UNIVERSIDADE FEDERAL DO CEARÁ

# TECHNICAL REPORT

Aluno: Alana Martinho dos Santos e João Santiago Souza Lúcio

## 1. Introdução

**Parte A - Classificação:**

O dataset “voice” tem como objetivo classificar vozes como masculinas ou femininas com base em características acústicas.

**Parte B - Regressão**

O dataset “insurance” é um conjunto de dados que é utilizado para a previsão dos custos de seguro médico individual com base em características da vida dos segurados.

## 2. Observações

**Parte A - Classificação:**

Os principais problemas foram a demora para alguns códigos serem processados, além de resultados diferentes dos mostrados nos gráficos.

**Parte B - Regressão**

Houve alguns problemas com a importação do dataset, por isso foi usado o caminho obtido ao clicar no arquivo, adicionou-se o “r”(raw string) antes do caminho, pois mesmo com o caminho colocado como está, ainda estava ocorrendo erro.

## 3. Resultados e discussão

**Classificação:**

### 3.1. Classificação:

O código inicia importando a biblioteca *pandas* e o dataset (voice.csv) é carregado para um DataFrame: df = pd.read_csv("./dataset/voice.csv").

Em seguida com o comando “*print(df.isnull().sum*” é verificado se há células vazias em cada coluna do dataframe. A saída mostra que não há valores ausentes nas colunas, assim não necessitando de tratamento. Ainda foi usado o comando “df =

df.dropna()", que removeria linhas com valores faltando, porém nesse caso o dataset não foi alterado, pois já estava com os valores completos.

```
Valores ausentes antes do tratamento:
+"meanfreq"    0
sd             0
median         0
Q25            0
Q75            0
IQR            0
skew           0
kurt           0
sp.ent         0
sfm            0
mode           0
centroid       0
meanfun        0
minfun         0
maxfun         0
meandom        0
mindom         0
maxdom         0
dfrange        0
modindx        0
label          0
dtype: int64
```

Com o comando "print(df['label'].value_counts())" foi analisada a distribuição da variável alvo, definida como a coluna "label" que classifica as vozes em feminina e masculina. A saída mostra que o dataset contém 1584 amostras para a classe male e 1584 para a classe female, o que demonstra um dataset bem equilibrado.

Em seguida o foi verificado se havia colunas categóricas no dataset, mostrando a coluna "label", como object. O LabelEncoder da biblioteca scikit-learn foi utilizado para converter essas categorias em números.

Na etapa seguinte, A função df.describe() gera um resumo estatístico para todas as colunas numéricas. Count: Confirma que há 3.168 amostras em todas as colunas. Mean: Mostra a média de cada característica. A média da coluna label é 0.5, o que confirma novamente o balanceamento perfeito entre as classes (0 e 1). Std: Apresenta o desvio padrão, indicando a dispersão dos dados em torno da média. Min, 25%, 50% (mediana), 75%, max: fornecem uma visão sobre a distribuição dos valores de cada característica. Por exemplo, na coluna label, o mínimo é 0 ('female') e o máximo é 1 ('male').

| | +"meanfreq" | sd | ... | modindx | label |
|---|---|---|---|---|---|
| count | 3168.000000 | 3168.000000 | ... | 3168.000000 | 3168.000000 |
| mean | 0.180907 | 0.057126 | ... | 0.173752 | 0.500000 |
| std | 0.029918 | 0.016652 | ... | 0.119454 | 0.500079 |
| min | 0.039363 | 0.018363 | ... | 0.000000 | 0.000000 |
| 25% | 0.163662 | 0.041954 | ... | 0.099766 | 0.000000 |
| 50% | 0.184838 | 0.059155 | ... | 0.139357 | 0.500000 |
| 75% | 0.199146 | 0.067020 | ... | 0.209183 | 1.000000 |
| max | 0.251124 | 0.115273 | ... | 0.932374 | 1.000000 |

Por fim, com o comando “df.to_csv("./dataset/classificacao_ajustado.csv", index=False)”, o DataFrame processado é salvo como novo arquivo chamado “classificacao_ajustado.csv”. O parâmetro index=False evita que o índice do DataFrame seja salvo como uma nova coluna no arquivo.

