CENTRO UNIVERSITÁRIO SALESIANO DE SÃO PAULO

UNIDADE LORENA

CURSO DE ENGENHARIA DE COMPUTAÇÃO

INTELIGÊNCIA ARTIFICIAL

PROFESSOR ME. WALMIR DUQUE

Breno Ryan de Andrade Fernandes

Daniel Marton Barbosa

Mariana Gonçalves de Freitas Ribeiro

João Vitor Ferreira Azevedo Pereira

"Agrupamento (Clustering) / Inteligência Artificial Generativa e PLN"

Lorena
2024

Sumário

Lorena
2024

Introdução

O presente trabalho tem como objetivo explorar e aplicar técnicas de ciência de dados e aprendizado de máquina, com ênfase no método de agrupamento, também conhecido como **clustering**. Essa técnica é amplamente utilizada para identificar padrões em dados não rotulados, organizando-os em grupos (ou clusters) baseados em semelhanças entre suas características. Dentro da vasta gama de algoritmos disponíveis para clustering, optamos pelo uso de **clusters baseados em centróides**, devido à sua simplicidade e eficácia para resolver problemas de classificação e segmentação de dados desconhecidos.

No contexto deste estudo, o clustering foi empregado com uma perspectiva prática e otimista, visando categorizar dados complexos em grupos homogêneos, que possam ser posteriormente analisados e interpretados por especialistas. A ideia central é permitir que os dados, organizados por similaridade, sirvam como base para análises mais profundas, facilitando o trabalho de profissionais ao fornecer insights iniciais de forma automatizada e objetiva. Isso torna o processo mais eficiente e reduz a carga de trabalho em etapas manuais.

Além disso, destacamos que a escolha do método de clustering por centróides, como o algoritmo **KMeans**, se deu pela sua robustez em diversos cenários práticos. Ele permite, por exemplo, a adaptação de parâmetros como o número de clusters e a métrica de distância, o que proporciona maior flexibilidade na análise e maior capacidade de atender às especificidades do conjunto de dados estudado.

Ao longo deste trabalho, buscamos não apenas demonstrar a aplicabilidade do clustering, mas também discutir seus desafios, limitações e potenciais melhorias. O objetivo é contribuir para um entendimento mais aprofundado da técnica, além de explorar como ela pode ser utilizada em cenários reais, como a segmentação de espécies biológicas ou outras aplicações relevantes para a ciência e a indústria.

1. Capítulo 1 - Agrupamento (Clustering) no Python

1.1 Business Understanding

Nosso projeto teve como objetivo entender e segmentar pinguins com base em suas características físicas (como comprimento e profundidade do bico), a fim de facilitar a análise biológica, como a diferenciação entre espécies ou populações. Nesse sentido, o modelo de escolhido foi o agrupamento (clustering), que permite identificar

padrões de grupos de pinguins com características semelhantes e prever a que cluster um novo pinguim pertencerá.

De acordo com as tarefas de Data Science e de maneira breve, visto que alguns pontos serão mais esclarecidos ao avançar no texto temos:

1. Pré-Processamento de Dados

- **Normalização/Padronização**:

  O código utiliza o StandardScaler para padronizar os dados. Essa tarefa é essencial para modelos baseados em distância, como o KMeans, pois evita que variáveis com escalas diferentes dominem o cálculo.

- **Divisão de dados**:

  A divisão entre conjuntos de treinamento e teste (Percentage Split) é um passo importante na validação de modelos, mesmo para clustering não supervisionado.

- **Transformação dos dados**:

  Transformação dos dados em uma forma numérica apropriada para análise.

2. Clustering (Agrupamento)

- **Técnicas de Clusterização**:

  Implementação do algoritmo KMeans com diferentes métodos de inicialização (random, k-means++).

  Uso de diferentes métricas de distância: Euclidiana e Manhattan.

- **Análise do Número Ótimo de Clusters**:

  O uso do WCSSE e o método do cotovelo para determinar o número ideal de clusters faz parte de um processo exploratório para avaliação de modelos.

3. Avaliação do Modelo

- **Métricas de Avaliação Interna**:

  WCSSE (Within-Cluster Sum of Squares): Avalia a compactação dos clusters.

  Silhouette Score: mede a separação entre os clusters e o quão bem os dados estão agrupados.

- **Validação Cruzada (Cross-Validation)**:

  Ainda que o clustering seja uma técnica não supervisionada, o código adaptou a ideia de validação cruzada para avaliar diferentes configurações de clusterização.

4. Visualização

- **Método do Cotovelo (Elbow Method)**:

  Produz um gráfico visual para ajudar na seleção do número ideal de clusters.

- **Exploração de Dados**:

  O código sugere a visualização de características médias dos clusters, o que pode ser feito por meio de gráficos.

5. Análise Exploratória dos Clusters

- **Interpretação das Características dos Clusters**:

  Cálculo das médias das variáveis em cada cluster para entender o perfil de cada grupo.

- **Perfilagem dos Grupos**:

  Identificação de padrões ou perfis dentro dos dados com base nos clusters.

6. Aplicação em Cenários Reais

- **Segmentação de Dados**:

  Agrupamento de espécies (como no caso dos pinguins).

- **Detecção de Padrões**:

  Identificação de padrões ocultos nos dados para apoio à tomada de decisão.

- **Análise de Dados Multivariados**:

  O código pode ser considerado apropriado para explorar e agrupar dados com várias dimensões.

7. Significado dos Clusters:

A explicação do significado de cada cluster é baseada nas características médias de cada grupo.

```python
import pandas as pd

# Função para calcular as médias das variáveis por cluster
def cluster_summary(X, labels):
    df = pd.DataFrame(X, columns=['culmen_length_mm', 'culmen_depth_mm ', 'body_mass_g ', 'flipper_length_mm '])
    df['Cluster'] = labels
    cluster_means = df.groupby('Cluster').mean()
    print("\nMédias das variáveis por cluster:")
    print(cluster_means)

# Após aplicar o KMeans, calcule as médias para cada cluster
model = KMeans(n_clusters=3, init='k-means++', random_state=42)
model.fit(Xpenguins_scaled)
labels = model.labels_

cluster_summary(Xpenguins_scaled, labels)
```

```
Médias das variáveis por cluster:
         culmen_length_mm  culmen_depth_mm   body_mass_g   flipper_length_mm
Cluster
0               -0.380418         0.611014     -0.090528           -0.612378
1                0.674719        -1.091634      0.011104            1.081051
2               -0.353574         1.550650     18.448833            0.060424
```

- Cada cluster representa grupos com diferentes características médias em termos de massa, tamanho do bico e comprimento das asas, provavelmente correspondendo a diferenças entre espécies ou subgrupos de pinguins.

8. Possíveis melhorias

Com pequenas adaptações, ele também poderia ser usado para:

- **Detecção de Anomalias**.
- **Redução de Dimensionalidade** (como prévia ao clustering)

Exemplo de como os dados estavam antes do nosso trabalho:

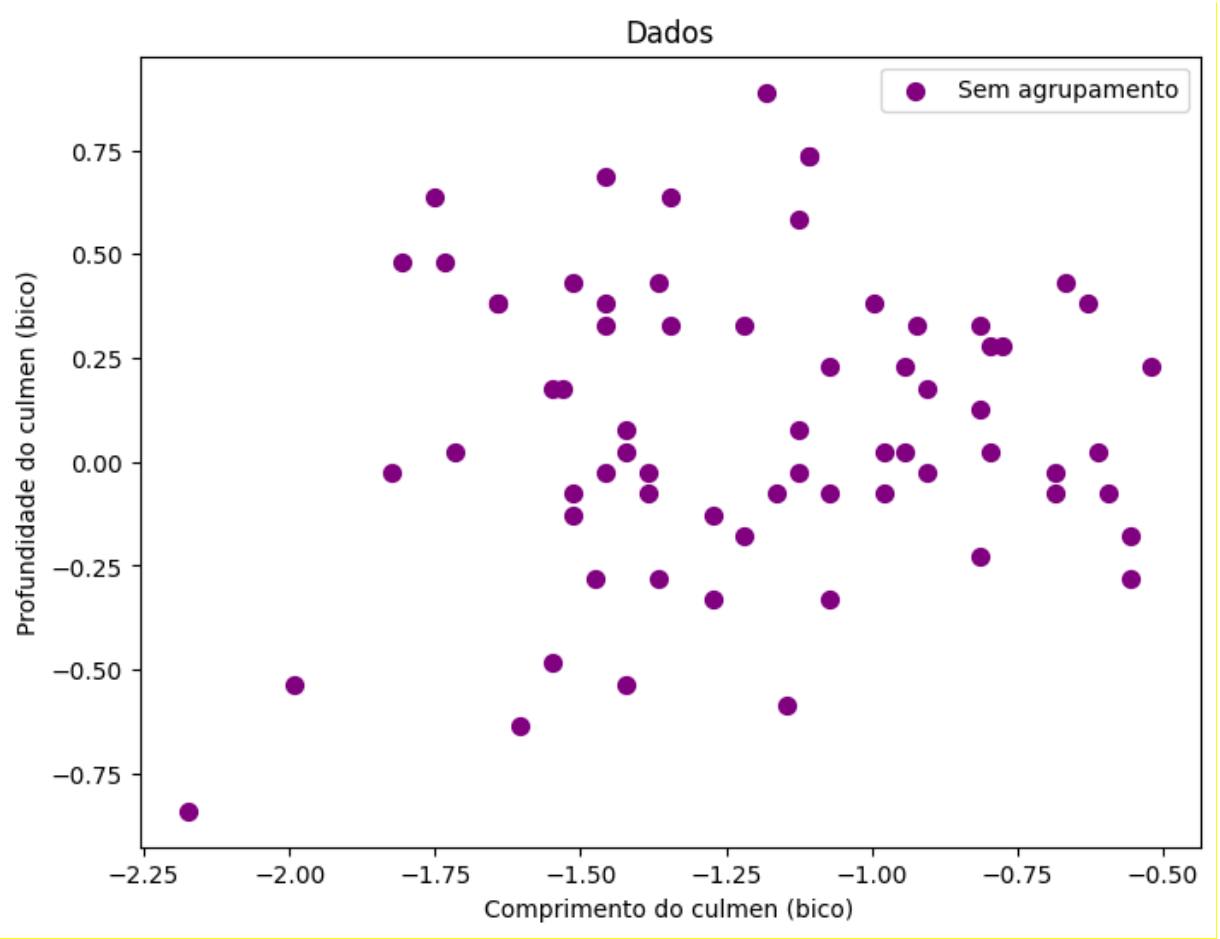

Dados
Sem agrupamento
Profundidade do culmen (bico)
Comprimento do culmen (bico)
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−2.25
−2.00
−1.75
−1.50
−1.25
−1.00
−0.75
−0.50

Tabela de Regras de Negócio e Requisitos

Item	Descrição	Atores Envolvidos
Regra de Negócio	O sistema deve segmentar pinguins com base em características físicas.	Biólogos, cientistas de dados
Requisito Funcional 1	Identificar clusters de pinguins com características semelhantes.	Biólogos, desenvolvedores
Requisito Funcional 2	Permitir a previsão de cluster para novos pinguins inseridos no sistema.	Desenvolvedores, analistas
Requisito Funcional 3	Exibir visualizações gráficas dos clusters formados para análise visual.	Cientistas de dados, biólogos
Requisito Funcional 4	Determinar o número ideal de clusters usando o método Elbow.	Cientistas de dados
Requisito Funcional 5	Criar clusters e avaliar os resultados com base nas características físicas dos pinguins.	Cientistas de dados, biólogos
Requisito Funcional 6	Uso de distância de Manhattan para formar clusters alternativos.	Cientistas de dados, desenvolvedores
Requisito Não Funcional 1	O sistema deve ser eficiente e suportar grandes volumes de dados.	Desenvolvedores, analistas de TI
Requisito Não Funcional 2	O modelo deve ser escalável para incluir novas variáveis ou características de espécies.	Desenvolvedores, analistas de TI
Requisito Não Funcional 3	Adicionar flexibilidade no cálculo de distâncias, permitindo alternar entre diferentes métricas.	Desenvolvedores, analistas de TI

Requisito Funcional 1: Identificar clusters de pinguins com características semelhantes

```
labels = kmeans.labels_
```

Requisito Funcional 2: Permitir a previsão de cluster para novos pinguins inseridos

```
# Criação de dataset de teste com uma única linha para verificação do modelo
novo_pinguim = scaler.transform([[45.5, 17.4, 210, 4500]])
cluster_predito = kmeans.predict(novo_pinguim)
print(f"O novo pinguim pertence ao cluster: {cluster_predito[0]}")
```

Lorena
2024

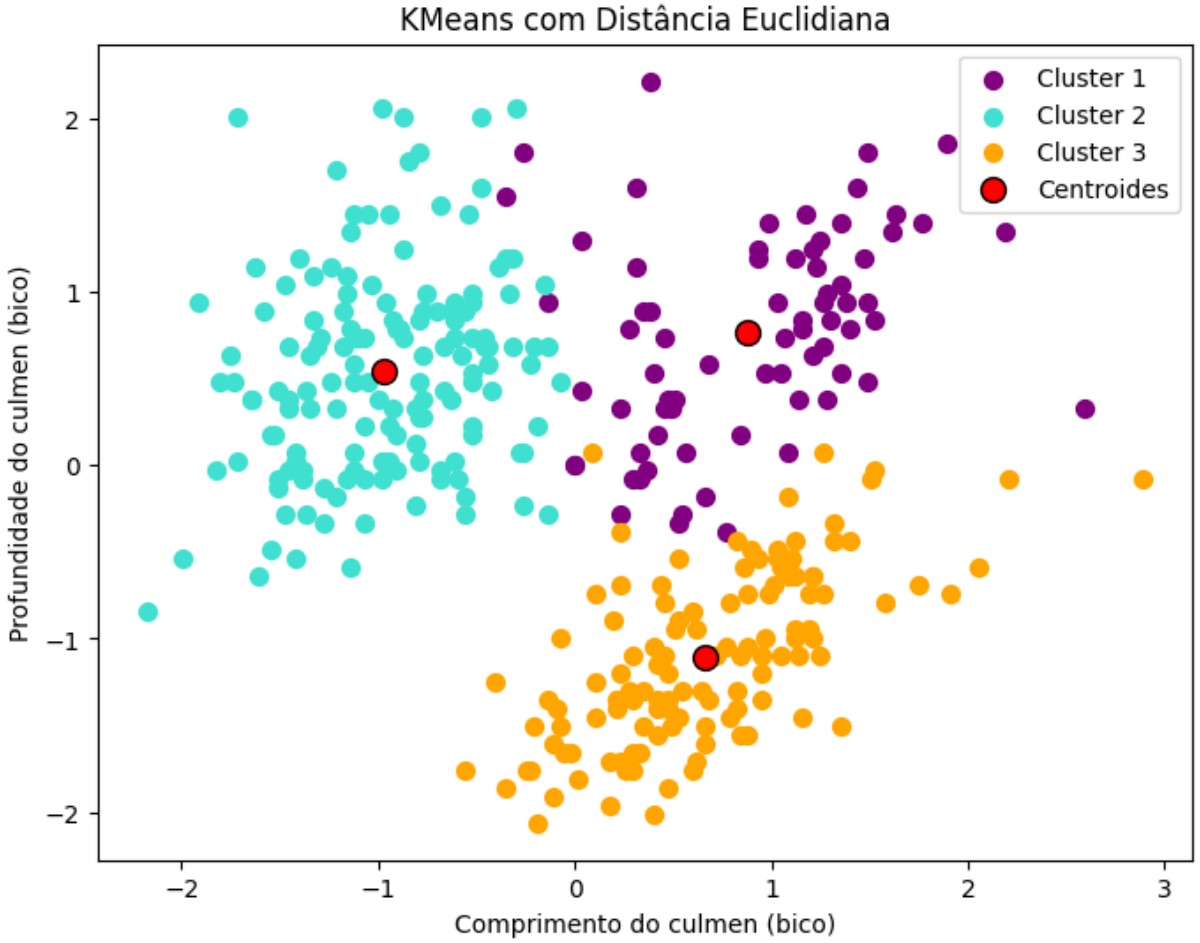
Requisito Funcional 3: Exibir visualizações gráficas dos clusters formados

Medida de Distância Euclidiana

```
# 1.4 Evaluation: Visualização e avaliação dos clusters formados
import matplotlib.pyplot as plt

# Definindo cores para cada cluster
colors = ['purple', 'turquoise', 'orange']
labels = kmeans.labels_

plt.figure(figsize=(8, 6))

# Plotando cada ponto com a cor do seu respectivo cluster
for i in range(n_clusters):  # Número de clusters = 3
    plt.scatter(
        Xpenguins_scaled[labels == i, 0],
        Xpenguins_scaled[labels == i, 1],
        s=50,
        c=colors[i],
        label=f'Cluster {i+1}'
    )
```

```python
# Plotando os centroides em uma cor destacada
plt.scatter(
    kmeans.cluster_centers_[:, 0],
    kmeans.cluster_centers_[:, 1],
    s=100,
    c='red',
    marker='o',
    edgecolor='black',
    label="Centroides"
)

# Legenda
plt.xlabel("Comprimento do culmen (bico)")
plt.ylabel("Profundidade do culmen (bico)")
plt.title("KMeans com Distância Euclidiana")
plt.legend()
plt.show()
```

