Universidade da Beira Interior
Departamento de Informática

Nº Trabalho Prático 4: *Unsupervised Learning*

Elaborado por:

João Miguel Baltazar Martins

Orientador:

Professor Doutor Hugo Pedro Proença

20 de dezembro de 2023

Acrónimos

CSV *Comma-separated values*

SOM *Self-Organizing Map*

t-SNE *t-Distributed Stochastic Neighbor Embedding*

DBSCAN *Density-Based Spatial Clustering of Applications with Noise*

WCSS *Within-Cluster-Sum-of-Squares*

BMU *Best Matching Unit*

Conteúdo

Lista de Tabelas

Lista de Figuras

Lista de Excertos de Código

Capítulo 1

Introdução

1.1 Enquadramento

O presente relatório aborda o trabalho prático quatro da unidade curricular de *Machine Learning*, que se concentra na aplicação de técnicas de aprendizagem não supervisionada a um conjunto de dados, *Credit Card Dataset for Clustering* disponível no *Kaggle* e no *site* da unidade curricular.

O conjunto de dados disponibilizado contém 18 variáveis, incluindo informações sobre saldos (`BALANCE`), frequência de compras (`BALANCE_FREQUENCY`), adiantamentos em dinheiro (`CASH_ADVANCE`), limites de crédito (`CREDIT_LIMIT`), entre outras.

1.2 Objetivo

Este trabalho prático tem como objetivo principal a identificação de grupos consistentes de clientes com base em padrões de comportamento semelhantes.

Para isso, será necessário:

1. Utilizar estratégias de *clustering*, como o *K-Means*, DBSCAN ou SOM para obter `k` grupos consistentes de clientes. Sendo `k` o valor apropriado.

1.3 Organização do Documento

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

2. O segundo capítulo – **Desenvolvimento** – tem três secções dedicadas ao *script* desenvolvido para o *K-Means*, DBSCAN e SOM, respetivamente, acompanhado das justificações necessárias para os valores dos hiperparâmetros relativos a cada técnica de *clustering*.

3. O terceiro capítulo – **Exposição e Análise dos Resultados** – são expostos e analisados os resultados obtidos para cada técnica de *clustering* implementada.

4. O quarto capítulo – **Conclusão** – refere as conclusões principais a tirar deste trabalho, como, por exemplo, o método de *clustering* que se adaptou melhor ao *dataset*, bem como uma reflexão sobre um trabalho futuro.

Capítulo 2

Desenvolvimento

2.1 Introdução

Neste capítulo, será apresentado as partes do *script* relevantes na implementação das diferentes técnicas de *clustering*, nomeadamente, *K-Means*, DBSCAN e SOM, assim como os valores dos hiperparâmetros relativos a cada técnica de *clustering*.

2.2 *Script K-Means*

Nesta secção será exposto e explicada as partes de maior relevância do *script* que permite fazer o *clustering* usando o *K-Means*, a partir do *dataset* do problema.

Bibliotecas Necessárias

O trecho de código ilustrado na figura 2.1 mostra as bibliotecas necessárias para a manipulação de dados, pré-processamento, *clustering* e métricas para avaliação do mesmo.

```
import pandas as pd
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score
```

Excerto de Código 2.1: Bibliotecas importadas *K-Means*

Leitura e Tratamento dos Dados

No trecho de código abaixo, ilustrado na figura 2.2, é lido o ficheiro *Comma-separated values* (CSV) que contém o conjunto de dados e armazena-o num *DataFrame* da biblioteca *pandas*.

```
df = pd.read_csv('/content/drive/MyDrive/CC GENERAL.csv')
```

Excerto de Código 2.2: Carrega e lê os dados do *dataset*

A parte do tratamento dos dados é ilustrado no trecho de código da figura 2.3.

Nas primeiras duas linhas a coluna `'CUST_ID'` é removida, pois é um identificador e por isso não é útil para o *clustering*. Caso haja valores em falta, estes são preenchidos utilizando o método de preenchimento para a frente, *'forward fill'*.