### 3.2. KNN manual com vária distâncias

No código, os dados são carregados, dessa vez usando o dataset ajustado. O dataset é dividido em treino e teste:

```
    X_train, X_test, y_train, y_test = train_test_split(X,
y, test_size=0.2, random_state=42)
```

São definidas funções para o cálculos das distâncias:

```
    #distancia euclidiana
    def euclidean_distance(x1, x2):
        return np.sqrt(np.sum((x1 - x2)**2))
    #distancia manhattan
    def manhattan_distance(x1, x2):
        return np.sum(np.abs(x1 - x2))
```

```
#distancia chebyshev
def chebyshev_distance(x1, x2):
    return np.max(np.abs(x1 - x2))
#distancia mahalanobis
def mahalanobis_distance(x1, x2, cov_inv):
    diff = x1 - x2
    return np.sqrt(diff.T @ cov_inv @ diff)
```

O Knn é calculado através das funções “predict_classification”, “knn” e “calculate_accuracy”.

predict_classification: prevê a classe de um único ponto de teste (test_row). Ela calcula a distância deste ponto para todos os pontos do conjunto de treino usando a distance_func fornecida. As distâncias são armazenadas junto com os rótulos correspondentes. A lista de distâncias é ordenada da menor para a maior. Os rótulos dos k vizinhos mais próximos são selecionados. A função retorna a classe que aparece com mais frequência entre os vizinhos (a "votação").

knn: gerencia o processo de classificação para todo o conjunto de teste (X_test). Ela itera sobre cada ponto no conjunto de teste e chama predict_classification para obter uma previsão para cada um. Caso a distância seja Mahalanobis, ela primeiro calcula a inversa da matriz de covariância dos dados de treino, que será usada em todas as predições. Ao final, retorna um array com todas as previsões.

calculate_accuracy: mede o desempenho, calculando a proporção de previsões corretas.

Acurácia = previsões corretas/ total de previsões

Na última parte do código, o K é fixado como 6 e o knn é calculado para cada distância. A acurácia é calculada e armazenada no dicionário “results”.

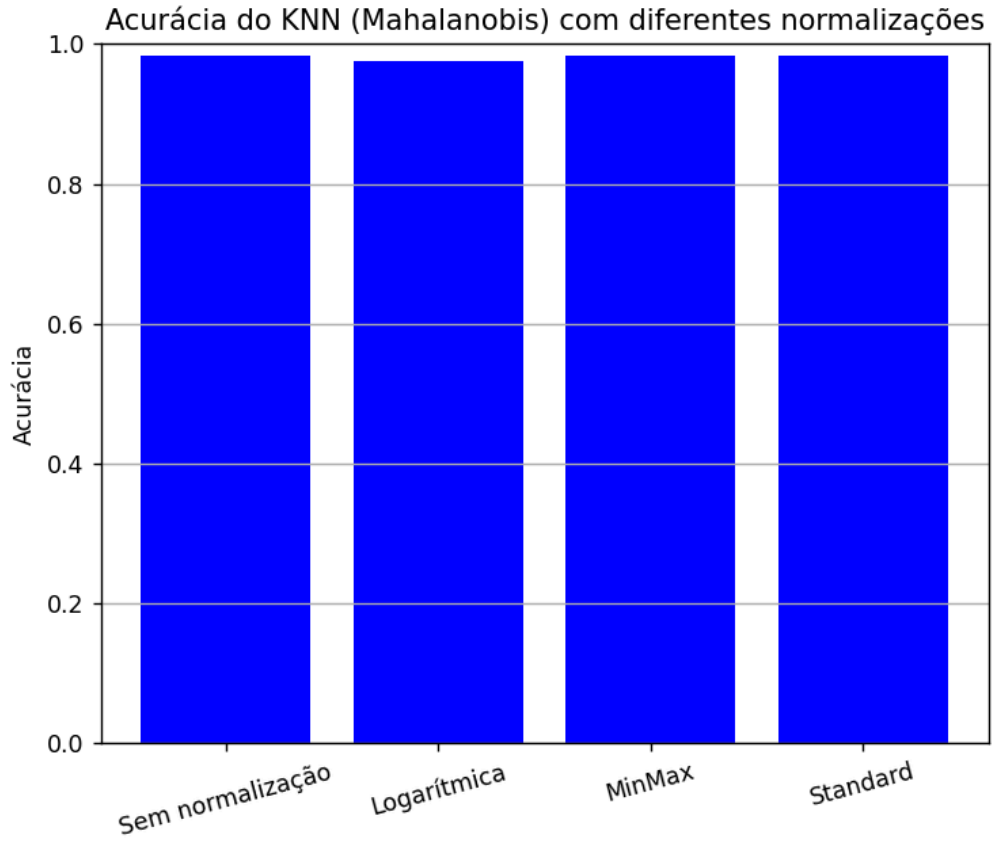
```
Comparação dos Resultados (Acurácia)
Valor de K (vizinhos): 6

Distância Euclidiana: 0.7224 (72.24%)
Distância Manhattan: 0.7965 (79.65%)
Distância Chebyshev: 0.7003 (70.03%)
Distância Mahalanobis: 0.9779 (97.79%)

--- Conclusão ---
A melhor métrica de distância para este problema (com K=6) foi a de Mahalanobis,
com uma acurácia de 97.79%.
```

