Universidade da Beira Interior
Departamento de Informática

Nº Trabalho Prático 1: *Regressão Linear*

Elaborado por:

João Miguel Baltazar Martins

Orientador:

Professor Doutor Hugo Pedro Proença

10 de outubro de 2023

UBI Universidade da Beira Interior

EQM Erro Quadrático Médio

BMI Body Mass Index

Conteúdo

Conteúdo **2**

1 Introdução **3**
1.1 Enquadramento 3
1.2 Organização do Documento 3

2 Desenvolvimento **5**
2.1 Introdução 5
2.2 Análise Exploratória de dados 5
2.3 Implementação do *script* que obtém o melhor modelo, segundo a descida do gradiente 11
2.4 Função de Medição de Desempenho e Validação K-Fold 15
2.5 Adaptação do *Script* para um Modelo Polinomial 16

3 Conclusões **21**

Capítulo 1

Introdução

1.1 Enquadramento

Elaborado no âmbito da disciplina de Aprendizagem Automática do 1º ano do mestrado de Engenharia Informática da Universidade da Beira Interior (UBI), este trabalho tem como objetivo analisar o conjunto de dados *"Medical Cost Personal Dataset"* disponível no *Kaggle.com* e investigar se é possível prever as despesas médicas cobradas pelo seguro de saúde com base em outras características dos pacientes. O conjunto de dados inclui informações sobre idade, sexo, índice de massa corporal (BMI), número de filhos, se o paciente é fumador, a região em que residem e as despesas médicas.

A previsão das despesas médicas é uma questão relevante, uma vez que pode ajudar as seguradoras a determinar prémios e políticas de seguro mais precisos e também pode fornecer informações úteis aos pacientes sobre os fatores que mais afetam os seus custos de saúde.

1.2 Organização do Documento

1. O primeiro capítulo – **Introdução** – apresenta o projeto, o enquadramento para o mesmo e a respetiva organização do documento.

2. O segundo capítulo – **Desenvolvimento** – é feita a análise dos dados, apresentado o *script* que nos dá o melhor modelo segundo o gradiente descendente, a função que permite medir o desempenho do modelo conforme a validação do esquema *k-fold*, o desempenho deste para diferentes k's e a adaptação do *script* anteriormente mencionado de forma a comportar um modelo polinomial aos nossos dados.

3. O terceiro capítulo – **Conclusão** – refere as conclusões principais a tirar deste trabalho, bem como uma reflexão sobre um trabalho futuro.

Capítulo 2

Desenvolvimento

2.1 Introdução

Neste capítulo, abordaremos várias etapas cruciais no processo de análise de dados e modelagem, cada uma contribuindo para o nosso objetivo de prever as despesas médicas faturadas pelo seguro de saúde. As secções a seguir descrevem essas etapas em detalhes:

1. Análise Exploratória de dados
2. Proposta de *Script* de Regressão Linear com Gradiente Descendente.
3. Função de Medição de Desempenho e Validação *K-Fold*.
4. Análise de Desempenho com Diferentes Valores de *K*.
5. Adaptação do *Script* para um Modelo Polinomial.

O nosso modelo de regressão linear simples foi construído com a seguinte equação:

$$y(\theta) = \theta_0 + \theta_1 \cdot \text{age} + \theta_2 \cdot \text{sex} + \theta_3 \cdot \text{bmi} + \theta_4 \cdot \text{children} + \theta_5 \cdot \text{smoker} + \theta_6 \cdot \text{region}$$

2.2 Análise Exploratória de dados

Converter todos os dados em valores numéricos

```
# Extract features and target variable
categorical_features = ['sex', 'smoker', 'region']
numerical_features = ['age', 'bmi', 'children']

# Convert categorical features into one-hot encoding
X_categorical = pd.get_dummies(data[categorical_features], drop_first=
    True)
X_numerical = data[numerical_features]
X = pd.concat([X_categorical, X_numerical], axis=1)
```

Excerto de Código 2.1: Conversão dos dados para valores númericos (*one-hot encoding*)

São definidas as características do conjunto de dados que são categóricas (como sexo, fumador e região) e as que são numéricas (como idade, índice de massa corporal e número de filhos).*region*. Converte as características categóricas numa representação numérica chamada *one-hot encoding* para que o algoritmo do gradiente descendente, posteriomente apresnetado, possa trabalhar com elas de forma eficaz. Isso cria variáveis binárias (0s e 1s) para cada categoria possível.