Nas duas últimas linhas os dados são padronizados (e não normalizados). Isto resulta em que cada *feature* do *DataFrame* foi padronizada de forma a ter uma média de zero e um desvio padrão de um.

```
df = df.drop('CUST_ID', axis=1)
df.fillna(method ='ffill', inplace = True)
scaler = StandardScaler()
df_scaled = scaler.fit_transform(df)
```

Excerto de Código 2.3: Tratamento dos dados do *dataset*

Dados Necessários para a Determinação do Melhor Número de *Clusters* `'K'`

A determinação do melhor número de *clusters* `'k'` a ser usado consoante as caraterísticas do *dataset* é feita pelo trecho de código ilustrado na figura 2.2.

Utiliza o índice de silhueta para determinar o número ótimo de clusters.

- Pontuação de Silhueta - é uma medida de quão semelhante um objeto é ao seu próprio *cluster* em comparação com outros *clusters*. Os valores da silhueta variam no intervalo de -1 a 1. Um valor de 1 significa que a instância está bem dentro do seu próprio *cluster* e longe de outros *clusters*, enquanto um valor de -1 significa que a instância está mais próxima do seu *cluster* vizinho do que do *cluster* ao qual está atribuída. Quanto maior a pontuação da silhueta, melhor o agrupamento.

Justificação do critério de paragem: Ao correr várias vezes o *script* com diferentes intervalos de *clusters* verificou-se que depois do 10 este já não evoluía de forma a que justifica-se uma iteração mais prolongada.

```
silhouette_scores = []
n_cluster = 1
max_clusters = 10

for n_cluster in range(2, max_clusters+1):
    kmeans = KMeans(n_clusters = n_cluster, n_init=10)
    cluster_labels = kmeans.fit_predict(df_scaled)
    score = silhouette_score(df_scaled, cluster_labels)
    silhouette_scores.append(score)
```

Excerto de Código 2.4: Determina o melhor número de *clusters* `'k'` a ser usado.

Método do Cotovelo (*Elbow Method*) *K-Means*

Conceitos a serem tomados em conta antes da explicação do código:

- Método do Cotovelo - é uma técnica utilizada para determinar o número ótimo de *clusters* na *clustering k-means*. O método consiste em traçar a variação em função do número de *clusters* e escolher o "cotovelo"da curva como o número de *clusters* a ser utilizado. O ponto do cotovelo é o ponto em que o gráfico começa a diminuir de forma mais lenta.

  A ideia é a seguinte: à medida que o número de *clusters* aumenta, a soma das distâncias dos pontos aos seus respetivos centros de *cluster* diminui. Essa soma de distâncias também é conhecida como *Within-Cluster-Sum-of-Squares* (WCSS). Inicialmente, com menos *clusters*, a diminuição do WCSS é significativa. Mas, após um certo número de clusters (o ponto do cotovelo), a diminuição do WCSS não é tão pronunciada. Esse ponto é considerado como o número apropriado de clusters.

Esta parte do código, ilustrada em 2.5, utiliza o Método do Cotovelo para determinar o número ótimo de *clusters* para o agrupamento *K-Means*.

Esta envolve a execução da *clustering K-Means* no conjunto de dados para uma gama de valores de `'k'` (de 1 a 10, neste caso), e para cada valor de `'k'`, calculando a soma das distâncias quadradas de cada ponto até o centro atribuído (inércia). Esses valores são armazenados na lista de distorções.

Envolve também, o desenho da curva do cotovelo. O eixo-x representa o número de *clusters* e o *eixo-y* representa a inércia. O "cotovelo"no gráfico é geralmente considerado como um indicador do número apropriado de clusters.

```
distortions = []
K = range(1,10)
for k in K:
    kmeanModel = KMeans(n_clusters=k)
    kmeanModel.fit_predict(df_scaled)
    distortions.append(kmeanModel.inertia_)

plt.figure(figsize=(16,8))
plt.plot(K, distortions, 'bx-')
plt.xlabel('k')
plt.ylabel('Distortion')
plt.title('The Elbow Method showing the optimal k')
plt.show()
```

Excerto de Código 2.5: Determina o número ótimo de *clusters* para o agrupamento *K-Means* (*Elbow Method*) e desenha a *Elbow Curve*