### 3.3. Normalização, alteração do valor de K e efeito no KNN

Este código demandou um esforço maior, sobretudo pela velocidade do processamento.

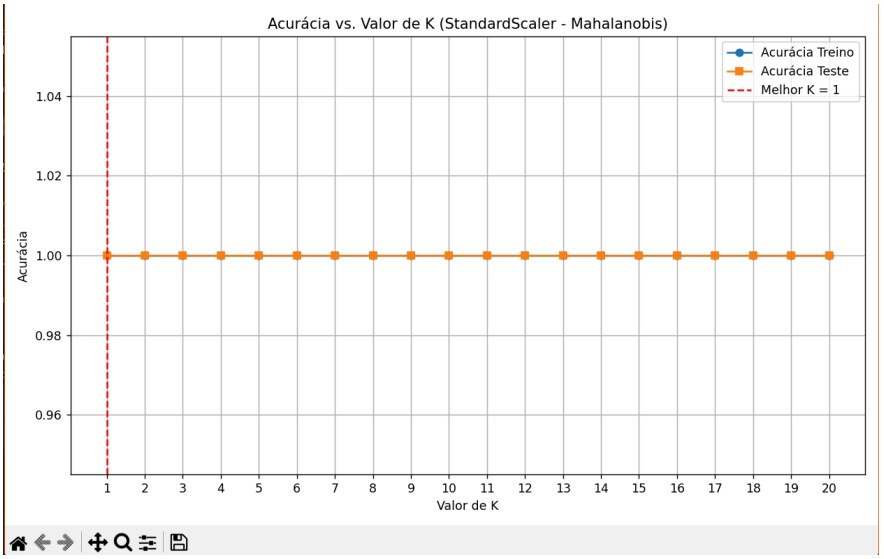
O início do código carrega o dataset classificacao_ajustado.csv e reutiliza as funções do KNN manual desenvolvidas anteriormente (mahalanobis_distance, predict_classification, knn, calculate_accuracy). É utilizada a metrica de distância Mahalanobis, definida anteriormente como a melhor distância.

Foi criada uma função para ajudar a avaliar o knn, automatizando o processo de avaliação com as normalizações, ele encapsula o processo de normalizar os dados (se necessário), dividir em treino/teste, rodar o KNN manual com distância de Mahalanobis, calcular e exibir a acurácia obtida.