De notar que, em conjuntos de dados com muitas categorias, o *one-hot encoding* pode aumentar significativamente a dimensionalidade dos dados.

Verificar se Existem Valores Nulos

```
# Check for missing values
missing_values = data.isnull().sum()
print(missing_values)
```

Excerto de Código 2.2: Verificação de valores nulos

```
age         0
sex         0
bmi         0
children    0
smoker      0
region      0
charges     0
dtype: int64

Process finished with exit code 0
```

Excerto de Código 2.3: Output ao executar a verificação dos valores nulos

Ao observar o *output* verifica-se que não existem valores nulos, também conhecidos como valores ausentes, podendo assim garantir que o modelo que advém destes dados seja robusto e preciso.

Gráficos de Barras e Estimativas de Densidade

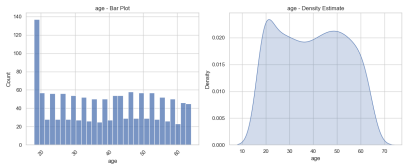

Figura 2.1: Gráfico da *feature* idade

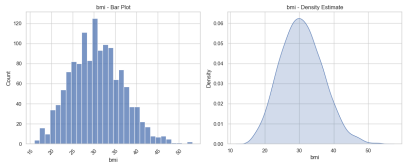

Figura 2.2: Gráfico da *feature BMI*

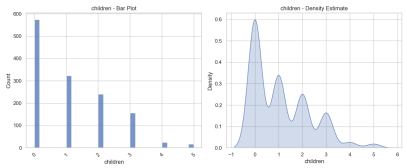

Figura 2.3: Gráfico da *feature* filhos

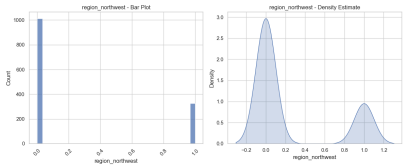

Figura 2.4: Gráfico da *feature* região *northwest*

- Gráfico da *feature* idade
  - Maioria das idades encontra-se na faixa até aos 20 anos.
- Gráfico da *feature BMI*

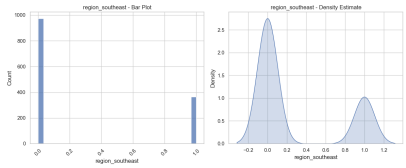

Figura 2.5: Gráfico da *feature* região *southeast*

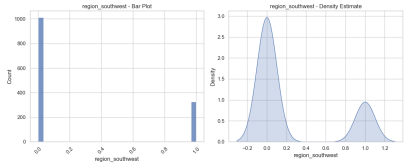

Figura 2.6: Gráfico da *feature* região *southwest*

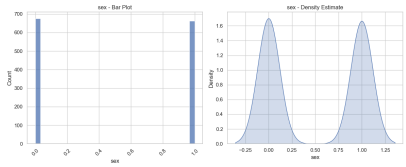

Figura 2.7: Gráfico da *feature* género

  - Há uma ampla variação nos valores de BMI.
  - Existem tanto casos de obesidade extrema como de baixo peso. Estes *outliers* podem ser pontos de interesse para análises mai aprofundadas.

- Gráfico da *feature* filhos
  - A maioria das pessoas não tem filhos e existe uma minoria com cinco filhos.

- Gráficos das regiões
  - As três regiões são retratadas nos dados de forma bastante equilibraria, ou seja, aparecem nos dados com a mesma predominância, entre 250 a 400 instâncias por região.