No trecho de código ilustrado em 2.6 é calculado o número ótimo de *clusters* encontrando o ponto que está mais afastado da linha que conecta o primeiro e o último ponto no gráfico de distorção.

```
first_point = [1, distortions[0]]
last_point = [10, distortions[-1]]
distances = []
for i in range(len(distortions)):
    x = i + 1
    y = distortions[i]
    numerator = abs((last_point[1]-first_point[1])*x - (last_point[0]-
        first_point[0])*y + last_point[0]*first_point[1] - last_point
        [1]*first_point[0])
    denominator = ((last_point[1] - first_point[1])**2 + (last_point[0]
        - first_point[0])**2)**0.5
    distances.append(numerator/denominator)
optimal_clusters_elbow = distances.index(max(distances)) + 1
```

Excerto de Código 2.6: Calculo do número ótimo de clusters *K-Means* (*Elbow Method*) e desenha a *Elbow Curve*

No trecho de código ilustrado em 2.7 realiza *clustering K-Means* com o número ótimo de *clusters* determinado pelo Método do Cotovelo.

```
kmeans_elbow = KMeans(n_clusters=optimal_clusters_elbow, n_init=10)
cluster_labels_elbow = kmeans_elbow.fit_predict(df_scaled)
```

Excerto de Código 2.7: *clustering K-Means* com o número ótimo de clusters determinado pelo Método do Cotovelo

Pontuação de Silhueta (*Silhouette Score*) *K-Means*

Esta parte do código, ilustrada em 2.8, utiliza a Pontuação de Silhueta para determinar o número ótimo de *clusters* para o agrupamento *K-Means*. Aqui está uma breve descrição:

1. **Determinar *Clusters* Ótimos:** Encontra o índice da pontuação de silhueta máxima na lista `silhouette_scores` e adiciona 2 a ele para obter o número ótimo de *clusters* de acordo com a Pontuação de Silhueta. Isso ocorre porque a faixa de números de *clusters* começou a partir de 2.

2. **Realizar Agrupamento *K-Means*:** O agrupamento *K-Means* é realizado com o número ótimo de *clusters*. O objeto `KMeans` é criado com `n_clusters` configurado para o número ótimo de *clusters* e `n_init` configurado para 10. O método `fit_predict` é chamado no objeto `KMeans` para ajustar o modelo aos dados normalizados e prever o *cluster* para cada ponto de dados.

3. **Calcular Pontuação de Silhueta:** A pontuação de silhueta para o agrupamento é calculada usando a função `silhouette_score`. Essa pontuação é uma medida de quão semelhante um objeto é ao seu próprio *cluster* em comparação com outros *clusters*, com pontuações mais altas indicando um melhor agrupamento.

```
n_clusters_silhouette = silhouette_scores.index(max(silhouette_scores))
    + 2

# Perform K-Means clustering with n clusters
kmeans_silhouette = KMeans(n_clusters=n_clusters_silhouette, n_init=10)
cluster_labels_silhouette = kmeans_silhouette.fit_predict(df_scaled)

# Calculate evaluation metrics for n clusters
silhouette_score_silhouette = silhouette_score(df_scaled,
    cluster_labels_silhouette)
```

Excerto de Código 2.8: Determina o número ótimo de *clusters* para o agrupamento *K-Means* (*Silhouette Score*)

2.3 *Script* DBSCAN

Nesta secção será exposto e explicada as partes de maior relevância do *script* que permite fazer o *clustering* com o DBSCAN a partir do *dataset* do problema. De referir que as partes comuns (leitura e tratamento dos dados) deste

ao *script K-Means* não serão referidas nesta secção porque já foram referenciadas em 2.2.