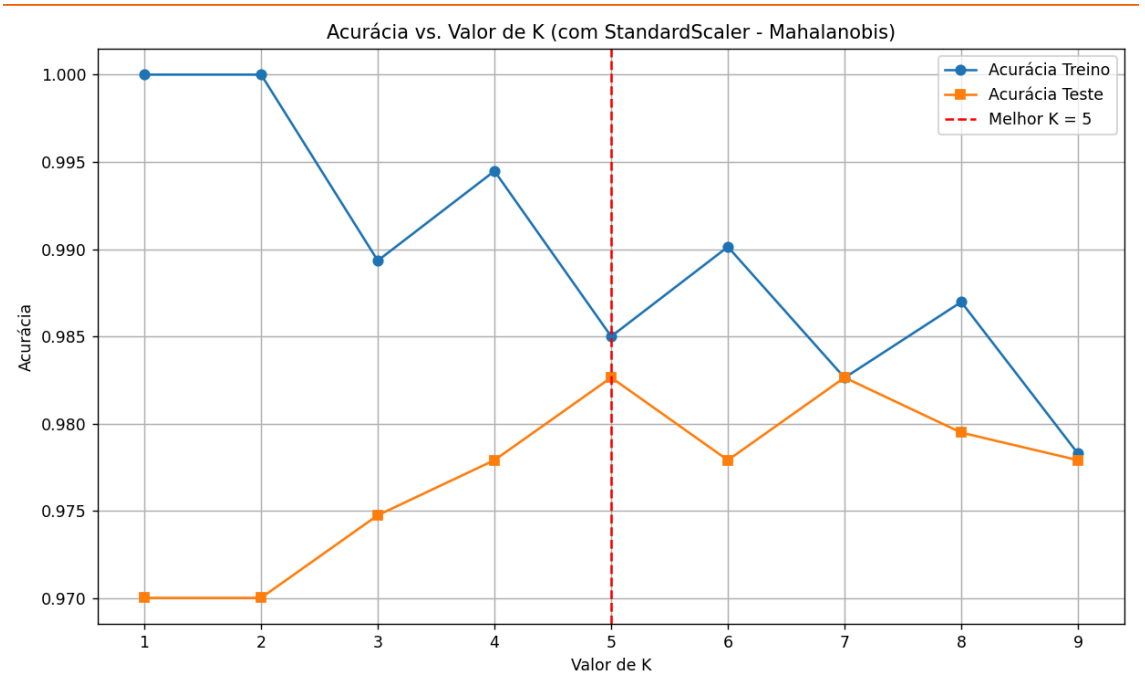



```
--- Avaliação de Acurácia com diferentes normalizações ---
Acurácia com sem normalização: 0.9826
Acurácia com logarítmica: 0.9763
Acurácia com MinMaxScaler: 0.9826
Acurácia com StandardScaler: 0.9826
```

A segunda parte do teste, deveria descobrir o melhor K para o KNN. Essa parte gerou problemas, primeiro pela demora no processamento, foi testado “k” com valores entre 1 e 21. Pela lentidão, foi decidido reduzir a amostra de dados, porém com a amostragem menor os resultados foram tendenciosos.




O teste então foi feito com o dataset completo e variação de valores de "k" entre 1 e 10. A saída entregou o melhor resultado com k= 5, porém no gráfico gerado percebe-se um melhor resultado quando k é igual a 7.




### 3.4. KNN com Sklearn e GridSearchCV

No código são utilizados o Pipeline e o GridSearch.

Pipeline: É uma ferramenta do scikit-learn que encadeia múltiplos passos de processamento em um único objeto. No código, ele primeiro aplica um normalizador (scaler) e depois treina o KNN. Ele garante que o normalizador seja ajustado (treinado) apenas com os dados de treino dentro de cada etapa da validação cruzada.

GridSearchCV: É uma técnica de otimização que realiza uma busca exaustiva por todas as combinações de parâmetros especificadas.

param_grid: Define os hiperparâmetros a serem testados. Aqui, ele testa todos os valores de K (n_neighbors) de 1 a 20.

```
param_grid = {
    'knn__n_neighbors': list(range(1, 21))
}
```

cv=5: Indica o uso de validação cruzada de 5 folds (5-fold cross-validation). O GridSearchCV divide os dados de treino em 5 partes, treina o modelo 5 vezes usando 4 partes para treino e 1 para validação, e calcula a média de desempenho.

O código itera sobre as normalizações (Log, MinMax, Standard), usando o GridSearchCV para encontrar o melhor K para cada uma delas e dá como saída:

Melhor configuração encontrada:

Normalização: MinMax

Melhor K: 6

Acurácia na Validação (CV): 0.9767

Acurácia no Teste: 0.9826

### 3.5. Cross-Validation e Avaliação Final

Para a cross-validation, foi aplicada a normalização MinMax, definida anteriormente como a que apresentou melhores resultados.

O modelo KNeighborsClassifier foi configurado com n_neighbors=6, o melhor valor de K encontrado no procedimento passado.

Foi utilizada validação cruzada com 5 folds (cv=5) para avaliar o desempenho geral do modelo:

Cross_val_score: calcula a acurácia média e desvio padrão.

Cross_val_predict: foi usado para gerar predições para toda a base, simulando como o modelo se comportaria em cada fold.