- Gráfico da *feature* género
  - O sexo feminino representado por um e o sexo masculino representado por zero. Observa-se que os dois géneros têm aproxima-

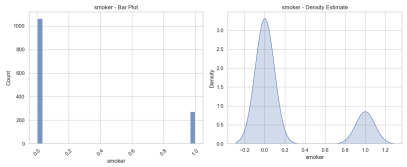

Figura 2.8: Gráfico da *feature* fumador

damente o mesmo número de instâncias no nosso conjunto de dados.

- Gráfico da *feature* fumador
  - O zero representa o não fumador e um o fumador. O gráfico de barras demonstra um desvio substancial na centralização das contagens, com a barra mais alta na contagem para os não fumadores, enquanto os fumadores têm uma contagem mais baixa.

Gráficos de Dispersão entre Cada Característica e a Variável Dependente

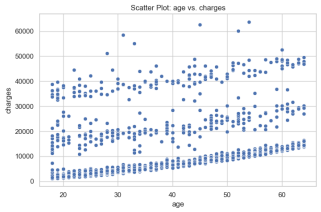

Figura 2.9: Gráfico da *feature* idade vs *charges*

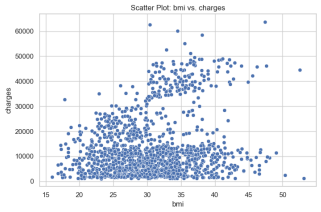

Figura 2.10: Gráfico da *feature BMI* vs *charges*

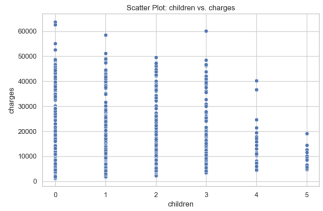

Figura 2.11: Gráfico da *feature* filhos vs *charges*

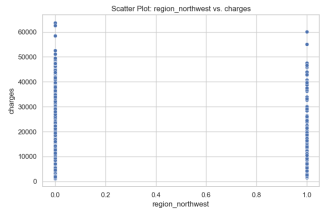

Figura 2.12: Gráfico da *feature* região *northwest* vs *charges*

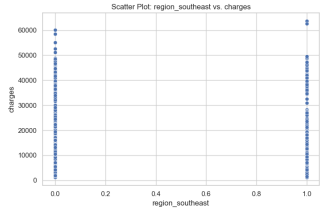

Figura 2.13: Gráfico da *feature* região *southeast* vs *charges*

1. **Idade (*age*):** Existe uma correlação positiva entre a idade e os *charges*. À medida que a idade aumenta, as cobranças tendem a aumentar. Isso sugere que a idade é uma variável importante a ser considerada no modelo.

2. **BMI** Existe relação positiva entre o BMI e os *charges*, embora não seja tão forte quanto a idade. À medida que o BMI aumenta, os *charges* tendem a aumentar. Deve-se incluir o BMI como uma característica no modelo.

3. **Número de Filhos:** A relação entre o número de filhos e os *charges* não é tão clara a partir do gráfico de dispersão. Pode ser útil, mas a sua influência parece menos óbvia em comparação com idade e o BMI.

4. **Fumador:** Há uma diferença notável nos grupos de fumadores e não fumadores em relação aos *charges*. Fumadores parecem ter *charges* substancialmente mais altas. Esta é uma característica muito importante a ser incluída no modelo.