Visualização



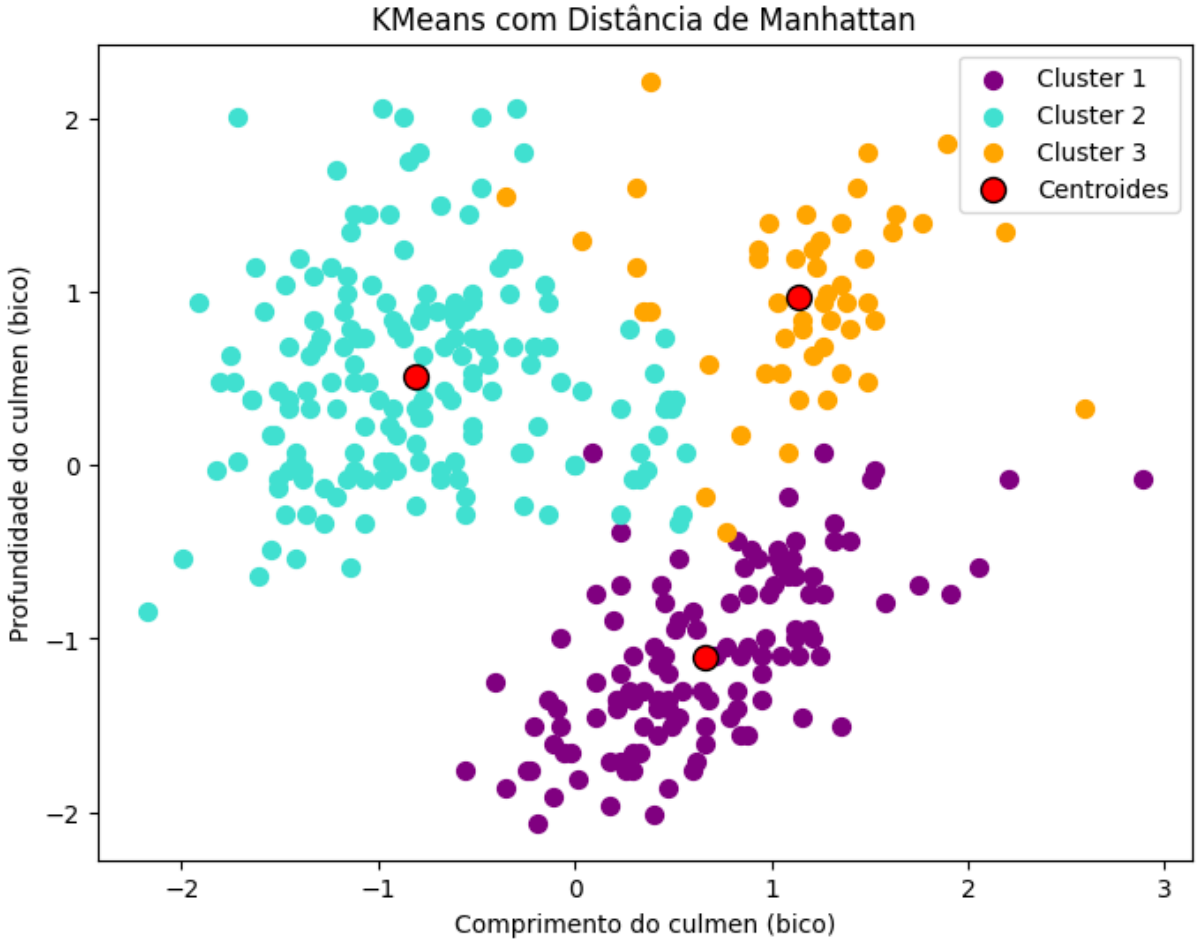
Medida de distância Manhattan

```python
# Plotando os clusters com a distância de Manhattan
colors = ['purple', 'turquoise', 'orange']

plt.figure(figsize=(8, 6))

# Plotando os pontos com cores baseadas nos clusters
for i in range(n_clusters):
    plt.scatter(
        Xpenguins_scaled[labels == i, 0],
        Xpenguins_scaled[labels == i, 1],
        s=50,
        c=colors[i],
        label=f'Cluster {i+1}'
    )

# Plotando os centroides (medoids)
plt.scatter(
    centroids[:, 0],
    centroids[:, 1],
    s=100,
    c='red',
    marker='o',
    edgecolor='black',
    label="Centroides"
)

plt.xlabel("Comprimento do culmen (bico)")
plt.ylabel("Profundidade do culmen (bico)")
plt.title("KMeans com Distância de Manhattan")
plt.legend()
```

```python
# Usando o kmeans_manhattan para calcular os clusters
centroids, labels = kmeans_manhattan(Xpenguins_scaled, n_clusters)

# Plotando os clusters com a distância de Manhattan
colors = ['purple', 'turquoise', 'orange']

plt.figure(figsize=(8, 6))

# Plotando os pontos com cores baseadas nos clusters
for i in range(n_clusters):  # Número de clusters
    plt.scatter(
        Xpenguins_scaled[labels == i, 0],
        Xpenguins_scaled[labels == i, 1],
        s=50,
        c=colors[i],
        label=f'Cluster {i+1}'
    )

# Plotando os centroides (medoids)
plt.scatter(
    centroids[:, 0],
    centroids[:, 1],
    s=100,
    c='red',
    marker='o',
    edgecolor='black',
    label="Centroides"
)
```

```python
plt.xlabel("Comprimento do culmen (bico)")
plt.ylabel("Profundidade do culmen (bico)")
plt.title("KMeans com Distância de Manhattan")
plt.legend()
plt.show()
```

Visualização:



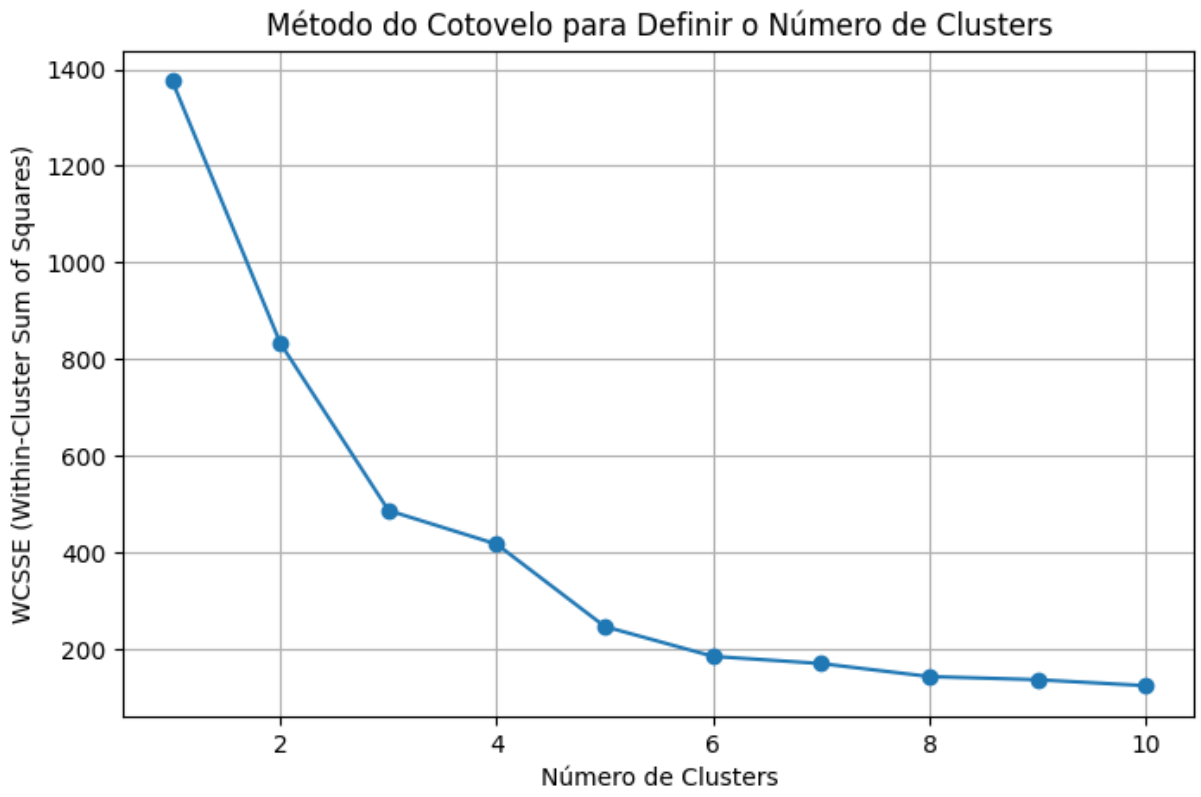
Requisito Funcional 4: Determinar o número ideal de clusters usando o método Elbow

```python
from sklearn.cluster import KMeans

wcsse = []
max_clusters = 10
for i in range(1, max_clusters + 1):
    kmeans = KMeans(n_clusters=i, init='k-means++', random_state=42)
    kmeans.fit(Xpenguins_scaled)
    wcsse.append(kmeans.inertia_)
```

Para a visualização:

```python
# Plotando o gráfico do método do cotovelo (Elbow Method)

plt.figure(figsize=(8, 5))
plt.plot(range(1, max_clusters + 1), wcsse, marker='o')
plt.title('Método do Cotovelo para Definir o Número de Clusters')
plt.xlabel('Número de Clusters')
plt.ylabel('WCSSE (Within-Cluster Sum of Squares)')
plt.grid(True)
plt.show()
```