São exibidos média, desvio padrão, matriz de confusão e `classification_report`:

Acurácia média (Validação - CV): 0.9489

Desvio padrão : 0.0209

Isso significa que, em média, o modelo acerta 954,89% das classificações nos dados de validação. O desvio padrão de 2,09% indica que há baixa variabilidade entre os folds, o que sugere um modelo estável.

```
Matriz de Confusão:
[[1541   43]
 [ 119 1465]]
```

Verdadeiros negativos (homens corretamente classificados): 1541

Falsos positivos (homens classificados como mulheres): 43

Falsos negativos (mulheres classificadas como homens): 119

Verdadeiros positivos (mulheres corretamente classificadas): 1465

O modelo erra mais em classificar mulheres como homens (FN = 119) do que o oposto (FP = 43), mas ambos os tipos de erro são relativamente baixos.

```
Relatório de Classificação:
              precision    recall  f1-score   support

        male       0.93      0.97      0.95      1584
      female       0.97      0.92      0.95      1584

    accuracy                           0.95      3168
   macro avg       0.95      0.95      0.95      3168
weighted avg       0.95      0.95      0.95      3168
```

Precisão maior para mulheres (0.97): quando o modelo prediz "female", ele acerta em 97% dos casos.

Revocação maior para homens (0.97): ele consegue identificar corretamente 97% dos homens.

F1-score equilibrado: 0.95 para ambas as classes, indicando bom equilíbrio entre precisão e recall.

**Regressão:**

### 3.6 Pré-processamento e Correlação

Script responsável por preparar o “insurance.csv” para a modelagem de regressão.

Aqui é importado o arquivo insurance, com ressalva do erro citado acima.

```
# Carregar o dataset
df = pd.read_csv(r"C:\Users\jsslu\OneDrive\Área de Trabalho\UFC\Inteligencia Artificial\git\IA-CD\AV1\datasets\insurance.csv")
```

É calculado o número dos valores ausentes em cada coluna, e caso haja valores ausentes, removendo os mesmo

```
# Tratar valores ausentes (verificação)
if df.isnull().sum().any():
    df = df.dropna()
```

Aqui é convertido variáveis categóricas em variáveis dummy(binárias)

```
# Codificar variáveis categóricas
df_codificado = pd.get_dummies(df, drop_first=True)
```

Primeiro é calculada a matriz de correlação entre todas as colunas, depois é selecionada a coluna alvo, classificando as correlações em ordem decrescente, mostrando quais as colunas têm maior correlação com a coluna alvo.

```
# Verificar correlação com a variável-alvo
correlacao = df_codificado.corr()['charges'].sort_values(ascending=False)
print("\nCorrelação com 'charges':\n", correlacao)
```

Aqui é separado as variáveis independentes com exceção da variável alvo e depois é definido qual é a variável alvo.

O *StandardScaler* é um padronizador que remove a média e escala os dados para a variância unitária. Isso é crucial para algoritmos que são sensíveis à escala dos dados.

Logo após, é calculado a média e o desvio padrão de cada feature, e depois os dados são transformados usando esses valores.

```
# Normalizar (padronizar) os dados (exceto variável-alvo)
features = df_codificado.drop('charges', axis=1)
alvo = df_codificado['charges']

scaler = StandardScaler()
features_scaled = scaler.fit_transform(features)
```

Os dados resultantes da padronização são convertidos de volta em um dataset, mantendo os nomes das colunas originais, sendo combinados com a variável-alvo.

```
# Combinar novamente em um DataFrame
features_scaled_df = pd.DataFrame(features_scaled, columns=features.columns)
df_final = pd.concat([features_scaled_df, alvo], axis=1)
```

O pré processamento dos dados foi bem sucedido, preparando os dados para a modelagem. A variável smoker_yes (indica se o indivíduo é fumante) é a que mais se relaciona com a variável charges.

```
Correlação com 'charges':
 charges              1.000000
smoker_yes            0.787251
age                   0.299008
bmi                   0.198341
region_southeast      0.073982
children              0.067998
sex_male              0.057292
region_northwest     -0.039905
region_southwest     -0.043210
Name: charges, dtype: float64
```

### 3.7 Regressão Linear Simples

Importando o dataset gerado na questão anterior.

```
# Carregar dados pré-processados
df = pd.read_csv(r"C:\Users\jsslu\OneDrive\Área de Trabalho\UFC\Inteligencia Artificial\git\IA-CD\AV1\datasets\regressao_ajustado.csv")
```

É calculado as correlações com a variável-alvo, removendo a correlação desta variável consigo mesma.