5. **Género:** O gráfico de dispersão de sexo vs. *charges* não mostra uma

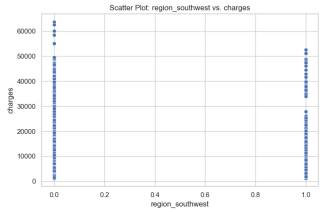

Figura 2.14: Gráfico da *feature* região *southwest* vs *charges*

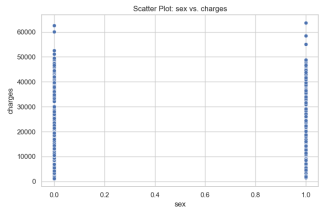

Figura 2.15: Gráfico da *feature* género vs *charges*

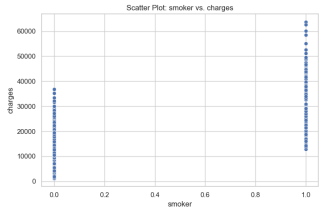

Figura 2.16: Gráfico da *feature* fumador vs *charges*

diferença clara entre homens e mulheres. Portanto, a correlação entre sexo e *charges* parece ser fraca o que resulta numa característica menos influente no modelo.

6. **Regiões:** A partir dos gráficos das regiões, é de notar que estas têm uma correlação limitada com os *charges*. A região pode ter alguma influência, mas não é tão forte quanto outras características.

2.3 Implementação do *script* que obtém o melhor modelo, segundo a descida do gradiente

Este código em Python implementa o algoritmo de descida de gradiente. A regressão linear é um método de aprendizagem supervisionado usado para

prever uma variável dependente (neste caso, 'charges') com base num conjunto de variáveis independentes (as *features* em 'X').

```
X = pd.concat([X_categorical, X_numerical], axis=1)
y = data['charges']


# Number of iterations and learning rate
num_iterations = 1000
learning_rate = 0.0001


# Define the hypothesis function for multiple features
def h(theta, x):
    return sum(theta[i] * x[i] for i in range(len(theta)))


# Define the cost function for multiple features
def cost_function(theta, X, y):
    m = len(X)
    error = 0
    for i in range(m):
        error += (h(theta, X.iloc[i]) - y.iloc[i]) ** 2
    return error / (2 * m)


# Gradient Descent
def gradient_descent(theta, X, y, lr):
    m = len(X)
    for iteration in range(num_iterations):
        errors = [h(theta, X.iloc[i]) - y.iloc[i] for i in range(m)]
        for j in range(len(theta)):
            gradient = sum(errors[i] * X.iloc[i][j] for i in range(m))
            theta[j] -= (lr * gradient) / m
        cost = cost_function(theta, X, y)
        print("Iteration", iteration + 1, "Cost:", cost)
    return theta

# Initialize theta with random values
def initialize_theta(num_features):
    return [random.uniform(-1, 1) for _ in range(num_features)]
```

Excerto de Código 2.4: *Script* Gradiente Descendente

Concatenação de Dados

As características do modelo são divididas em duas partes, `X_categorical` e `X_numerical`, que são posteriormente concatenadas ao longo do eixo 1 (co-

lunas) para formar a matriz de características 'X'.

Definimos os nossos dados de treino sabendo que:

- X é uma matriz de características independentes, onde cada linha representa uma observação e cada coluna representa uma característica.
- y é um vetor da variável dependente que queremos prever.

Definição da Função de Hipótese (`h(theta, x)`)

A função `h` é a função de hipótese que calcula a previsão com base nos parâmetros do modelo (`theta`) e nas características de entrada (`X`). É uma soma ponderada das características com os pesos (`theta`) correspondentes.

Definição da Função de Custo (`cost_function(theta, X, y)`)

A função `cost_function` calcula o Erro Quadrático Médio (EQM) entre as previsões da hipótese e a variável dependente 'y' para todo o conjunto de dados 'X'. O objetivo é minimizar o EQM durante o treino.

Descida de Gradiente (`gradient_descent(theta, X, y, lr)`)

Este é o núcleo do algoritmo. Ele realiza a otimização dos parâmetros do modelo (`theta`) usando o método de descida de gradiente.