Visualização:



Requisito Funcional 5: Criar clusters e avaliar os resultados com base nas características físicas dos pinguins

```
# Definindo o número de clusters com base no Elbow
n_clusters = 3

# Parâmetros alternativos
param_combinations = [
    {"init": "random", "n_clusters": n_clusters, "metric": "euclidean"},
    {"init": "k-means++", "n_clusters": n_clusters, "metric": "euclidean"},
]
```

Requisito Funcional 6: Uso de distância de Manhattan para formar cluster alternativos

```
from scipy.spatial.distance import cdist
import numpy as np

# Função para calcular o KMeans com distância de Manhattan
def kmeans_manhattan(X, n_clusters, max_iters=300, random_state=None):
    np.random.seed(random_state)

    # Inicializando os centroides aleatoriamente
    centroids = X[np.random.choice(X.shape[0], n_clusters, replace=False)]

    for _ in range(max_iters):
        # Calculando a matriz de distâncias de Manhattan
        distances = cdist(X, centroids, metric='cityblock')  # 'cityblock' é a distância de Manhattan

        # Atribuindo cada ponto ao cluster mais próximo
        labels = np.argmin(distances, axis=1)

        # Atualizando os centroides
        new_centroids = np.array([X[labels == i].mean(axis=0) for i in range(n_clusters)])

        # Verificando se houve mudança nos centroides
        if np.all(centroids == new_centroids):
            break

        centroids = new_centroids

    return centroids, labels
```

```
# Usando o kmeans_manhattan para calcular os clusters
centroids, labels = kmeans_manhattan(Xpenguins_scaled, n_clusters)
```

Requisitos Não Funcionais:

- O modelo precisa ser eficiente para grandes volumes de dados, uma vez que esse número tende a crescer.

- Deve ser escalável para poder agregar mais variáveis ou espécies.

1.2 Data Understanding e Data Preparation

Variáveis (Features):

Valores mínimos

```
1    Xpenguins.min()
✓ 0.0s
culmen_length_mm       32.1
culmen_depth_mm        13.1
flipper_length_mm    -132.0
body_mass_g          2700.0
dtype: float64
```

Valores maxímos

```
1    Xpenguins.max()
✓ 0.0s
culmen_length_mm       59.6
culmen_depth_mm        21.5
flipper_length_mm    5000.0
body_mass_g          6300.0
dtype: float64
```

Valores médios

```
1    Xpenguins.mean()
✓ 0.0s
culmen_length_mm       43.921930
culmen_depth_mm        17.151170
flipper_length_mm     214.014620
body_mass_g          4201.754386
dtype: float64
```

- culmen_length_mm tipo float — Representando o comprimento do bico.

- culmen_depth_mm tipo float — Representando a profundidade do bico.

- flipper_length_mm tipo float — Representando o comprimento da barbatana.

- body_mass_g tipo float — Peso do pinguim.

- Excluímos da análise a feature “sex”, pois, nesse caso, a diferença entre os sexos não influencia o tipo de espécie.

Target e Classes:

- Não há variável-alvo (target) porque o problema é de agrupamento (clustering).

- Os clusters formados (0, 1 e 2) são as "classes" que o algoritmo deve identificar.

Preparação de Dados:

Verificamos como o estava o dataset com esse input:

```
# Exibindo as primeiras linhas do dataset
penguins.head()
```

Obtivemos esse output:

	culmen_length_mm	culmen_depth_mm	flipper_length_mm	body_mass_g	sex
0	39.1	18.7	181.0	3750.0	MALE
1	39.5	17.4	186.0	3800.0	FEMALE
2	40.3	18.0	195.0	3250.0	FEMALE
3	NaN	NaN	NaN	NaN	NaN
4	36.7	19.3	193.0	3450.0	FEMALE

Como demonstra a imagem existem valores nulos (NaN).

Aplicamos outra verificação:

```
# Exibe o número de valores ausentes por coluna
print(penguins.isnull().sum())
```

Output:

```
culmen_length_mm     2
culmen_depth_mm      2
flipper_length_mm    2
body_mass_g          2
sex                  9
dtype: int64
```

Normalizamos utilizando a classe StandardScaller da biblioteca scikit-learn para fazer o tratamento:

```
from sklearn.preprocessing import StandardScaler

# Seleção e descrição das variáveis numéricas para o modelo de clustering
Xpenguins = penguins[['culmen_length_mm', 'culmen_depth_mm', 'flipper_length_mm', 'body_mass_g']]

# Tratamento de valores ausentes (substituição por média da coluna)
Xpenguins = Xpenguins.fillna(Xpenguins.mean())

# Normalização das variáveis numéricas
scaler = StandardScaler()
Xpenguins_scaled = scaler.fit_transform(Xpenguins)
```

Nova verificação:

```
# Exibe o número de valores ausentes por coluna
print(Xpenguins.isnull().sum())
```

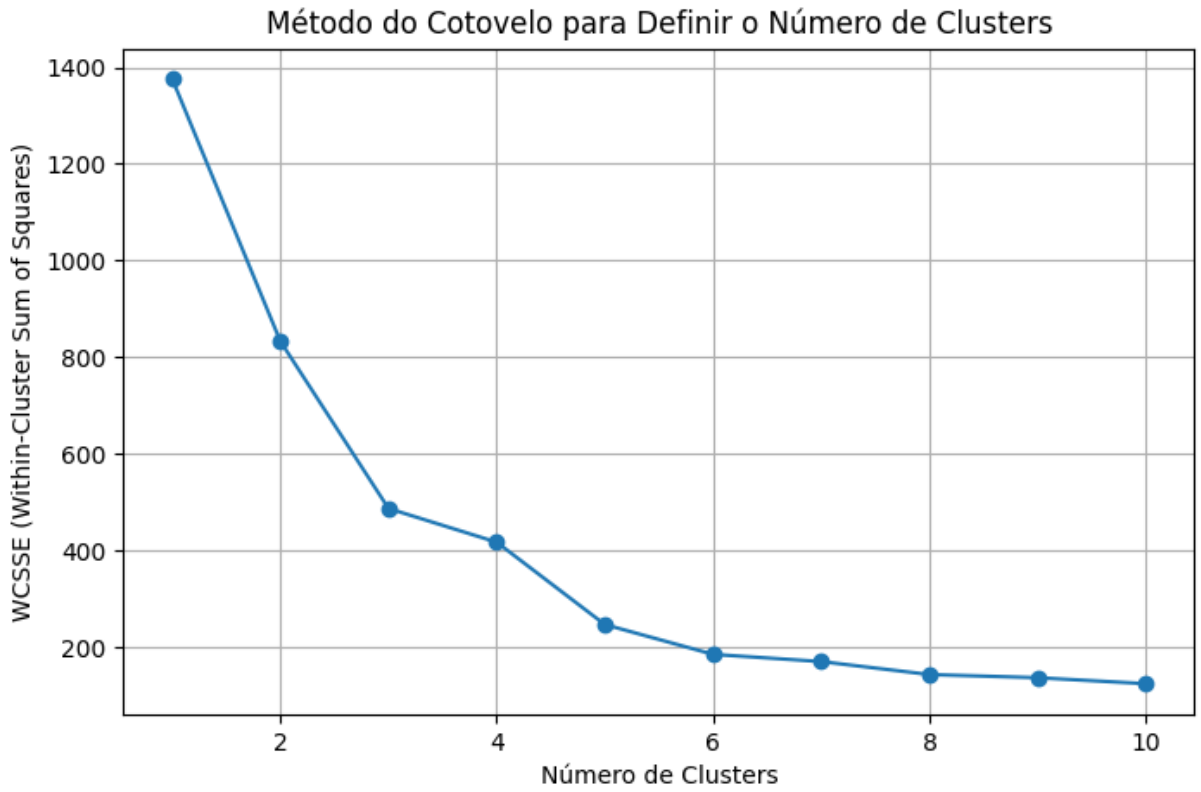
Output:

```
culmen_length_mm     0
culmen_depth_mm      0
flipper_length_mm    0
body_mass_g          0
dtype: int64
```

1.3 Modeling e Evaluation

Método do Cotovelo para Determinação de K

O método do cotovelo (Elbow Method) é utilizado para encontrar o número ideal de clusters. Esse método nos revela que, a partir de certo ponto, o aumento de clusters se torna menos significativo do que no início.



Este gráfico ajuda a identificar o ponto em que a diminuição da WCSS (within-cluster sum of squares) começa a ser mais suave, indicando o número ideal de clusters. Percebe-se que, a partir de 3 clusters, a diferença não diminuiu significativamente, por isso definimos em nosso modelo o K = 3.