Se pega o valor absoluto das correlações para encontrar a feature com a maior correlação com a variável alvo.

Retorna-se o nome da feature que obtém o maior valor absoluto.

```
# Identificar a feature mais correlacionada com a variável-alvo
correlacao = df.corr()['charges'].drop('charges')
feature_mais_correlacionada = correlacao.abs().idxmax()
print(f"Feature mais correlacionada: {feature_mais_correlacionada}")
```

É separado um DataFrame contendo apenas a feature mais correlacionada e um contendo a variável-alvo.

```
# Separar variáveis independentes e alvo
X = df[[feature_mais_correlacionada]]
y = df['charges']
```

São separados os conjuntos de treino e teste, na proporção 80-20

```
# Dividir em treino e teste
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

É inicializado o modelo de regressão linear, treinando o modelo com os dados de treino e fazendo predições no conjunto de testes.

```
# Regressão Linear
model = LinearRegression()
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
```

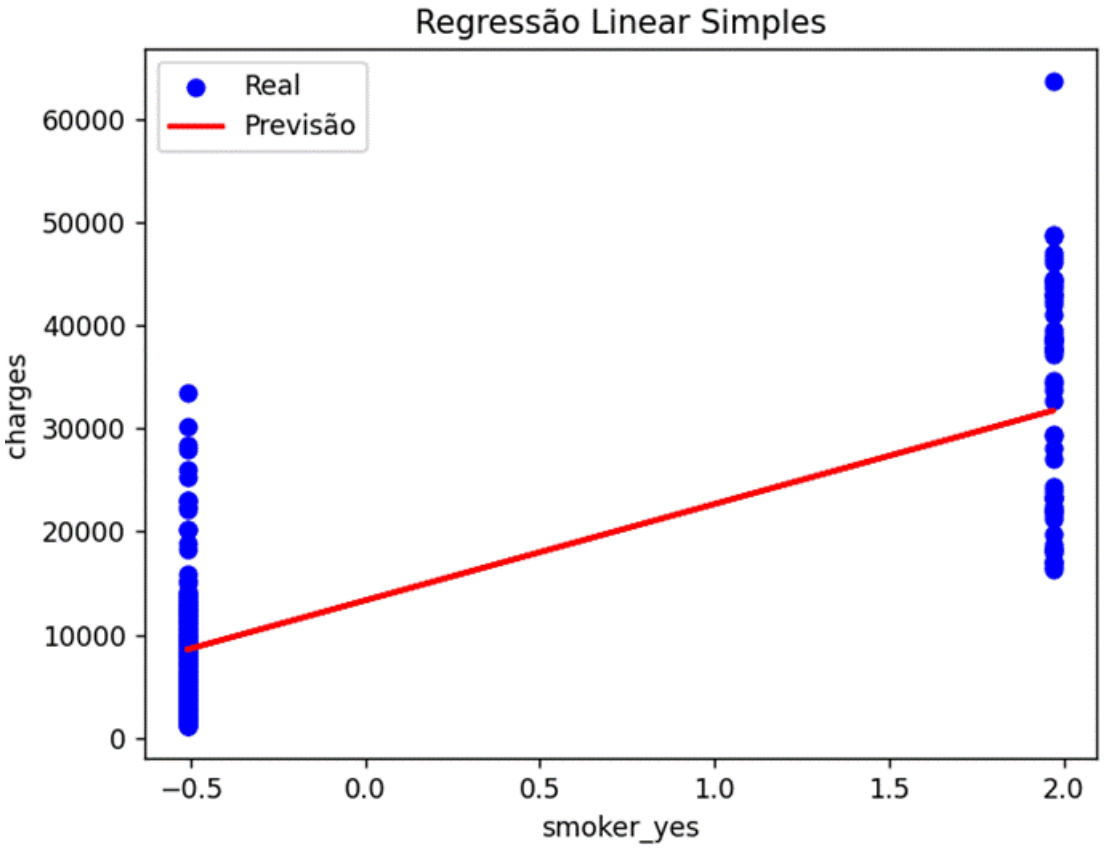
Primeiro se calcula o Erro Quadrático Médio(MSE), depois é calculado a Raiz do Erro Quadrático Médio(RMSE), que está na mesma unidade da variável-alvo e é mais interpretável que o MSE.

Calcula-se o Coeficiente de Determinação ($R^2$).

