de entrada utilizados no modelo, que incluem as variáveis independentes (features) e a variável dependente (target). Compreender esses parâmetros é vital, pois eles influenciam diretamente as previsões feitas pelo modelo.

Estas representações visuais oferecem uma compreensão clara dos mecanismos subjacentes ao modelo de regressão linear, permitindo uma análise mais profunda do seu funcionamento e eficácia.

```
########################################################################
#   Função de Custo J(θ) - agora com múltiplos parâmetros
########################################################################
def J(X, y, theta):
    preds = np.matmul(X, theta)  # Previsões da equação linear
    error = preds - y  # Diferença entre as previsões e os valores reais
    return np.sum(error ** 2) / (2 * len(y))  # Cálculo do Mean Squared Error (MSE)

    #Poderia ter sido utilizado o Root Mean Square Error (RMSE), o que é uma métrica válida.
    #Contudo, o cálculo do erro quadrático médio regular (MSE) costuma ser mais comum na
    #otimização com descida do gradiente, porque RMSE inclui uma raiz quadrada que pode complicar
    #o cálculo dos gradientes.
```

Figura 3.4: Função de Custo

```
#########################################################################
#   Método de Descida Gradiente para Regressão Linear
#########################################################################
def gradient_descent(theta, lr, tol, X, y):
    it = 0
    Js = []  # Lista para armazenar a evolução da função de custo
    while True:
        # Calcula o vetor de previsões
        preds = np.dot(X, theta)
        # Erro da previsão em relação aos valores reais
        error = preds - y
        # Gradiente da função de custo
        gradient = np.dot(X.T, error) / len(y)  # Operação vetorizada
        # Armazena o valor anterior de theta
        theta_old = np.copy(theta)
        # Atualiza theta com o gradiente
        theta = theta - lr * gradient
        # Calcula a diferença entre os valores antigos e novos de theta
        delta = np.sum(np.abs(theta - theta_old))
        # Calcula e armazena a função de custo
        Js.append(J(X, y, theta))
        # Imprime o status atual
        print(f"[{it}] J=%.5f, theta={theta}, delta={delta}" % Js[-1])
        # Gráfico da evolução do custo
        plt.figure(1)
        plt.plot(range(len(Js)), Js, '-bo')  # Gráfico de linha da função de custo
        plt.title("Evolução do Custo (J)")
        plt.xlabel("Iterações")
        plt.ylabel("Custo (J)")
        plt.grid(True)
        plt.pause(0.1)  # Pausa para que o gráfico seja atualizado em tempo real
        # Verifica critério de convergência
        if delta < tol:
            break

        it += 1

    return theta, Js
```

Figura 3.5: Descida Gradiente para Regressão Linear

```
##########################################################################
#   Main
##########################################################################
learning_rate = 0.0001  # Taxa de aprendizado para descida de gradiente
tolerance = 0.01    # Tolerância para convergência

# Carrega o dataset
X, y = load_dataset('insurance.csv')

# Inicializa theta aleatoriamente
theta = np.random.rand(X.shape[1])

# Treina o modelo com descida de gradiente
theta_gd, evolution_J = gradient_descent(theta, learning_rate, tolerance, X, y)
```

Figura 3.6: Parâmetros de Entrada

3.5 Validação do Modelo

No processo de desenvolvimento de modelos preditivos, a validação do modelo desempenha um papel crucial para garantir a sua eficácia e generalização em dados não vistos. No item 4 e 5 da ficha do projeto prático foi solicitado que fosse implementada a validação κ-fold e que fossem feitas análises de desenho com diferentes kappas, que está representado no item abaixo; adicionalmente, foi implementado o método de validação LOO-CV.