Teste de Parâmetros

Inicialização	Medida de Distância	Resultado
Random	Euclidiana	Modelo com inicialização random e distância euclidean: Centroides dos clusters: [[0.88014656 0.76091601 0.19198729 -0.43831116] [0.65914815 -1.10319208 0.01222874 1.09494883] [-0.96943473 0.54299781 -0.10286725 -0.69208256]] WCSSE: 654.5232338830422 ----------
Kmeans	Euclidiana	Modelo com inicialização k-means++ e distância euclidean: Centroides dos clusters: [[0.88014656 0.76091601 0.19198729 -0.43831116] [-0.96943473 0.54299781 -0.10286725 -0.69208256] [0.65914815 -1.10319208 0.01222874 1.09494883]] WCSSE: 654.5232338830422 ----------
Random	Manhattan	Modelo com inicialização aleatória e distância de Manhattan: Centroides dos clusters: [[1.0979033 -0.63290264 0.02777633 1.61269011] [-0.38445028 0.60474313 -0.00663886 -0.62038409] [0.29421691 -1.44087106 -0.00280141 0.6169]] WCSSE (Manhattan): 531.234673312759 ---------- O novo pinguim pertence ao cluster: 2

Kmeans	Manhattan	Modelo com inicialização KMeans++ e distância de Manhattan: Centroides dos clusters: [[-3.80418100e-01 6.11013600e-01 -9.05283284e-02 -6.12377715e-01] [6.74718852e-01 -1.09163410e+00 1.11038004e-02 1.08105077e+00] [-3.53573761e-01 1.55064965e+00 1.84488327e+01 6.04240905e-02]] WCSSE (Manhattan): 568.525925518727 --- O novo pinguim pertence ao cluster: 0

Medida de distância Euclidiana (Random e kmeans) :

```python
from sklearn.metrics.pairwise import manhattan_distances
param_combinations = [
    {"init": "random", "n_clusters": n_clusters, "metric": "euclidean"},
    {"init": "k-means++", "n_clusters": n_clusters, "metric": "euclidean"},
]
```

Medida de distância Manhattan:

```python
from scipy.spatial.distance import cdist
import numpy as np

# Função para calcular o KMeans com distância de Manhattan
def kmeans_manhattan(X, n_clusters, max_iters=300, random_state=None):
    np.random.seed(random_state)

    # Inicializando os centroides aleatoriamente
    centroids = X[np.random.choice(X.shape[0], n_clusters, replace=False)]

    for _ in range(max_iters):
        # Calculando a matriz de distâncias de Manhattan
        distances = cdist(X, centroids, metric='cityblock')  # 'cityblock' é a distância de Manhattan

        # Atribuindo cada ponto ao cluster mais próximo
        labels = np.argmin(distances, axis=1)
```

```
        # Atualizando os centroides
        new_centroids = np.array([X[labels == i].mean(axis=0) for i in range(n_clusters)])

        # Verificando se houve mudança nos centroides
        if np.all(centroids == new_centroids):
            break

        centroids = new_centroids

    return centroids, labels
```

Nesse ponto da análise, o que despertou nossa curiosidade foi o fato de que somente com a inicialização Random e a distância Manhattan é que o "Novo pinguim" foi classificado para o cluster 2, sendo esse teste o que teve o melhor índice de WCSS, de ~531.

Tabela de Estratégias de Testes

Inicialização	Distância	Validação	Resultado
Random	Euclidiana	Percentage_split	Percentage Split: Random e Euclidean Modelo (random, euclidean) - Silhouette Score: 0.39620981446300
Kmeans	Euclidiana	Percentage_split	Percentage Split: KMeans++ e Euclidean Modelo (k-means++, euclidean) - Silhouette Score: 0.3962098144630079
Random	Manhattan	Percentage_split	Modelo (random, manhattan) - Silhouette Score: 0.3836143000032205
Kmeans	Manhattan	Percentage_split	Modelo (k-means++, manhattan) - Silhouette Score: 0.3836143000032205
Random	Euclidiana	Cross_validation	Cross Validation: Random e Euclidean Modelo (random, euclidean) - Cross-Validation Silhouette Score: 0.4690610487228878
Kmeans	Euclidiana	Cross_validation	Cross Validation: KMeans++ e Euclidean Modelo (k-means++, euclidean) - Cross-Validation Silhouette Score: 0.47669001125137855
Random	Manhattan	Cross_validation	Cross Validation: Random e Manhattan Modelo (random, manhattan) - Cross-Validation Silhouette Score: 0.44407995725812305
Kmeans	Manhattan	Cross_validation	Cross Validation: KMeans++ e Manhattan Modelo (k-means++, manhattan) - Cross-Validation Silhouette Score: 0.48807449888015303

Métrica Silhouette Score

Utilizamos o Silhouette Score, pois é uma medida que considera tanto a coesão quanto a separação dos clusters. Para isso, foi necessário importar da biblioteca Sklearn. Se, por exemplo, chamarmos coesão de x e separação de y, o Silhouette Score para um único ponto é obtido pela fórmula:

s = (y - x) / (max(x, y))

O valor dessa métrica varia de -1 a 1. Valores próximos de 1 indicam que o ponto está bem ajustado ao seu próprio cluster e mal ajustado aos clusters vizinhos. Se o valor é próximo de zero, temos um ponto que está próximo a um limite de decisão entre dois clusters. Já um valor negativo indica que o ponto pode ter sido atribuído ao cluster errado. Além disso, podemos calcular o Silhouette score para todo o conjunto de pontos.

Testando com um Novo Pinguim (usando um dataset com apenas uma linha)

```
# Criação de dataset de teste com uma única linha para verificação do modelo
novo_pinguim = scaler.transform([[45.5, 17.4, 210, 4500]])
cluster_predito = kmeans.predict(novo_pinguim)
print(f"O novo pinguim pertence ao cluster: {cluster_predito[0]}")
```

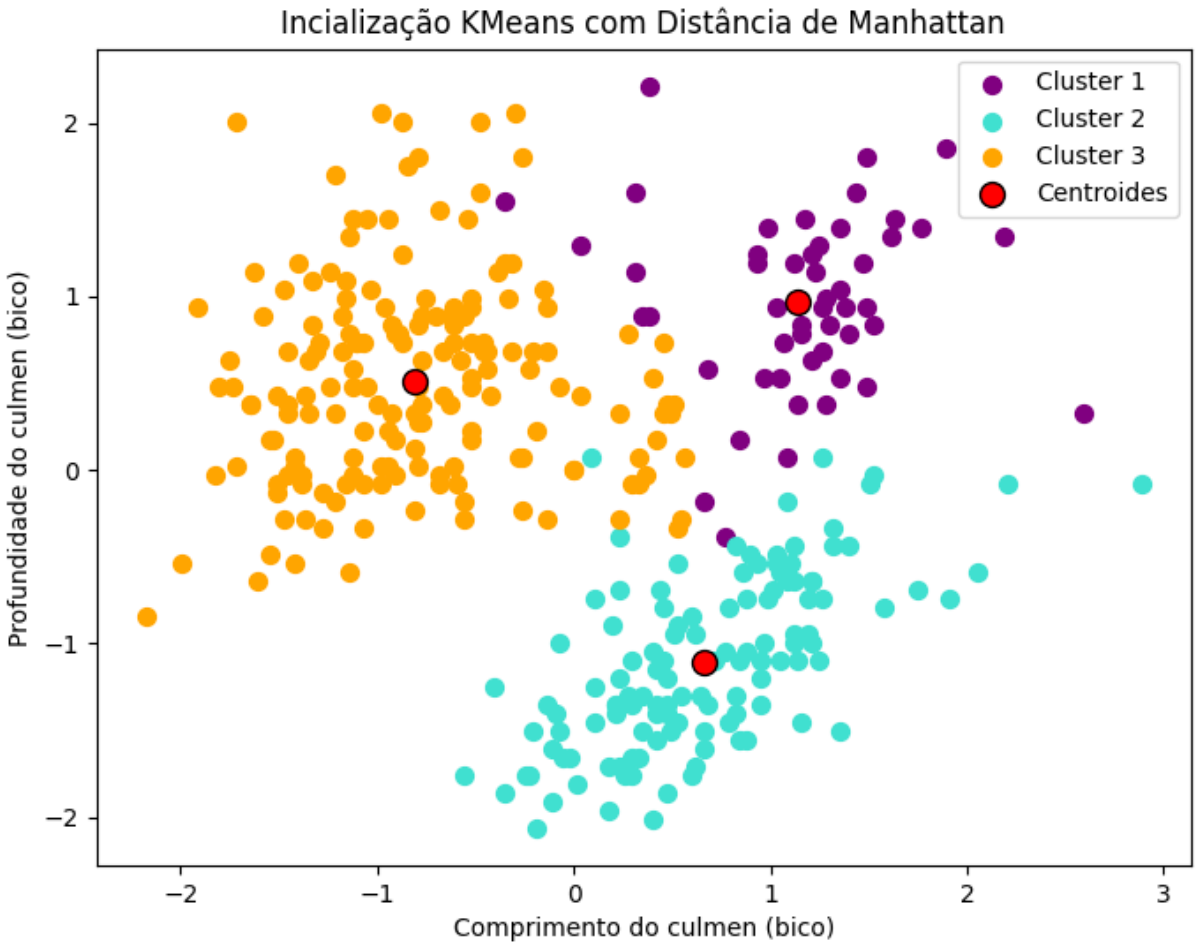
Melhor Modelo ?