```
# Avaliação
rmse = np.sqrt(mean_squared_error(y_test, y_pred))
r2 = r2_score(y_test, y_pred)
print(f"\nRMSE: {rmse:.2f}")
print(f"R²: {r2:.4f}")
```

Ao analisar a váriavel *smoker_yes* nota-se que cerca de 66% da variância nos custos médicos é explicável pela condição de fumante do indivíduo. O RMSE indica que, em média, as previsões geradas pelo modelo se desviam cerca de $7262 dos valores reais.

```
Feature mais correlacionada: smoker_yes

RMSE: 7262.64
R²: 0.6602
```




### 3.8 Linear vs Ridge vs Lasso

Se importa novamente o dataset pré-processado, separando novamente as features e a variável-alvo.

```
# Carregar dados pré-processados
df = pd.read_csv(r"C:\Users\jsslu\OneDrive\Área de Trabalho\UFC\Inteligencia Artificial\git\IA-CD\AV1\datasets\regressao_ajustado.csv")
X = df.drop('charges', axis=1)
y = df['charges']
```

São definidos os modelos, aplicado penalidades ao Modelo Rigde e ao Modelo Lasso.

```python
# Modelos
models = {
    'LinearRegression': LinearRegression(),
    'Ridge': Ridge(alpha=1.0),
    'Lasso': Lasso(alpha=0.1)
}
```

É realizada uma validação cruzada K-Fold, dividindo o dataset em 5 partes, 4 para treino e 1 para teste.

```python
# Avaliação com Cross-Validation (5 folds)
results = []
for name, model in models.items():
    neg_mse = cross_val_score(model, X, y, scoring='neg_mean_squared_error', cv=5)
    r2 = cross_val_score(model, X, y, scoring='r2', cv=5)

    rmse_scores = np.sqrt(-neg_mse)
    results.append({
        'Modelo': name,
        'RMSE Médio': rmse_scores.mean(),
        'R² Médio': r2.mean()
    })
```

São apresentados os resultados médios de RMSE e $R^2$ para cada modelo em um dataframe.

```python
# Exibir resultados
results_df = pd.DataFrame(results)
print("\nComparação dos Modelos:")
print(results_df.sort_values(by='RMSE Médio'))
```

O resultado da validação cruzada mostra que nos três modelos o desempenho é semelhante. A pequena diferença entre eles sugere que a regularização não teve impacto significativo na performance, o que pode indicar que o modelo linear já é uma boa escolha para o dataset.

| | Modelo | RMSE Médio | R² Médio |
|---|---|---|---|
| 1 | Ridge | 6072.381790 | 0.746869 |
| 2 | Lasso | 6072.404096 | 0.746862 |
| 0 | LinearRegression | 6072.409387 | 0.746862 |

### 3.9 Coeficientes e Seleção de Atributos

Se importa novamente o dataset pré-processado, separando novamente as features e a variável-alvo.

```
# Carregar dados pré-processados
df = pd.read_csv(r"C:\Users\jsslu\OneDrive\Área de Trabalho\UFC\Inteligencia Artificial\git\IA-CD\AV1\datasets\regressao_ajustado.csv")
X = df.drop('charges', axis=1)
y = df['charges']
```

É inicializado o Modelo Lasso com *alpha=0.1*. Um valor pequeno em *alpha* indica menos regularização. Treina-se o modelo em todo o dataset.

```
# Treinar modelo Lasso
lasso = Lasso(alpha=0.1)
lasso.fit(X, y)
```

*lasso.coef_* é o atributo que contém os coeficientes que o modelo atribuiu a cada feature. É criado uma Série pandas para associar os coeficientes aos nomes das colunas. Os coeficientes são filtrados em ordem decrescente, mostrando os atributos com maior impacto.