- O *loop* externo executa um número fixo de iterações (1000 no seu caso).
- No *loop* interno, para cada iteração, calculamos os erros entre as previsões e a variável dependente 'y' (diferença entre a saída da hipótese e o valor real) para todos os exemplos no conjunto de treino.
- Em seguida, para cada peso (`theta[j]`), calculamos o gradiente, que é uma medida da direção e magnitude na qual devemos atualizar o peso para minimizar o erro.
- Atualizamos cada peso `theta[j]` subtraindo a taxa de aprendizagem (`lr`) multiplicada pelo gradiente médio para todas as amostras.
- Calculamos o custo atual e o exibimos.

Inicialização dos Parâmetros Theta (`initialize_theta(num_features)`)

Os parâmetros do modelo (`theta`) são inicializados com valores aleatórios no intervalo de -1 a 1. Isso é importante para começar o processo de otimização.

Análise dos Resultados obtidos do *Script*, *Learning Rate* e Número de Iterações

```
# Add a bias term (intercept) to the features
X.insert(0, 'bias', 1)

# Initialize theta with random values
initial_theta = initialize_theta(len(X.columns))

# Perform gradient descent
final_theta = gradient_descent(initial_theta, X, y, learning_rate)

print("Final theta:", final_theta)
```

Excerto de Código 2.5: Testar o *Script* do Gradiente Descendente

```
Iteration 1 Cost: 117329636.51879233
Iteration 2 Cost: 93596640.55881062
Iteration 3 Cost: 80697190.97735885

(...)

Iteration 998 Cost: 63838619.571890384
Iteration 999 Cost: 63837147.79146957
Iteration 1000 Cost: 63835676.06790444
Final theta: [-15.88899370061683, 26.91031380535677, 380.7342308318982,
    -18.86237407047587, 23.48945111375446, -27.230246153579746,
    208.2643143952807, 168.81477133512072, 60.979886895732136]
```

Excerto de Código 2.6: Output do *Script*

- Hiperparâmetro Taxa de Aprendizagem - Sabendo que, se a taxa de aprendizagem for muito grande, o algoritmo pode divergir, fazendo com que o custo aumente a cada iteração em vez de diminuir. Foi utilizada uma taxa de aprendizagem de 0.0001 pois vou observado que com valores de 0.01 e 0.001 o custo começa a aumentar em vez de diminuir.

- Hiperparâmetro Número de iterações - Sabendo que, se o número de iterações for muito pequeno, o algoritmo pode parar antes de atingir a

convergência, resultando em parâmetros do modelo sub-otimizados e um custo ainda alto e se o número de iterações for muito grande, o algoritmo pode continuar rodando sem melhorar muito mais o custo, o que é ineficiente, foram escolhidas 1000 iterações, pois foi observado que para mais iterações (10000) o custo não diminuía significativamente e para menos (100) o custo ainda diminui de forma relevante.

- Valores de custo - Os valores de custo estão a ser impressos a cada iteração. Como esperado, o custo diminui a cada iteração, o que indica que o algoritmo de gradiente descendente está a funcionar corretamente e otimizando os parâmetros *theta*

- Valores finais - O valor final dos parâmetros *theta* é exibido como [-15.88899370061683, 26.91031380535677, 380.7342308318982, -18.86237407047587, 23.48945111375446, -27.230246153579746, 208.2643143952807, 168.81477133512072, 60.979886895732136]. Estes valores representam os pesos atribuídos a cada característica durante o treino e são os parâmetros finais do modelo de regressão linear após o treino. Podem ser utilizados para fazer previsões em novos dados ou avaliar o desempenho do modelo num conjunto de teste separado.

