UNIVERSIDADE FEDERAL DO CEARÁ

TECHNICAL REPORT

Aluno: Luan Sales Barroso

1. Introdução

Water quality:

Este é um dataset criado a partir de dados imaginários de qualidade da água em um ambiente urbano. O conteúdo desse dataset são linhas que têm valores numéricos que dizem a quantidade de determinado ingrediente na água e as colunas dizem os nomes dos ingredientes de água sendo a última coluna, *is-safe*, a conclusão se a água é ou não segura(0 - não seguro, 1 - seguro). Neste, todas as colunas têm relevância, mas o foco principal seria a última coluna a *is-safe*. O que será analisado neste dataset, .

2. Observações

3. Resultados e discussão

Questão 1:

Para esta questão foi pedido que a biblioteca *GridSearchCV* fosse utilizada para descobrir o melhor valor para K e a melhor métrica de distância para o mesmo dataset trabalhado na AV1 e uma comparação fosse feita caso o resultado fosse diferente. Temos algo interessante aqui:

	Valor de K	Distância	Acurácia
AV1	7	Mahalanobis	91.24562281140571
AV2	12	Manhattan	88.79717222009449

Infelizmente eu não consegui rodar com a distância de Mahalanobis no código da AV2, pois dava alguns erros na hora de passar a matriz de covariância inversa e eu não consegui resolver esses problemas. O fluxo de execução do código ficou assim:

Carregar os Dados

- `load_water_quality()`: Carrega o conjunto de dados da qualidade da água.

Preparação dos Dados

- Separa os atributos preditores (X) e a variável alvo (y).
- Divide os dados em conjunto de treino (70%) e teste (30%) usando `train_test_split`, garantindo balanceamento entre as classes com `stratify=y`.

Definição do Modelo

- Cria um modelo `KNeighborsClassifier()` sem especificar hiperparâmetros.

Definição da Grade de Hiperparâmetros

- Define o intervalo de valores para o número de vizinhos (`n_neighbors` de 1 a 20).
- Especifica as métricas de distância: Euclidiana, Manhattan e Minkowski.

Configuração da Validação Cruzada

- Configura a validação cruzada com `KFold(n_splits=5, shuffle=True, random_state=42)`, dividindo os dados em 5 partes para reduzir viés nos resultados.

Busca pelos Melhores Hiperparâmetros (Grid Search)

- `GridSearchCV` testa diferentes combinações de `n_neighbors` e `metric` usando validação cruzada para encontrar a melhor configuração.
- Treina o modelo no conjunto de treino (`grid.fit(X_train, y_train)`).

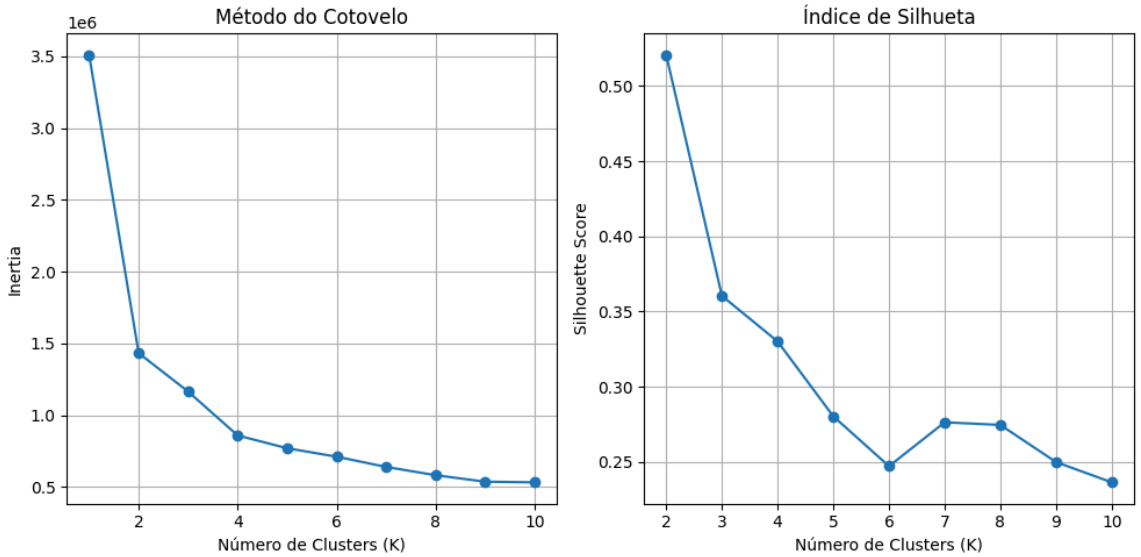
Exibir Resultados

- Exibe os melhores hiperparâmetros encontrados (`grid.best_params_`).
- Mostra a melhor acurácia obtida durante a validação cruzada (`grid.best_score_`).