Depende.

O WCSSE mede a compactação dos clusters: quanto menor o WCSSE, mais próximos estão os dados dentro de cada um. Um WCSSE menor é, portanto, desejável, mas não deve ser analisado de maneira isolada. Precisamos equilibrá-lo com outras métricas, como o Silhouette Score, que avalia separação e coesão.

Revisão do Modelo Kmeans + Manhattan com validação Cross_Validation

- Silhouette Score: 0.48807).



Se priorizarmos a compactação (WCSSE)

Temos os modelo Random + Manhattan e Kmeans + Euclidean.

Kmeans + Euclidean

- Percentage_split + WCSSE:47.035

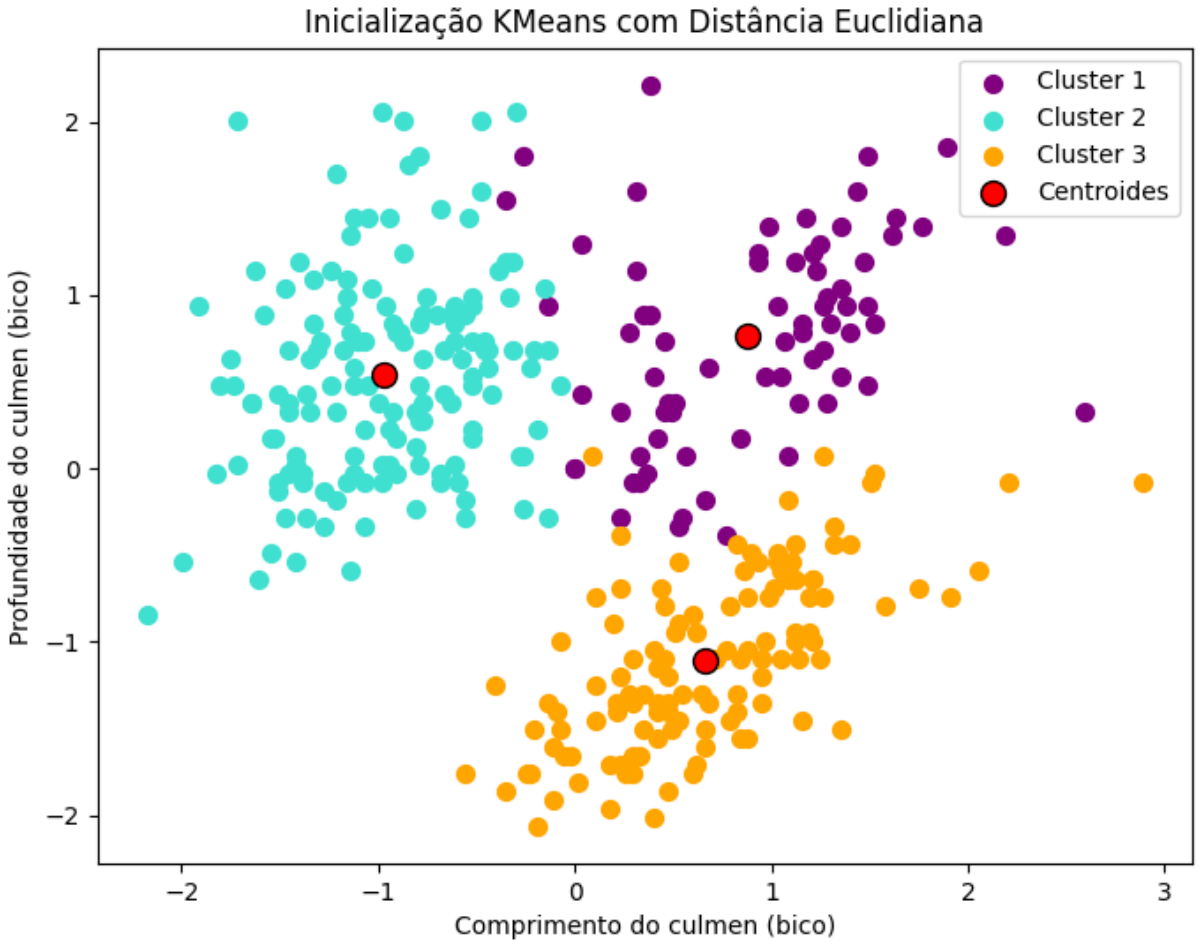
Random + Manhattan

- Cross_validation + WCSSE:149.81

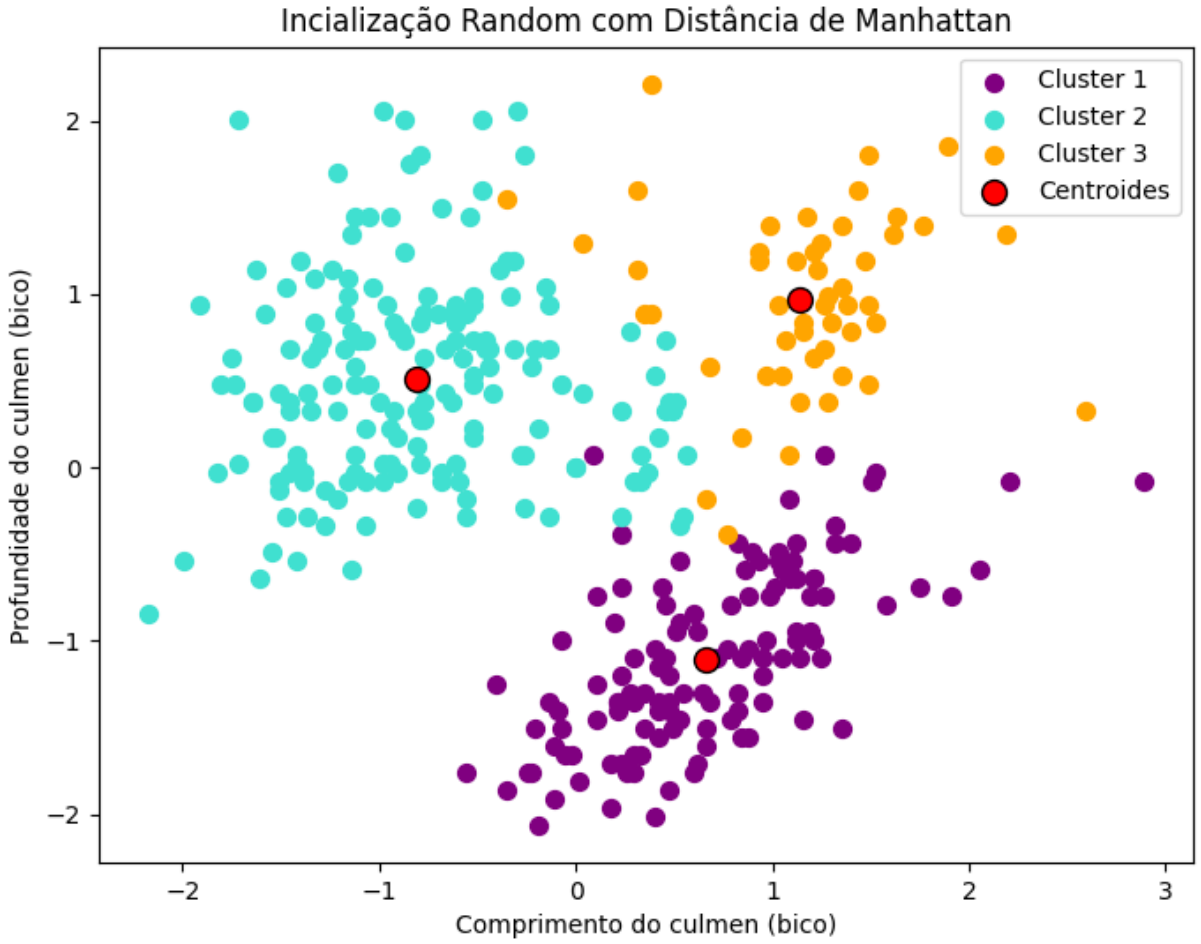
O melhor modelo depende do objetivo específico:

Lorena
2024

Se a separação clara entre clusters for mais relevante (por exemplo, para fins interpretativos ou classificatórios), escolha KMeans++ com Euclidean.



Se a compactação intra-cluster for mais importante como, minimizar custos ou distância em análises operacionais escolha Random + Manhattan.



Em cenários gerais, a separação e a compactação equilibradas fazem do KMeans++ com Euclidean a melhor escolha. No entanto, se o objetivo específico priorizar a compactação extrema, o Random + Manhattan é o vencedor.

Conclusão

Podemos afirmar que nosso trabalho apresentou um bom grau de desafio, mesmo considerando que, em teoria, a técnica de clustering baseada em centróides é uma das abordagens mais acessíveis para implementação e compreensão, especialmente por profissionais de áreas que não sejam diretamente ligadas à ciência de dados ou aprendizado de máquina. Apesar dessa aparente simplicidade, o desenvolvimento deste projeto nos proporcionou uma rica experiência prática e acadêmica, envolvendo desde o tratamento inicial dos dados até a aplicação e avaliação de modelos, passando por diversas etapas críticas que exigiram atenção e ajustes meticulosos.