3.5.1 Validação κ-fold

Neste estudo, a validação κ-fold foi implementada com o objetivo de avaliar a robustez do modelo de regressão linear. Foram testados diferentes valores de κ (3, 5, 10, 15, 50, 100, 200, 400, 800 e 1338), com o último representando a totalidade das 1338 instâncias disponíveis no conjunto de dados. A escolha de múltiplos valores de κ permitiu analisar como o desempenho do modelo varia com a quantidade de subdivisões dos dados, oferecendo insights sobre a estabilidade e a fiabilidade das previsões. Cada execução do κ-fold envolveu a divisão do conjunto de dados em κ subconjuntos, em que o modelo foi treinado em κ - 1 folds e testado no fold restante. Esta abordagem proporcionou uma avaliação abrangente da generalização do modelo.

```
########################################################################
#   Função para validar usando k-fold cross-validation
########################################################################
def k_fold_validation(X, y, k, lr, tol):
    # Cria uma instância do KFold para dividir os dados em 'k' partes
    kf = KFold(n_splits=k)
    errors = []  # Lista para armazenar os erros (RMSE) de cada iteração

    # Loop sobre cada divisão dos dados, onde 'train_index' e 'test_index' são os índices dos dados
    for train_index, test_index in kf.split(X):
        # Divide os dados em conjuntos de treinamento e teste
        X_train, X_test = X[train_index], X[test_index]  # Conjunto de treinamento e teste de características
        y_train, y_test = y[train_index], y[test_index]  # Conjunto de treinamento e teste de rótulos

        # Inicializa os parâmetros do modelo (theta) aleatoriamente
        theta = np.random.rand(X_train.shape[1])  # Cria um vetor de parâmetros aleatório do tamanho das características

        # Executa o algoritmo de descida de gradiente para ajustar os parâmetros usando o conjunto de treinamento
        theta, _ = gradient_descent_notplt(theta, lr, tol, X_train, y_train)  # Atualiza 'theta' com os melhores parâmetros

        # Calcula as previsões no conjunto de teste usando os parâmetros ajustados
        preds = np.matmul(X_test, theta)  # Previsões no conjunto de teste
        # Calcula o erro (RMSE) entre as previsões e os valores reais do conjunto de teste
        error = np.sqrt(np.mean((preds - y_test) ** 2))  # Erro quadrático médio da previsão
        errors.append(error)  # Adiciona o erro à lista de erros

    # Retorna a média e o desvio padrão dos erros obtidos em todas as iterações
    return np.mean(errors), np.std(errors)  # Média e desvio padrão dos RMSEs
```

Figura 3.7: Função para Validação κ-fold

3.5.2 Validação LOO-CV

Além da validação κ-fold, foi implementada a validação Leave-One-Out (LOO-CV), que consiste numa abordagem extrema de validação cruzada. Neste método, cada instância do conjunto de dados é utilizada uma vez como conjunto de teste, enquanto o restante serve como conjunto de treino. O LOO-CV é particularmente valioso para conjuntos de dados pequenos, pois maximiza a utilização de dados para o treino e assegura uma avaliação rigorosa do modelo. Ao aplicar o LOO-CV, garantimos que cada ponto de dados contribui para a validação, o que é fundamental para entender a capacidade de generalização do modelo em cenários com poucos dados disponíveis.

```
##########################################################################
#   Função para validar usando LeaveOneOut()
##########################################################################
def leave_one_out_validation(X, y, lr, tol):
    loo = LeaveOneOut()
    errors = []

    for train_index, test_index in loo.split(X):
        X_train, X_test = X[train_index], X[test_index]
        y_train, y_test = y[train_index], y[test_index]

        # Inicializa theta aleatoriamente
        theta = np.random.rand(X_train.shape[1])

        # Executa o algoritmo de descida de gradiente
        theta, _ = gradient_descent_notplt(theta, lr, tol, X_train, y_train)

        # Calcula o erro no conjunto de teste
        preds = np.matmul(X_test, theta)
        error = np.sqrt(np.mean((preds - y_test) ** 2))  # RMSE no conjunto de teste
        errors.append(error)

    return np.mean(errors), np.std(errors)
```