1.

Questão 2:

Para esta questão foi pedido uma análise para descobrir a quantidade ideal de clusters usando o método do cotovelo e o índice da silhueta sem considerar a coluna alvo do meu dataset, indo direto ao ponto:



O método do cotovelo apresentou um gráfico “padrão” em relação a aparência mas o valor ideal segundo ele ficou algo entre 3 ou 4, aparentemente os dois valores servem bem para o número de clusters, agora o índice de silhueta mostra que o número ideal de clusters seria 2 mas para ter um melhor equilíbrio entre separação e granularidade também pode ser usado algo entre 3 ou 4. O fluxo de execução do código ficou assim:

Carregar os Dados

- `load_water_quality()`: Carrega o conjunto de dados da qualidade da água.

Inicializar Parâmetros

- Define o intervalo de valores para o número de clusters (K de 1 a 10).
- Cria listas vazias para armazenar os valores de inércia e silhouette score.

Calcular a Inércia para Diferentes Valores de K (Método do Cotovelo)

- Para cada valor de K no intervalo definido:
  - Cria e treina um modelo `KMeans` com `n_clusters=k`.
  - Armazena o valor da inércia do modelo.

Calcular o Índice de Silhueta para Diferentes Valores de K

- Para cada valor de K de 2 a 10:
    - Cria e treina um modelo `KMeans` com `n_clusters=k`.
    - Calcula o `silhouette_score` e armazena o resultado.

Gerar Gráficos para Visualização

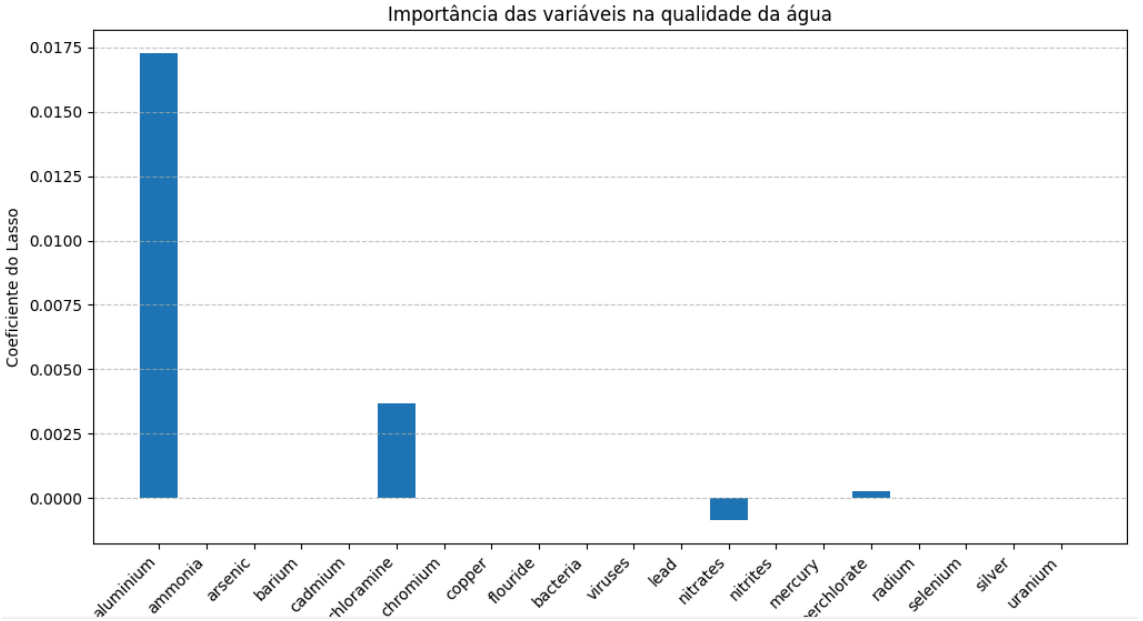
- Criar dois subgráficos (`fig, axes = plt.subplots(1, 2, figsize=(10, 5))`):
    - **Gráfico do Método do Cotovelo**:
        - Plota os valores de K versus inércia para identificar o ponto onde a redução da inércia começa a estabilizar.
    - **Gráfico do Índice de Silhueta**:
        - Plota os valores de K versus silhouette score para avaliar a qualidade dos agrupamentos.