Bibliotecas Necessárias

O trecho de código ilustrado na figura 2.9 mostra as bibliotecas necessárias para a manipulação de dados, pré-processamento, `DBScan` para *clustering*, `silhouette_score` para avaliação da qualidade do *clustering*, `TSNE` para o algoritmo *t-SNE* que é uma técnica de redução de dimensionalidade que é particularmente adequada para a visualização de conjuntos de dados de alta dimensão.

```
import pandas as pd
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import DBSCAN
from sklearn.metrics import silhouette_score
from sklearn.manifold import TSNE
import numpy as np
import matplotlib.pyplot as plt
```

Excerto de Código 2.9: Bibliotecas importadas DBSCAN

Afinar os Hiperparâmetros (`eps`, `min_samples`) do DBSCAN

O trecho de código ilustrado em 2.10 foi concebido para realizar uma pesquisa em grelha a fim de encontrar os parâmetros ótimos para o algoritmo de agrupamento DBSCAN. Os parâmetros a serem otimizados são `'eps'` e `'min_samples'`.

Aqui está uma análise detalhada do script:

- **Definição do Espaço de Parâmetros**: O script começa por definir o espaço de parâmetros para *eps* e *min_samples*. *eps* é a distância máxima entre dois pontos para que sejam considerados na mesma vizinhança. *min_samples* é o número mínimo de pontos numa vizinhança para que um ponto seja considerado como um ponto central. Os valores de *eps* são gerados utilizando `numpy.linspace` para criar 20 valores uniformemente espaçados entre 0.1 e 2.0. Os valores de *min_samples* são gerados usando `range` para criar uma lista de inteiros de 1 a 20.

- **Pesquisa em Grelha**: O script realiza então uma pesquisa em grelha sobre o espaço de parâmetros. Para cada combinação de *eps* e *min_samples*, ajusta o modelo DBSCAN aos dados dimensionados e prevê os rótulos

dos *clusters*. Se mais de um *cluster* for criado, calcula o índice de silhueta do agrupamento. O índice de silhueta é uma medida de quão semelhante um objeto é ao seu próprio *cluster* em comparação com outros *clusters*, sendo um índice mais elevado indicativo de um agrupamento melhor.

- **Seleção dos Parâmetros Ótimos**: Se o índice de silhueta para uma combinação específica de parâmetros for superior ao índice ótimo atual, o script atualiza os parâmetros e o índice ótimos.

- **Saída**: Por fim, o script imprime os valores ótimos de *eps*, *min_samples* e o índice de silhueta.

Esta pesquisa é uma maneira simples, mas eficaz, de ajustar os parâmetros do algoritmo DBSCAN para obter os melhores resultados de agrupamento de acordo com o índice de silhueta. No entanto, é importante notar que o índice de silhueta é apenas uma medida de qualidade de agrupamento, e os parâmetros ótimos podem variar dependendo das características específicas dos dados e dos objetivos da tarefa de agrupamento.

```
eps_space = np.linspace(0.1, 2.0, 20)  # generates 20 evenly spaced
    numbers from 0.1 to 2.0
min_samples_space = range(1, 21)  # generates numbers from 1 to 20

# Grid search
for eps in eps_space:
    for min_samples in min_samples_space:
        dbscan = DBSCAN(eps=eps, min_samples=min_samples)
        labels = dbscan.fit_predict(df_scaled)
        if len(set(labels)) > 1:  # more than 1 cluster created
            score = silhouette_score(df_scaled, labels)
            if score > optimal_score:
                optimal_eps = eps
                optimal_min_samples = min_samples
                optimal_score = score

print(f'Optimal eps: {optimal_eps}')
print(f'Optimal min_samples: {optimal_min_samples}')
print(f'Optimal silhouette score: {optimal_score}')
```

Excerto de Código 2.10: Pesquisa dos Valores Ótimos para os Hiperparametros do DBSCAN

Realização *Clustering* DBSCAN nos Dados

O DBSCAN é um algoritmo de agrupamento baseado em densidade. Ele agrupa pontos que estão próximos no espaço de características, o que significa que uma quantidade suficiente de pontos está próxima uns dos outros.

Funciona definindo um **raio (`eps`)** e um **número mínimo de pontos que devem existir nesse raio para formar um *cluster* (`min_samples`)**.

O algoritmo começa com um ponto aleatório no conjunto de dados. Se houver `min_samples` dentro de um raio `eps` desse ponto, um novo *cluster* é criado. O algoritmo então adiciona todos os pontos dentro do raio `eps` desse ponto (bem como os pontos dentro do `eps` desses pontos e assim por diante) ao *cluster*. Este processo é repetido até que não possam ser adicionados mais pontos ao *cluster*.