Figura 3.8: Validação LOO-CV

3.5.3 Plotagem dos Resultados de Validação

```
########################################################################
#   Executa validação para diferentes valores de k e LOOCV.
########################################################################
def validate_models(X, y, lr, tol):
    k_values = [3, 5, 10, 15, 50, 100, 200, 400, 800, 1338]  # Kappa values
    results = []

    # Executa validação k-fold para diferentes valores de k
    for k in k_values:
        mean_error, std_error = k_fold_validation(X, y, k, lr, tol)
        results.append((k, mean_error, std_error))
        print(f"k={k}, Erro Médio: {mean_error}, Desvio Padrão: {std_error}")

    # Executa Leave-One-Out Cross-Validation
    loo_mean_error, loo_std_error = leave_one_out_validation(X, y, lr, tol)
    print(f"LOOCV, Erro Médio: {loo_mean_error}, Desvio Padrão: {loo_std_error}")

    # Plota os resultados
    kappa_vals = [result[0] for result in results]
    mean_errors = [result[1] for result in results]
    std_errors = [result[2] for result in results]

    plt.errorbar(kappa_vals, mean_errors, yerr=std_errors, fmt='o-', capsize=5)
    plt.title('Erro Médio e Desvio Padrão para Diferentes Valores de k')
    plt.xlabel('k (Número de Grupos)')
    plt.ylabel('Erro Médio (RMSE)')
    plt.xscale('log')
    plt.grid()
    plt.axhline(y=loo_mean_error, color='r', linestyle='--', label='LOOCV (Erro Médio)')
    plt.legend()
    plt.show()
```

Figura 3.9: Função com a chamada das validações

Após a implementação das funções de validação, foi realizada uma plotagem dos erros médios e dos desvios padrão correspondentes para todos os valores de κ testados. Essa visualização fornece uma forma clara de comparar o desempenho do modelo em diferentes configurações de validação. A análise gráfica dos resultados ajuda a identificar padrões, como a diminuição do erro médio com o aumento do número de folds, o que indica uma melhoria na capacidade do modelo de generalizar a partir dos dados de treino. Com isso, as informações obtidas a partir das plotagens são cruciais para a tomada de decisão sobre qual abordagem de validação adoptar em futuras análises.

Capítulo

4

Resultados

4.1 Introdução

Nesta seção, será apresentado o desempenho do modelo de regressão com base em diferentes configurações de iterações e tolerância. Foram realizados testes utilizando validação cruzada κ-fold e Leave-One-Out Cross Validation (LOO-CV), com a intenção de observar a convergência e a estabilidade do algoritmo, bem como o comportamento dos erros em relação ao número de iterações e à tolerância do algoritmo.

4.2 Análise de Desempenho

1. **Convergência e Comportamento do Algoritmo**

   Durante a execução do modelo, foi observado um comportamento não esperado de convergência, descrito visualmente como um padrão de "L", o que indica que o algoritmo não está a ajustar adequadamente os parâmetros ao longo das iterações. Esse padrão de "L"pode indicar uma falha no processo de ajuste do gradiente, sugerindo que o modelo está "travado"num patamar e, portanto, incapaz de encontrar o mínimo global da função de custo.

2. **Ajuste de Iterações e Tolerância**

   Para investigar a causa desse comportamento, foi realizada uma análise comparativa dos resultados com diferentes números de iterações e valores de tolerância. Abaixo estão as observações:

   - **Com 1000 iterações:**

- A performance do modelo, em termos de erro médio e desvio padrão, apresenta uma redução significativa conforme o valor de k aumenta. Isso sugere que, com poucos folds, o modelo é mais sensível a variações nos dados de treino.

- Erro Médio e Desvio Padrão: Os valores de erro médio variam de aproximadamente 11.325,75 (com $k = 3$) para 9.026,49 (com LOO-CV). O desvio padrão também aumenta substancialmente com o crescimento de k, chegando a 6.833,11 no LOO-CV.