Exibir os Gráficos

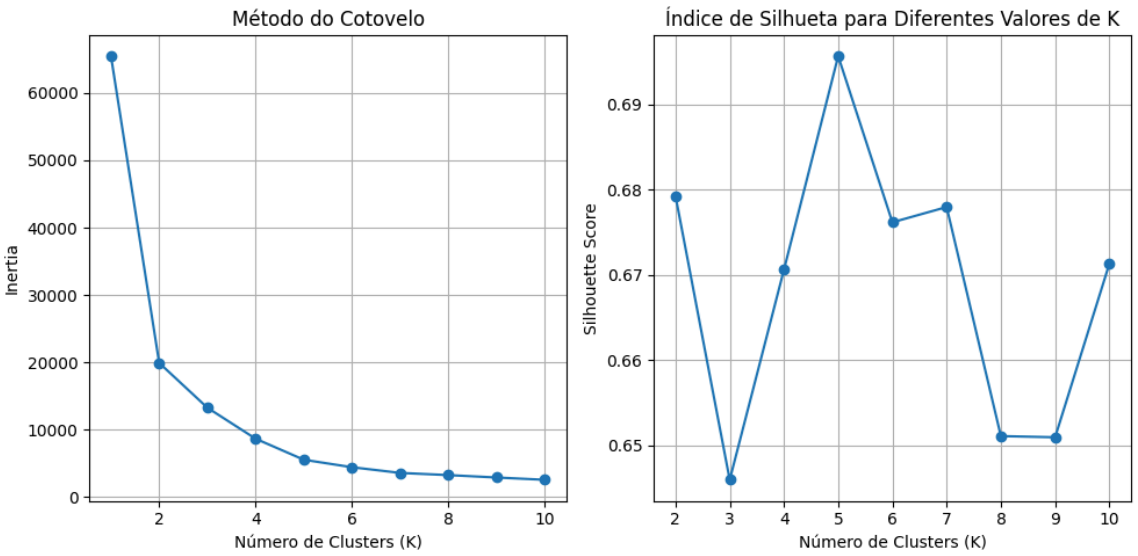
- Ajusta o layout (`plt.tight_layout()`).
- Mostra os gráficos na tela (`plt.show()`).

Questão 3:

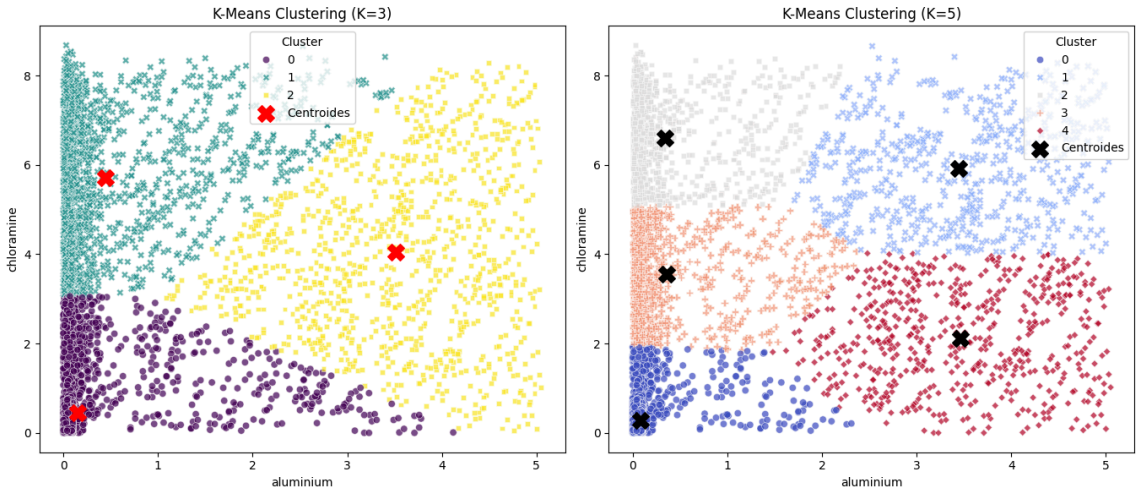
Para esta questão foi pedido que fosse utilizado o método do *Lasso* para descobrir os dois atributos mais relevantes para a predição da coluna alvo que para o meu caso ficou definido como: *aluminium, chloramine*. Como pode ser visualizado no gráfico a seguir:



Após essa definição, era necessário recalcular a quantidade de clusters com o método do cotovelo e o índice de silhueta com base nesses atributos, caso os resultados nos dois casos mudassem em comparação com a questão anterior, que foi o que aconteceu como podemos ver:



Seria necessário realizar um *scatterplot* para fazer uma análise visual. O *scatterplot* ficou assim:



Ambos utilizam as colunas que foram selecionadas como as mais relevantes. O primeiro utiliza K = 3 por conta dos resultados da questão anterior e para ter uma diferença um pouco maior em relação aos resultados do segundo que utiliza o K = 5 pois é mais nítido que esse é o valor baseado no que foi pedido. Sinceramente o segundo parece mais bem distribuído porém ainda parece muito longe do ideal, visualmente falando. O fluxo de execução do código ficou assim:

Carregamento de Dados:

- Carrega o conjunto de dados de qualidade da água utilizando a função `load_water_quality()` e armazena na variável `water_df`.