2.4 Função de Medição de Desempenho e Validação K-Fold

```
# Function to perform k-fold cross-validation
def k_fold_cross_validation(X, y, k, learning_rate):
    n = len(X)
    fold_size = n // k
    mse_scores = []

    for i in range(k):
        start = i * fold_size
        end = (i + 1) * fold_size
        X_test = X[start:end]
        y_test = y[start:end]
        X_train = pd.concat([X[:start], X[end:]])
        y_train = pd.concat([y[:start], y[end:]])

        initial_theta = initialize_theta(len(X_train.columns))
        final_theta = gradient_descent(initial_theta, X_train, y_train,
            learning_rate)

        y_pred = [h(final_theta, x) for _, x in X_test.iterrows()]
```

```
        mse = sum((y_pred[i] - y_test.iloc[i]) ** 2 for i in range(len(
            y_test))) / len(y_test)
        mse_scores.append(mse)

    return mse_scores
```

Excerto de Código 2.7: Função de Medição de Desempenho e Validação K-Fold

A função realiza `k-fold cross-validation` para avaliar o desempenho de um modelo de regressão linear com base num conjunto de dados X e um vetor de alvos y. Ela divide o conjunto de dados em k partições (ou *folds*), treina o modelo em k-1 partições e avalia o modelo na partição de teste. A função retorna uma lista de EQM para cada *fold*, que podem ser usados para avaliar o desempenho médio do modelo.

Análise do Desempenho para Diferentes K-Folds

```
Average Mean Squared Error (MSE) over 3 folds: 128281408.7916392


Average Mean Squared Error (MSE) over 5 folds: 128137379.57754359


Average Mean Squared Error (MSE) over 7 folds: 128204367.99045934
```

Excerto de Código 2.8: Ouput para diferentes K-Folds

Os valores de EQM em torno de 128 milhões sugerem que o modelo tem algum erro na previsão das despesas médicas. A diferença entre os resultados de três, cinco e sete *folds* é pequena, o que sugere que o desempenho do modelo não é significativamente afetado pela escolha do número de *folds* na validação cruzada.

2.5 Adaptação do *Script* para um Modelo Polinomial

```
# Function to add polynomial features
def add_polynomial_features(X, p):
    X_poly = X.copy()
    for feature in X.columns:
        if feature != 'bias':
            for degree in range(2, p + 1):
                X_poly[f"{feature}^{degree}"] = X[feature] ** degree
    return X_poly
```

```
# Set the polynomial order
p = 2  # You can change the polynomial order as needed

# Add polynomial features to the data
X_poly = add_polynomial_features(X, p)

'''
TEST GRADIENT DESCENT
# Initialize theta with random values
initial_theta = initialize_theta(len(X_poly.columns))

# Perform gradient descent
final_theta = gradient_descent(initial_theta, X_poly, y, learning_rate)

print("Final theta:", final_theta)
'''


# Function to perform k-fold cross-validation
def k_fold_cross_validation(X, y, k, learning_rate):
    n = len(X)
    fold_size = n // k
    mse_scores = []

    for i in range(k):
        start = i * fold_size
        end = (i + 1) * fold_size
        X_test = X.iloc[start:end]
        y_test = y.iloc[start:end]
        X_train = pd.concat([X.iloc[:start], X.iloc[end:]])
        y_train = pd.concat([y.iloc[:start], y.iloc[end:]])

        initial_theta = initialize_theta(len(X_train.columns))
        final_theta = gradient_descent(initial_theta, X_train, y_train,
            learning_rate)

        y_pred = [h(final_theta, x) for _, x in X_test.iterrows()]
        mse = sum((y_pred[i] - y_test.iloc[i]) ** 2 for i in range(len(
            y_test))) / len(y_test)
        mse_scores.append(mse)

    return mse_scores


# Perform k-fold cross-validation
k = 5  # You can change the number of folds as needed
mse_scores = k_fold_cross_validation(X_poly, y, k, learning_rate)
```

```
# Calculate and print the average MSE
average_mse = np.mean(mse_scores)
print("Average Mean Squared Error (MSE) over", k, "folds:", average_mse)
```

Excerto de Código 2.9: Adaptação do *Script* para um Modelo Polinomial

Propósito: A função tem como objetivo criar *features* polinomiais a partir de um conjunto de *features* originais. Isto é útil para modelos de regressão polinomial, onde se deseja capturar relações não lineares entre as *features* e a variável de destino.