k=3, Erro Médio: 11325.754327685381, Desvio Padrão: 124.2489333143832
k=5, Erro Médio: 11306.05897033882, Desvio Padrão: 411.23712994021923
k=10, Erro Médio: 11298.063491289851, Desvio Padrão: 656.0577761659667
k=15, Erro Médio: 11285.792565299611, Desvio Padrão: 921.3059635956294
k=50, Erro Médio: 11160.110817077135, Desvio Padrão: 1942.1372108817425
k=100, Erro Médio: 10965.1492173692, Desvio Padrão: 2849.9798931150626
k=200, Erro Médio: 10625.110346830385, Desvio Padrão: 3940.368965689519
k=400, Erro Médio: 10147.241684077377, Desvio Padrão: 5048.922974323697
k=800, Erro Médio: 9525.605260287006, Desvio Padrão: 6264.383927688719
k=1338, Erro Médio: 9026.488888828759, Desvio Padrão: 6833.116948941676
LOO-CV, Erro Médio: 9026.506817835252, Desvio Padrão: 6833.128171975122

- **Com 10.000 iterações:**

  - Um aumento no número de iterações trouxe melhorias no desempenho. O erro médio diminuiu em todos os cenários, o que é esperado, visto que mais iterações permitem que o modelo explore melhor o espaço de parâmetros.

  - Comportamento: Mesmo com 10.000 iterações, o erro médio ainda se mantém elevado em relação ao esperado para um modelo com bom ajuste (variando entre 10.297,33 para k=3 e 9.632,16 para k=200).

  - Desvio Padrão: O desvio padrão também diminuiu com mais iterações, sugerindo maior estabilidade no aprendizado do modelo. Contudo, o desvio padrão permanece considerável, indicando que o modelo ainda tem dificuldades para generalizar.

  k=3, Erro Médio: 10297.330870863456, Desvio Padrão: 106.30424025758644
  k=5, Erro Médio: 10271.96031859773, Desvio Padrão: 380.05502385749764
  k=10, Erro Médio: 10268.50757686343, Desvio Padrão: 618.4007050216621
  k=15, Erro Médio: 10253.871157284004, Desvio Padrão: 880.0956835240198
  k=50, Erro Médio: 10135.659292755743, Desvio Padrão: 1799.153137721298
  k=100, Erro Médio: 9953.237315645289, Desvio Padrão: 2634.1985430641835
  k=200, Erro Médio: 9632.157230693176, Desvio Padrão: 3642.2458678038847

- **Com Tolerância aumentada para 1:**

- Ao aumentar a tolerância, o comportamento de convergência foi acelerado, porém a precisão do modelo diminuiu. O erro médio aumentou ligeiramente, com valores variando de 11.444,73 (k=3) para 9.105,71 (LOO-CV).
- Análise: Esse aumento na tolerância permitiu que o modelo convergisse mais rapidamente, mas comprometeu a capacidade de ajuste fino dos parâmetros. O erro e o desvio padrão mostram que o modelo está parando de ajustar cedo demais, antes de atingir o melhor ajuste possível.

3. **Discussão sobre o Desempenho**

   A análise dos resultados sugere que o modelo está tendo dificuldades em convergir adequadamente. Isso pode estar relacionado a diversos fatores:

   - **Taxa de aprendizagem**: Um fator importante que pode estar causando esse comportamento de "L"é a taxa de aprendizagem utilizada no algoritmo. Se a taxa for muito alta, o modelo pode estar oscilando e não convergindo. Uma possível solução seria reduzir a taxa de aprendizagem.
   - **Número de Iterações**: Mesmo com 10.000 iterações, o modelo ainda não está convergindo para uma solução satisfatória. Embora o aumento nas iterações tenha reduzido o erro médio, ele não está resolvendo o problema completamente. Isso pode indicar que há outros problemas, como um mau dimensionamento dos dados ou problemas na inicialização dos pesos.
   - **Tolerância**: Aumentar a tolerância, embora acelere a convergência, pode estar levando a uma solução subótima, onde o modelo para o ajuste antes de encontrar o verdadeiro mínimo. Esse comportamento explica o aumento do erro médio ao usar uma tolerância de 1, o que sugere que essa configuração não é ideal.
   - **Generalização**: Os resultados do desvio padrão indicam que o modelo tem dificuldade em generalizar adequadamente, mesmo com diferentes valores de k. Isso aponta para a necessidade de ajustes no processo de validação ou no próprio modelo, como introduzir regularização, normalizar as variáveis, ou ajustar a função de custo. Foram feitas tentativas nesse sentindo, mas não obteve-se bons resultados.