Pré-processamento de Dados:

- Divide os dados em variáveis independentes (X) e dependentes (y), excluindo a coluna `'is_safe'` em X e atribuindo a coluna `'is_safe'` a y.
- Armazena as colunas de X em `water_columns`.

Modelo Lasso:

- Criação e ajuste do modelo Lasso (`lasso = Lasso(alpha=0.1)`).
- Ajuste do modelo Lasso aos dados (`lasso.fit(X, y)`).
- Exibição dos coeficientes do modelo Lasso (importância das variáveis) em um gráfico de barras.

Método do Cotovelo e Silhueta:

- **Cálculo da Inércia (Método do Cotovelo):**

    - Criação do conjunto de dados `water_gra` com duas variáveis (`aluminium` e `chloramine`).
    - Cálculo da inércia para diferentes valores de K (1 a 10).
    - Armazenamento das inércias em uma lista.
    - Plotagem do gráfico do Método do Cotovelo.
- **Cálculo do Índice de Silhueta:**
    - Cálculo do índice de silhueta para valores de K (2 a 10).
    - Armazenamento dos valores de silhueta em uma lista.
    - Plotagem do gráfico do Índice de Silhueta.

Análise de Clustering - K-Means:

- **Primeiro Clustering (K=3):**
    - Aplicação do K-Means com K=3.
    - Atribuição das etiquetas de cluster à coluna `'cluster'` do DataFrame.
    - Criação de um gráfico de dispersão (scatter plot) para visualizar os clusters, incluindo os centróides (marcados em vermelho).
- **Segundo Clustering (K=5):**
    - Aplicação do K-Means com K=5.
    - Atribuição das etiquetas de cluster à coluna `'cluster'` do DataFrame.
    - Criação de um gráfico de dispersão (scatter plot) para visualizar os clusters, incluindo os centróides (marcados em preto).

Exibição dos Gráficos:

- Exibição dos gráficos criados, incluindo:
    - O gráfico de coeficientes do Lasso.
    - Os gráficos do Método do Cotovelo e Índice de Silhueta.
    - Os gráficos de dispersão (scatter plot) para K=3 e K=5 com os centróides de cada cluster.

Questão 4:

Para essa questão foi pedido que fosse realizado um *crosstab* para que possamos identificar como ficou a distribuição de cluster de acordo com as classes presentes na coluna alvo do meu dataset. Além disso, foi solicitado a utilização do *KMeans* e o valor

do *n_clusters* deveria ser o valor de K obtido pelo índice de silhueta. Com isso temos o seguinte resultado:

Cluster (row_0)	Água não segura (col_0 = 0)	Água segura (col_0 = 1)
Cluster 0	379	210
Cluster 1	419	196
Cluster 2	954	167
Cluster 3	960	143
Cluster 4	4372	196

Baseado nesse resultado e como visto na questão anterior, a divisão dos clusters ficou razoavelmente balanceada porém muito longe do ideal e aparentemente, baseado nas colunas *aluminium, chloramine* a predominância das amostras de água é de água não segura, mas acredito que seja um pouco mais complexo do que isso já que no meu dataset todas as colunas são importantes para a coluna alvo, que é a definição se a água é ou não segura, afinal qualquer coluna que passe do valor máximo definido para ela classifica a água como não segura. O fluxo de execução do código ficou assim:

Carregamento de Dados:

- Utiliza `load_water_quality()` para carregar o conjunto de dados de qualidade da água e armazena na variável `water_df`.

Seleção de Variáveis para Clustering:

- Seleciona as colunas `aluminium` e `chloramine` para análise e armazena em `samples`.
- Extrai a variável alvo `is_safe` e armazena em `water`.

Configuração e Treinamento do Modelo K-Means:

- Define o modelo `KMeans` com `n_clusters=5`.
- Ajusta o modelo aos dados `samples` e obtém os rótulos dos clusters resultantes (`labels`).

Criação de um DataFrame com os Resultados:

- Cria um DataFrame `df` contendo os rótulos dos clusters (`labels`) e a variável `is_safe` original (`varieties`).

Geração da Tabela Cruzada:

- Usa `pd.crosstab(labels, water)` para gerar uma tabela cruzada comparando os clusters gerados pelo K-Means com a variável `is_safe`.
- Exibe a tabela resultante com `print()`.

4. Conclusões

Baseado nas demandas e nos resultados alcançados, acredito que os resultados foram satisfatórios, é interessante ver as diferentes formas de manipular o mesmo dataset. Acredito que ainda deve ter muita coisa passível de melhoria tanto nos códigos quanto na manipulação do dataset, mas por enquanto estou satisfeito com a situação atual.

5. Próximos passos

Não tenho muito a adicionar, mas as atividades propostas foram interessantes e mais satisfatórias (opinião particular) porque acredito que dessa vez eu consegui executar melhor o que foi pedido.