```
# Coeficientes
coef = pd.Series(lasso.coef_, index=X.columns)
coef_nonzero = coef[coef != 0].sort_values(ascending=False)

print("\nAtributos mais relevantes (coeficientes diferentes de zero):")
print(coef_nonzero)
```

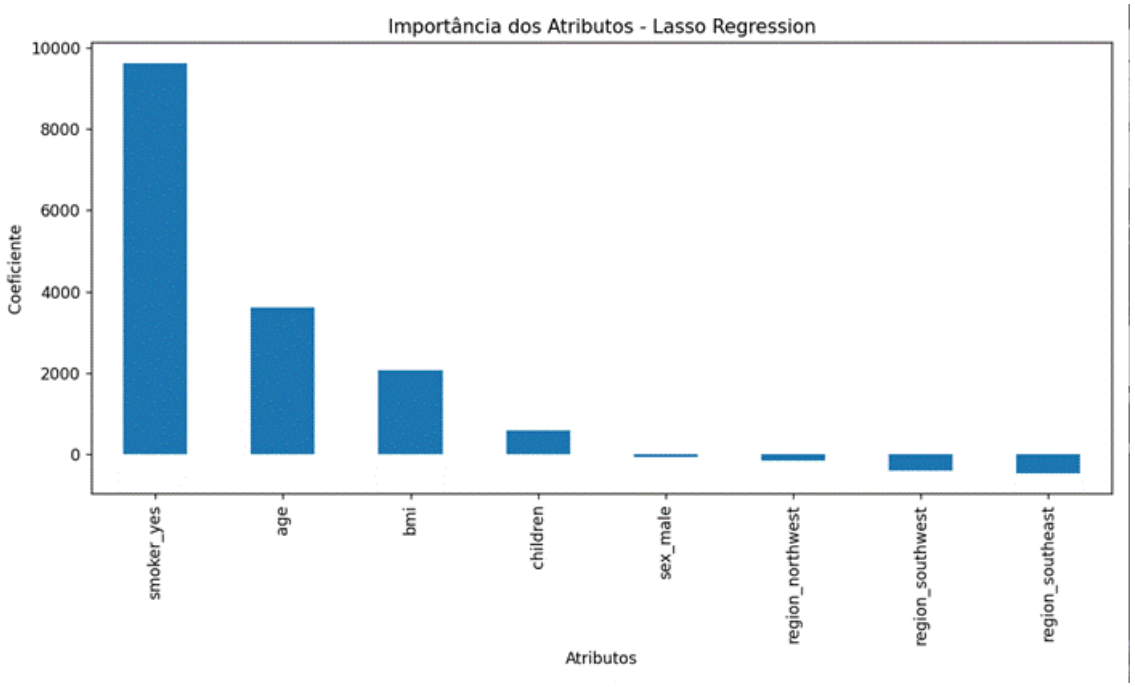
O modelo Lasso confirma a importância das seguintes variáveis:

- smoker_yes: de longe o atributo mais impactante, com um coeficiente positivo muito alto, sugerindo que ser fumante está associado a um aumento substancial nos custos médicos;
- age: indica o aumento de custos médicos de acordo com a idade;
- bmi: sugere que um peso elevado se associa a maiores custos médicos;
- indica que ter filhos também está associado a o aumento dos custos médicos, mas com menor influência.

- sex_male: com um coeficiente negativo, indica que se comparado com a cadegoria base(sexo feminino) está associado a custos menores.

```
Atributos mais relevantes (coeficientes diferentes de zero):
smoker_yes          9623.802081
Atributos mais relevantes (coeficientes diferentes de zero):
smoker_yes          9623.802081
smoker_yes          9623.802081
age                 3607.394258
age                 3607.394258
bmi                 2067.542650
children             572.895789
sex_male             -65.539505
region_northwest    -151.051206
region_southwest    -411.378792
region_southeast    -460.235480
dtype: float64
```

O gráfico reforça que *smoker_yes, age* e *bmi* são as variáveis mais importantes dos custos médicos apresentados no dataset.




## 4. Conclusões

*Os resultados esperados foram satisfeitos? Se não, qual o motivo? Qual a sua análise?*

### 4.1 Classificação

Em parte os resultados foram satisfatórios, embora alguns cálculos possam ter apresentado erros, sobretudo em relação ao KNN, o modelo pôde ser bem avaliado. A matriz de confusão e os valores de precision, recall e f1-score indicam um bom equilíbrio e estabilidade no modelo, embora haja uma leve tendência de erro na classificação de vozes femininas como masculinas

### 4.2 Regressão

Em parte, os resultados foram sim satisfatórios. Os códigos foram executados e insights foram obtidos com base na influência das variáveis. Sem sombra de dúvidas, a característica do tabagismo, além de outros aspectos como idade avançada, está fortemente atrelada aos altos custos médicos dos indivíduos contidos no dataset.