4. **Testes de Validação (κ-fold e LOO-CV)** Ao realizar validação cruzada com diferentes valores de κ e utilizando LOO-CV, foi possível observar

que o erro médio diminui conforme κ aumenta. O método LOO-CV, embora computacionalmente mais caro, proporcionou o menor erro médio, com aproximadamente 9.026,49 para 1.000 iterações e 9.105,54 para uma tolerância de 1.

Esses resultados indicam que o modelo tende a se beneficiar de mais dados de treinamento ao realizar a validação, pois as versões de κ menores (como κ=3 ou κ=5) apresentaram erros maiores e desvios padrões menores. No entanto, o desvio padrão se torna consideravelmente elevado conforme o número de folds aumenta, sugerindo uma possível falta de robustez no modelo.

4.2.1 Considerações modelo inicial

A análise dos resultados revela que o algoritmo está enfrentando dificuldades em convergir de maneira eficiente e precisa. O aumento no número de iterações e na tolerância tem mostrado impactos mistos no desempenho do modelo. A fim de melhorar a convergência e reduzir o erro médio, recomenda-se investigar os seguintes aspectos:

- Reduzir a taxa de aprendizagem;
- Ajustar o número de iterações para permitir que o modelo explore melhor os parâmetros;
- Avaliar a necessidade de regularização ou normalização dos dados;
- Considerar um ajuste fino da função de custo.

A persistência do comportamento em "L"sugere que o algoritmo pode estar preso em um mínimo local, ou que está "travando"em uma área plana da função de custo. O ajuste dos hiperparâmetros mencionados acima pode ajudar a superar esse problema e melhorar o desempenho geral.

4.3 Analise de Desempenho - Ajuste polinomial

Os resultados demonstram a performance de diferentes valores de k no modelo polinomial, avaliado sob diferentes configurações de repetições e tolerância. Abaixo estão as principais tendências observadas:

- **Com 1000 iterações:**

Tendência geral: À medida que o valor de κ aumenta, o erro médio diminui, mas o desvio padrão aumenta. Para valores baixos de κ (como 3), temos um erro médio relativamente alto (11075.93), com um desvio padrão baixo (129.33), o que sugere consistência, mas um ajuste subótimo.

Conforme o valor de κ aumenta (até κ=100), o erro médio cai para 10748.36, indicando um melhor ajuste ao modelo, mas com um desvio padrão muito maior (2678.17), o que pode indicar variabilidade maior nos resultados. Conclusão: Um κ mais alto oferece um ajuste mais preciso, mas à custa de uma maior variabilidade nas previsões.

- **Com 10000 iterações:**

  Tendência geral: Com mais iterações, o erro médio é significativamente menor, o que indica que o aumento no número de repetições ajuda a melhorar a performance do modelo.

  Para κ=3, o erro médio é reduzido para 8758.43, com um desvio padrão de 101.26, o que mostra um bom compromisso entre precisão e consistência.

  À medida que κ aumenta, o erro médio continua a diminuir até κ=50, com um valor de 8600.45, mas o desvio padrão também cresce, atingindo 1483.05, o que mostra uma maior variação nas previsões.

  Conclusão: Mais repetições proporcionam uma melhora significativa na performance geral, com κ=50 oferecendo o melhor erro médio com um equilíbrio aceitável no desvio padrão.

- **Tolerância 1:**

  Tendência geral: Com tolerância mais alta, os erros médios também são mais altos, mas uma tendência semelhante se mantém: valores mais altos de k reduzem o erro médio, enquanto aumentam a variabilidade.

  Para κ=3, temos o erro médio mais alto de 11441.56, com desvio padrão de 163.21, demonstrando menor variabilidade.

  Conforme o valor de κ aumenta até κ=1338, o erro médio cai significativamente para 8816.17, mas o desvio padrão se eleva para 7276.25, indicando grande variabilidade.

  Aumentar κ melhora o erro médio, mas a variabilidade é muito maior, especialmente com tolerância mais alta. O κ=1338 tem o menor erro, mas o desvio padrão elevado implica uma grande instabilidade no modelo.