Entradas:

- X: Um conjunto de dados contendo as *features* originais representadas como um *DataFrame* ou matriz.
- p: A ordem do polinómio que determina quantas novas *features* polinomiais serão criadas para cada *feature* original.

Saída: X_{poly}: Um novo conjunto de dados contendo tanto as *features* originais quanto as novas *features* polinomiais criadas. As novas *features* são geradas elevando as *features* originais às potências de 2 até p.

Funcionamento Interno:

- Para cada *feature* no conjunto original (X), excepto a *feature* de viés (normalmente chamada de "bias") ou feature da variável dependente (*'charges'*:
  - Um *loop* interno gera novas *feature* elevando a *feature* original a diferentes potências (de 2 até p).
  - Cada nova feature é nomeada no formato "nome_da_feature$^{\text{grau}}$".
  - As novas features são adicionadas ao conjunto de dados X_{poly}.
- A função retorna o conjunto de dados X_{poly} com as novas *features* polinomiais.

Uso Típico:

- A função é usada quando se suspeita que as relações entre as *feature* originais e a variável de destino não são estritamente lineares, ou seja, quando as relações são mais complexas e curvilíneas.
- A criação de *feature* polinomiais permite que um modelo de regressão polinomial seja treinado para capturar essas relações mais complexas.

Importância:

- A função é importante em tarefas de regressão onde a relação entre as *feature* e o alvo é não linear. Ela aumenta a flexibilidade do modelo, mas também pode aumentar a sua complexidade, exigindo cuidados com *overfitting*.

Ajuste de p:

- A escolha de p (a ordem do polinómio) é um hiperparâmetro crítico. Valores maiores de p podem levar a um modelo supercomplexo e sobreajustado, enquanto valores menores podem não capturar relações importantes. O ajuste de p deve ser feito com validação cruzada ou outras técnicas de seleção de hiperparâmetro.

Exemplo de Uso:

- Se p for definido como 2, a função criará novas *features* quadráticas (elevadas ao quadrado) para cada *feature* original, permitindo ao modelo de regressão polinomial capturar relações quadráticas entre as *features* e a variável de destino.

Capítulo 3

Conclusões

Conclusões Principais

As principais conclusões a serem tiradas deste trabalho são:

- Fatores Significativos - O trabalho identificou que fatores como idade, tabagismo e IMC (Índice de Massa Corporal) têm um impacto significativo nos custos médicos. Em particular, ser fumante está fortemente associado a custos mais elevados, seguido por idade e IMC.
- Correlações Claras - A análise de correlação destacou correlações positivas entre idade, IMC e tabagismo com os custos médicos. Isso sugere que à medida que esses fatores aumentam, os custos médicos também tendem a aumentar. Por outro lado, o número de filhos tem uma correlação menos pronunciada com os custos.
- Validação Cruzada K-fold - A utilização da validação cruzada k-fold ajudou a avaliar a capacidade de generalização do modelo em diferentes conjuntos de dados de treino e teste. Isso é crucial para garantir que o modelo seja robusto e confiável.
- *Polynomial Regression*: A adaptação do modelo para incluir termos polinomiais de ordem superior permitiu explorar não-linearidades nos dados. Isso pode ser útil para ajustar o modelo a padrões complexos nos dados, mas também pode aumentar o risco de *overfitting*.

Limitações e Próximos Passos

O trabalho reconheceu as limitações, como a necessidade de mais dados e a possibilidade de outras características importantes que não estão incluídas no conjunto de dados. Portanto, futuros passos podem envolver a colheita de

mais informações ou a consideração de técnicas mais avançadas de seleção de recursos.

Aplicações Práticas: As conclusões podem ser aplicadas em cenários do mundo real, como na precificação de seguros médicos. Os resultados do modelo podem ser usados para tomar decisões informadas sobre como ajustar as políticas de seguro com base nos perfis dos segurados.