Durante o processo, enfrentamos diferentes etapas que, embora previsíveis, demandaram um esforço significativo, como a normalização dos dados, a escolha das métricas de avaliação mais adequadas e a adaptação dos algoritmos para melhor desempenho em nossos conjuntos de dados. Além disso, testamos diferentes configurações de parâmetros para identificar aquelas que melhor se adequavam às características do problema abordado, o que nos permitiu compreender mais profundamente os impactos dessas decisões no resultado final.

Embora estejamos satisfeitos com o resultado alcançado, reconhecemos que há melhorias a serem feitas para elevar o desempenho e a qualidade geral da solução. Entre essas possíveis melhorias, podemos destacar o ajuste mais fino de parâmetros, a adoção de técnicas avançadas, como redução de dimensionalidade prévia ao clustering, e a integração de outras métricas de avaliação que complementem as utilizadas. Essas mudanças poderiam ampliar a robustez e a aplicabilidade do modelo, tornando-o ainda mais eficiente e adaptável a novos cenários.

Em síntese, acreditamos que o trabalho foi conduzido de maneira sólida e meticulosa, atendendo aos objetivos propostos e demonstrando a aplicabilidade prática da técnica. Mais do que isso, ele nos proporcionou um aprendizado valioso, reforçando nossa disposição e entusiasmo para continuar explorando e aprimorando técnicas de ciência de dados e aprendizado de máquina no futuro.

Introdução

No trabalho a seguir, buscamos explorar o processamento de linguagem natural (PLN) aplicado ao conjunto de dados “Notícias publicadas no Brasil”, com foco no tema esportes. A escolha desse gênero textual foi motivada por sua acessibilidade e estrutura previsível, facilitando o tratamento e análise do conteúdo. Os objetivos principais incluem a categorização de subtemas, resumos de notícias para facilitar a leitura, identificação de tendências e a personalização de conteúdo para o desenvolvimento de sistemas de recomendação. Além disso, o processo envolveu técnicas robustas de limpeza textual, lematização, tokenização e geração de embeddings, permitindo a criação de uma base de dados vetorial para consulta semântica eficiente. Este documento descreve detalhadamente o fluxo de trabalho, desde o carregamento e filtragem do dataset até a construção de um modelo capaz de identificar similaridades entre textos e fornecer insights relevantes.

2. Capítulo 2 - Inteligência Artificial Generativa

2.1 - **Definição de texto a ser trabalhado, explicação do seu gênero textual e objetivos de negócio**

O texto utilizado foi obtido por meio do conjunto de dados “Notícias publicadas no Brasil”, disponibilizado pelo site Kaggle. Esse conjunto foi criado a partir da raspagem de dados (web scraping) em diversos sites, como G1 e SportTV, feita pelo criador do conjunto. Dentro desse conjunto, há alguns temas específicos de notícias (política, esporte, economia e famosos). Optamos por filtrar o tema em "esporte", pois se trata de uma temática mais próxima da grande maioria do público, que não necessita de um repertório mais técnico para lidar com assuntos como "economia", nem causa polêmica com "política" ou textos, geralmente de menor profundidade intelectual, "famosos". Segundo o site Mundo Educação, notícia é um texto jornalístico cuja função é relatar acontecimentos cotidianos, com forte presença de elementos narrativos e descritivos. A notícia é um dos gêneros textuais mais dinâmicos e lidos atualmente. Partindo disso, a escolha do gênero textual se deu justamente por sua estrutura previsível e sua linguagem formal, o que facilitou o processamento (remoção de ruídos). Os objetivos de negócio foram a filtragem e categorização de subtemas para melhor análise, resumo de notícias para facilitar a leitura e obtenção de insights, análise de tendências e identificação de palavras-chave ou temas que estão se destacando nas notícias de esportes ao longo do tempo. Além disso, foi possível personalizar o conteúdo e desenvolver um sistema de recomendação para exibir notícias relevantes aos interesses dos usuários.

2.2 - PLN - Tratamento do texto: Remoção de Ruídos, Homogeneização e Stopwords

Como mencionado acima, antes mesmo de iniciarmos o processo de tratamento do texto, precisamos filtrar o conjunto de dados para usar apenas o que, em nosso limitado entendimento, julgamos fundamental para começarmos o trabalho propriamente dito. Com o recurso do framework Pandas, extraímos o CSV e, a partir dele, criamos nosso conjunto de dados filtrado somente com o assunto “esportes”. Para o tratamento de ruídos, foi utilizada a biblioteca nativa do Python Re para lidar com expressões regulares e para formatação de palavras em minúsculas, evitando diferenciação entre palavras como "Tenista" e "tenista" e suas inúmeras variações. Além disso, somado ao processo de tokenização, utilizamos a biblioteca NLTK (Natural Language Toolkit) através do corpus padrão "stopwords", adaptado para a língua portuguesa. Juntamente com a nossa lista de palavras stopwords customizadas, “a, e, i, o, u, né, aí, tá, então”, não tivemos muitos problemas ou necessidades especiais, como gírias, coloquialismostal tal como “né e tu”, pois se trata de um texto jornalístico. Valores numéricos ou caracteres especiais “!,@, $” também foram tratados, assim como outras particularidades utilizadas em nossa linguagem cotidiana sem perceber. Por fim, foram feitos testes para encontrar similaridade entre textos após a integração de modelos de embeddings fornecidos pela biblioteca Sentence Transformers e bancos de dados vetoriais disponibilizados por outra biblioteca chamada ChromaDB. Assim, acreditamos termos produzido um texto com alguma qualidade de homogeneização.

Carregamento dos dados

```
# Passo 1: Carregar o dataset
import pandas as pd

dados = pd.read_csv("Historico_de_materias.csv")
```

Teste para conferir as colunas antes da filtragem

```
# Para exibir somente os tipos de assunto
valores_unicos = dados['assunto'].unique()

print("Valores únicos na coluna 'assunto':")
print(valores_unicos)
```

Criação do novo dataframe somente com os dados relacionados ao tema esportes

```
# Passo 2: Criar um novo dataframe somente para o assunto 'esporte'
dados_esportes = dados[dados['assunto'] == 'esportes']
```

Tratamento das stopwords

```
# Passo 4: Configurar palavras de parada

# Essa biblioteca oferece ferramentas para análise de texto, como stemmers (radicalizadores),
#  stopwords, tokenizadores, e entre outras funções
import nltk

# Importa o conjunto de palavras de parada
from nltk.corpus import stopwords

# Verificar stopwords
# lista padrão da biblioteca pré-definida de palavras irrelevantes
nltk.download('stopwords')

# Carrega as stopwords em português do NLTK e as converte em um conjunto Python
api_stop_words = set(stopwords.words('portuguese'))
# Incluir vogais isoladas não parece contribuir para o significado.
minhas_stop_words = {'a', 'e', 'i', 'o', 'u', 'né', 'aí', 'tá', 'então'}
# Combinação das palavras de parada padrão do NLTK com "minhas_stop_words"
stop_words = api_stop_words | minhas_stop_words
```

Formatação das palavras em minúsculas, limpeza, lematização e tokenização

```
# Passo 5: Função de tratamento de texto

# Biblioteca padrão do Python para expressões regulares
import re

# Expressões regulares permitem manipular e encontrar padrões em strings,
# como remover caracteres indesejados ou verificar formatos específicos (ex.: datas, e-mails).
# Para realizar a limpeza do texto removendo caracteres que não sejam letras ou espaços.

# função usada para pré-processar textos antes de realizar análises mais avançadas.
def tratamento_pln(texto):
    # Converte todo o texto para letras minúsculas
    texto = texto.lower()
    # Substitui todos os caracteres indesejados por uma string vazia
    texto = re.sub(r'[^a-zA-Záéíóú\s]', '', texto)
    # Lematiza as palavras de acordo com o pipeline da spacy e divide o texto em tokens
    doc = nlp(texto)
    # Cria uma lista de palavras tratadas que não são pontuações nem stopwords
    clean_tokens = [token.lemma_ for token in doc if token.text not in stop_words and not token.is_punct]
    # Combina a lista de palavras limpas (clean_tokens) em uma única string
    clean_text = ' '.join(clean_tokens)
    return clean_text
```

Tratamento de valores ausentes

```
# Aplicar o tratamento
# Usada para dividir um texto grande em partes menores (chunks) de tamanho controlado.
from langchain.text_splitter import RecursiveCharacterTextSplitter

# Substitui valores ausentes por uma string vazia e converte todos os valores da coluna
# "conteudo_noticia" para o tipo string
dados_esportes['conteudo_noticia'] = dados_esportes['conteudo_noticia'].fillna("").astype(str)
# Aplica a função tratamento_pln para cada valor na coluna "conteudo_noticia"
dados_esportes['conteudo_tratado'] = dados_esportes['conteudo_noticia'].apply(tratamento_pln)
```

2.3 - PLN - Stemming / Lemmatization

Quanto à escolha entre Stemming ou Lemmatization, a resposta foi avaliada com certa intuição, uma vez que, ao utilizar o stemming (radicalização), há uma grande redução de palavras, o que resulta em maior desempenho. No entanto, por se tratar de um texto informativo, optamos pela lematização, garantindo uma maior compreensão e integridade da escrita original. Para tal, foi utilizada a biblioteca Spacy com o modelo pré-treinado spacy.load("pt_core_news_sm").