4.3.1 Considerações para o modelo polinomial

- **Erro Médio:** O aumento de κ geralmente reduz o erro médio, mas em compensação aumenta o desvio padrão, sugerindo que o modelo está sendo mais preciso em alguns casos, mas também pode ter maior variação.
- **Número de Repetições:**Aumentar as iterações para 10000 melhora significativamente o desempenho geral do modelo, com erros médios menores e maior estabilidade.
- **Tolerância:** Uma tolerância mais alta parece aumentar os erros médios, mas não impacta tanto na tendência geral da variação do erro conforme o valor de κ aumenta.

Na escolha ideal de κ, é importante balancear a precisão (erro médio) com a consistência (desvio padrão), especialmente dependendo do caso de uso. Valores intermediários de κ (como entre κ=10 e κ=50) parecem oferecer um bom compromisso em diferentes cenários.

4.4 Comparação dos modelos | Conclusão

Tanto o modelo normal quanto o modelo polinomial utilizando valores de κ diferentes, repetindo as simulações várias vezes, e variando o número de iterações e a tolerância. Aqui está uma comparação detalhada dos dois modelos, focando em três aspectos principais: erro médio, desvio padrão e impacto do valor de κ.

4.4.1 1000 Iterações

Modelo Normal (Resultados anteriores):

- k=3: Erro Médio: 11449.40, Desvio Padrão: 203.43
- k=5: Erro Médio: 11432.37, Desvio Padrão: 437.21
- k=10: Erro Médio: 11425.61, Desvio Padrão: 636.29
- k=15: Erro Médio: 11416.49, Desvio Padrão: 880.91
- k=50: Erro Médio: 11293.96, Desvio Padrão: 1955.96
- k=100: Erro Médio: 11093.86, Desvio Padrão: 2878.84

Modelo Polinomial:

- k=3: Erro Médio: 11075.93, Desvio Padrão: 129.33
- k=5: Erro Médio: 11057.56, Desvio Padrão: 372.66
- k=10: Erro Médio: 11051.93, Desvio Padrão: 589.55
- k=15: Erro Médio: 11044.46, Desvio Padrão: 798.32
- k=50: Erro Médio: 10928.88, Desvio Padrão: 1797.42
- k=100: Erro Médio: 10748.36, Desvio Padrão: 2678.17

Análise Comparativa:

- **Erro Médio:**

  O modelo polinomial mostra consistentemente erros médios mais baixos em comparação com o modelo normal. Isso sugere que o modelo polinomial está conseguindo capturar melhor as relações não-lineares, o que o torna mais eficaz.

- **Desvio Padrão:**

  No modelo polinomial, o desvio padrão também é mais baixo, especialmente para valores menores de κ (como κ=3 e κ=5), indicando previsões mais consistentes em relação ao modelo normal.

- **Conclusão:**

  Para 1000 repetições, o modelo polinomial oferece tanto uma maior precisão quanto maior consistência em relação ao modelo normal.

4.4.2 10000 Iterações

Modelo Normal (Resultados anteriores):

- k=3: Erro Médio: 8824.45, Desvio Padrão: 159.91
- k=5: Erro Médio: 8796.69, Desvio Padrão: 329.74
- k=10: Erro Médio: 8786.68, Desvio Padrão: 518.21
- k=15: Erro Médio: 8781.31, Desvio Padrão: 711.83
- k=50: Erro Médio: 8676.64, Desvio Padrão: 1538.52

Modelo Polinomial:

- k=3: Erro Médio: 8758.43, Desvio Padrão: 101.26
- k=5: Erro Médio: 8706.79, Desvio Padrão: 312.90
- k=10: Erro Médio: 8712.45, Desvio Padrão: 522.65
- k=15: Erro Médio: 8698.97, Desvio Padrão: 733.54
- k=50: Erro Médio: 8600.45, Desvio Padrão: 1483.05

Análise Comparativa:

- **Erro Médio:**

  Para 10000 repetições, o modelo polinomial ainda se sai melhor em termos de erro médio. Os valores de erro médio são ligeiramente menores em quase todos os valores de κ, com destaque para κ=5, onde o erro médio é 8706.79 no polinomial, em comparação com 8796.69 no normal.

- **Desvio Padrão:**

  O modelo polinomial apresenta desvios padrões menores para valores mais baixos de κ (como κ=3), mas essa vantagem diminui à medida que κ aumenta.

- **Conclusão:**

  Com 10000 iterações, o modelo polinomial ainda mostra uma vantagem no erro médio e, até certo ponto, no desvio padrão, indicando que ele continua a ser mais preciso.

4.4.3 Com Tolerância 1

Modelo Normal (Resultados anteriores):

- Modelo Normal (Resultados anteriores):
- k=3: Erro Médio: 11451.51, Desvio Padrão: 160.21
- k=5: Erro Médio: 11427.93, Desvio Padrão: 398.36
- k=10: Erro Médio: 11418.27, Desvio Padrão: 603.77
- k=15: Erro Médio: 11412.32, Desvio Padrão: 838.55

- k=50: Erro Médio: 11283.83, Desvio Padrão: 1946.48
- k=100: Erro Médio: 11086.92, Desvio Padrão: 2867.58
- k=200: Erro Médio: 10773.38, Desvio Padrão: 3975.72
- k=400: Erro Médio: 10305.61, Desvio Padrão: 5075.86
- k=800: Erro Médio: 9608.21, Desvio Padrão: 6558.28
- k=1338: Erro Médio: 8799.55, Desvio Padrão: 7205.78

Modelo Polinomial:

- k=3: Erro Médio: 11441.56, Desvio Padrão: 163.21
- k=5: Erro Médio: 11416.29, Desvio Padrão: 398.09
- k=10: Erro Médio: 11410.42, Desvio Padrão: 606.44
- k=15: Erro Médio: 11402.78, Desvio Padrão: 855.50
- k=50: Erro Médio: 11277.09, Desvio Padrão: 1923.97
- k=100: Erro Médio: 11083.43, Desvio Padrão: 2828.81
- k=200: Erro Médio: 10746.64, Desvio Padrão: 3936.48
- k=400: Erro Médio: 10241.70, Desvio Padrão: 5117.82
- k=800: Erro Médio: 9455.35, Desvio Padrão: 6571.90
- k=1338: Erro Médio: 8816.17, Desvio Padrão: 7276.25

Análise Comparativa:

- **Erro Médio:**

  Para valores de κ menores (até κ=100), o modelo polinomial e o modelo normal apresentam resultados quase idênticos. Porém, a partir de κ=200, o modelo polinomial começa a se destacar, com erros médios consistentemente mais baixos.

- **Desvio Padrão:**

  Para os valores mais altos de κ, o desvio padrão no modelo polinomial é ligeiramente mais baixo, embora ainda haja grande variabilidade nos resultados para valores altos de κ.

- **Conclusão:**

  Com uma tolerância de 1, o modelo polinomial se comporta de maneira similar ao normal nos valores baixos de κ, mas oferece uma melhoria significativa nos valores mais altos de κ, tanto em termos de erro médio quanto de desvio padrão

4.5 Conclusão Geral:

- Precisão: O modelo polinomial é superior ao modelo normal em termos de erro médio em quase todos os cenários, sugerindo que ele captura melhor as complexidades do problema.
- Consistência: O modelo polinomial geralmente apresenta um desvio padrão mais baixo para valores menores de κ, indicando maior consistência nas previsões, embora a variabilidade aumente à medida que κ cresce.
- Repetições: Aumentar o número de iterações melhora significativamente ambos os modelos, mas o modelo polinomial mantém uma vantagem em termos de erro médio.

Em resumo, o modelo polinomial oferece maior precisão e, em muitos casos

Bibliografia

C. Bishop. **Pattern Recognition and Machine Learning**, Springer, ISBN-13: 978-0387310732, 2011.

M. Mohri, A. Rostamizadeh, A. Talwalkar, F. Bach. **Foundations of Machine Learning**, ISBN-13: 978-0262039406, 2018.