Importação do spacy para lematizar e tokenizar

```
# Passo 3: Carregar spaCy
# Biblioteca de processamento de linguagem natural para tarefas como tokenização,
# lematização entre outras.
import spacy

# Modelo pré-treinado otimizado para a língua portuguesa
nlp = spacy.load("pt_core_news_sm")
```

2.4 - Chunk / Embedding

Quanto ao limite de chunk, mantivemos a quantidade máxima e o número máximo de sobreposição de caracteres, conforme utilizado em sala de aula, por conta da própria natureza do texto trabalhado. Poderíamos reduzir a redundância apenas reduzindo o overlap ou aumentar a complexidade semântica do texto aumentando o número de chunks. Essa última parte foi mais bem vista em esboços desse trabalho, quando utilizamos texto do tipo romance, com mais figuras de linguagem, metáforas e poesia.

Divisão do texto em chunks

```
# Passo 6: Dividir o texto em partes (chunks)
# Para controlar o tamanho
# Cria um objeto text_splitter da classe RecursiveCharacterTextSplitter com os seguintes parâmetros:
# chunk_size=200 tamanho máximo dos pedaços de texto (chunks) como 200 caracteres.
# chunk_overlap=50: os pedaços de texto terão uma sobreposição de 50 caracteres entre eles.
text_splitter = RecursiveCharacterTextSplitter(chunk_size=200, chunk_overlap=50)
# Cria uma lista de chunks de todos os textos na coluna "conteudo_tratado"
chunks = [
    # Itera por cada notícia (linha) na coluna conteudo_tratado
    chunk for noticia in dados_esportes['conteudo_tratado']
    # Para cada notícia, o split_text divide o texto em partes menores (chunks),
    # de acordo com os parâmetros definidos anteriormente (tamanho de 200 caracteres e
    #  sobreposição de 50 caracteres)
    for chunk in text_splitter.split_text(noticia)
]
```

Resultados

Análise semântica de singularidade de cosseno

```python
from sklearn.metrics.pairwise import cosine_similarity

# Função para encontrar notícias semelhantes e calcular a similaridade de cosseno
def encontrar_similares(texto):
    # Gerar embeddings para a nova notícia
    texto_tratado = tratamento_pln(texto)
    embedding_novo = model.encode([texto_tratado])

    # Consultar no ChromaDB
    resultados = collection.query(
        query_embeddings=embedding_novo,
        n_results=5,  # Quantidade de resultados desejados
    )

    # Calcular similaridade de cosseno entre o embedding da nova notícia e os encontrados
    similaridades = cosine_similarity(embedding_novo, embeddings).flatten()

    # Retornar as notícias semelhantes, seus metadados e os graus de similaridade
    documentos = resultados['documents']
    metadados = resultados['metadatas']
    ids = [int(metadata['id']) for metadata in metadados]
    graus_similaridade = similaridades[ids]

    return documentos, metadados, graus_similaridade

# Testar a função
nova_noticia = "O jogador marcou um gol decisivo na final do campeonato."
noticias_similares, metadatas_similares, similaridades = encontrar_similares(nova_noticia)

# Exibir os resultados
print("Notícias semelhantes e graus de similaridade:\n")
for noticia, metadata, similaridade in zip(noticias_similares[0], metadatas_similares[0], similaridades):
    print(f"Notícia: {noticia}")
    print(f"Grau de Similaridade (Cosseno): {similaridade:.4f}\n")
```

Saida:

```
Notícia: "Jogador fez história com um gol que garantiu o título ao time."
Grau de Similaridade (Cosseno): 0.7984

Notícia: "O campeonato foi decidido com um gol nos acréscimos."
Grau de Similaridade (Cosseno): 0.8301

Notícia: "A estrela do time marcou dois gols importantes na semifinal."
Grau de Similaridade (Cosseno): 0.8476

Notícia: "Na final do campeonato, o time venceu após um gol decisivo na prorrogação."
Grau de Similaridade (Cosseno): 0.9107

Notícia: "O atacante garantiu a vitória com um gol incrível no último minuto da partida."
Grau de Similaridade (Cosseno): 0.9523
```

```
Notícia: "Com um gol de cabeça, o jogador garantiu a vitória histórica na prorrogação."
Grau de Similaridade (Cosseno): 0.7215
```

```
Notícia: "O destaque da final foi um gol épico no último minuto do segundo tempo."
Grau de Similaridade (Cosseno): 0.7329
```

```
Notícia: "A final histórica foi marcada por um gol que emocionou os torcedores."
Grau de Similaridade (Cosseno): 0.7420
```

```
Notícia: "O jogador se destacou ao garantir a vitória com um gol incrível na final."
Grau de Similaridade (Cosseno): 0.7654
```

```
Notícia: "A torcida vibrou quando o gol do título foi marcado no último lance."
Grau de Similaridade (Cosseno): 0.7802
```

Conclusão

A aplicação de inteligência artificial generativa no contexto de notícias esportivas revelou-se eficaz tanto no processamento quanto na análise de dados textuais. As etapas detalhadas de limpeza, tratamento e geração de embeddings permitiram transformar um grande volume de dados em informações organizadas e relevantes. A escolha por técnicas de lematização e a utilização de ferramentas avançadas como spaCy, Sentence Transformers e ChromaDB garantiram a qualidade dos resultados, permitindo consultas semânticas e a identificação de padrões no conteúdo. Por fim, este trabalho destaca como o uso de IA pode simplificar a análise de textos jornalísticos, oferecendo soluções práticas para problemas reais, como personalização de conteúdo e sistemas de recomendação. As metodologias aqui empregadas podem servir de base para futuras pesquisas ou aplicações em outros contextos e gêneros textuais.

6. Referências Bibliográficas

GLOBO ESPORTE. Nadal precisa de mais de duas horas para vencer número 74 do mundo. *ge.globo*, 13 jan. 2014. Disponível em: https://ge.globo.com/tenis/noticia/2014/01/nadal-precisa-de-mais-de-duas-horas-para-vencer-numero-74-do-mundo.html. Acesso em: 16 nov. 2024.

DATA GEEKS. Embeddings. *Data Geeks*, [s.d.]. Disponível em: https://www.datageeks.com.br/embeddings/. Acesso em: 16 nov. 2024.

ELASTIC. O que é Word Embedding? *Elastic*, [s.d.]. Disponível em: https://www.elastic.co/pt/what-is/word-embedding. Acesso em: 16 nov. 2024.

ANCHIÊTA, Rafael; NETO, Francisco A. R.; MARINHO, Jeziel C.; MOURA, Raimundo. **PLN: das técnicas tradicionais aos modelos de deep learning**. In: **TÍTULO DO LIVRO**. [s.d.]. Capítulo 1. *Sociedade Brasileira de Computação*. Disponível em: https://books-sol.sbc.org.br/index.php/sbc/catalog/download/79/341/600?inline=1. Acesso em: 16 nov. 2024.

BARBOSA, Wellington. Processamento de Linguagem Natural (PLN). *Wellbar - Computação*, 17 jan. 2023. Mini curso de Processamento de Linguagem Natural. Disponível em: https://www.youtube.com/watch?v=q8NomaZCpjs&list=PLxKmxm_IWVIQCoWxMr-0WoB1gz3o0MhUI Acesso em: 16 nov. 2024

ESCOLA DNC. **Clusterização: o guia definitivo para análise de dados não supervisionada**. Escola DNC, [s.d.]. Disponível em: https://www.escoladnc.com.br/blog/clusterizacao-o-guia-definitivo-para-analise-de-dados-nao-supervisionada/. Acesso em: 18 nov. 2024.

DATA GEEKS. **Clustering**. Data Geeks, [s.d.]. Disponível em: https://www.datageeks.com.br/clustering/. Acesso em: 18 nov. 2024.

Lorena
2024

AWARI. **Aprenda sobre clustering em machine learning**. Awari, [s.d.]. Disponível em: https://awari.com.br/aprenda-sobre-clustering-em-machine-learning/. Acesso em: 18 nov. 2024.

MAGALHÃES, Lúcia Helena de; SOUZA, Renato Rocha. **Agrupamento automático de notícias de jornais on-line usando técnicas de machine learning para clustering de textos no idioma português**. *Múltiplos Olhares em Ciência da Informação*, [s.d.]. Disponível em: https://periodicos.ufmg.br/index.php/moci/article/view/19170/16237. Acesso em: 18 nov. 2024