Em seguida, o algoritmo passa para o próximo ponto não visitado no conjunto de dados e repete o processo. Se um ponto visitado não tiver `min_samples` dentro do seu `eps`, esse ponto é considerado como ruído.

No trecho de código da figura 2.11, o DBSCAN é inicializado com `eps` igual ao valor optimal obtido em 2.10 e `min_samples` igual ao valor obtido correspondente em 2.10.

```
dbscan = DBSCAN(eps=optimal_eps, min_samples=optimal_min_samples).fit(
    df_scaled)
```

Excerto de Código 2.11: Inicialização do DBSCAN

Os rótulos de *cluster* resultantes são então adicionados aos dados originais, como ilustrado na figura 2.12:

```
df['cluster'] = dbscan.labels_
```

Excerto de Código 2.12: Rótulos do *cluster* adicionados aos dados originais

Estes (rótulos de *cluster*) são números inteiros que representam o *cluster* ao qual cada ponto pertence. Pontos rotulados como -1 são considerados ruído pelo algoritmo DBSCAN.

Pontuação de Silhueta (*Silhouette Score*) DBSCAN

Ilustrado na figura 2.13, o trecho de código utiliza a Pontuação de Silhueta cuja explicação já se encontra neste relatório em 2.2

O número de *clusters* é calculado como o número de rótulos únicos gerados pelo DBSCAN. Se o rótulo -1 estiver presente (o que DBSCAN usa para indicar ruído ou pontos que não se encaixam bem em nenhum *cluster*), ele é subtraído do total, pois não representa um *cluster* válido.

```
# Print silhouette score and number of clusters
print("Silhouette score for DBSCAN clustering:")
print(silhouette_avg)
print("Number of clusters (including noise):")
print(len(set(dbscan.labels_)) - (1 if -1 in dbscan.labels_ else 0))
```

Excerto de Código 2.13: *Silhouette Score* DBSCAN

Dados sobre os *Clusters* Formados pelo Algoritmo DBSCAN

O trecho de código ilustrado na figura 2.14, está a fornecer informações sobre os agrupamentos formados pelo algoritmo DBSCAN.

Inicialmente, são apresentados os tamanhos de cada *cluster*, indicando o número de pontos de dados atribuídos a cada agrupamento.

Em seguida, são apresentados os valores médios de cada variável para cada *cluster*. Estes valores médios são calculados agrupando os dados de acordo com o rótulo do *cluster* e determinando a média de cada grupo.

```
# Print the size of each cluster
print("Cluster sizes:")
print(df['cluster'].value_counts())

# Print the mean values of each variable for each cluster
print("Cluster centroids:")
print(df.groupby('cluster').mean())

# Visualize the clusters (only works if you have 2 or 3 variables)
if df_scaled.shape[1] <= 3:
    plt.scatter(df_scaled[:, 0], df_scaled[:, 1], c=dbscan.labels_)
    plt.show()
```

Excerto de Código 2.14: *Silhouette Score* DBSCAN

Implementação do Algoritmo t-SNE

O trecho de código ilustrado na figura 2.15 emprega a técnica t-SNE para reduzir a dimensionalidade dos dados para duas dimensões, facilitando assim a visualização de dados de alta dimensão.

Um gráfico de dispersão é produzido para visualizar os *clusters*. Cada *cluster* é colorido de forma distinta. O código itera sobre os *clusters* únicos e "plota"os pontos de dados pertencentes a cada *cluster* numa cor diferente. A legenda do gráfico é configurada para representar os rótulos únicos dos *clusters*.

Assim, este trecho de código é utilizado para visualizar os *clusters* num espaço bidimensional (`n_components=2`) usando a técnica t-SNE, sendo esta

uma abordagem útil para compreender como o algoritmo DBSCAN agrupou os dados.

```
# Perform t-SNE
tsne = TSNE(n_components=2, random_state=0)
df_tsne = tsne.fit_transform(df_scaled)

# Create a DataFrame with the t-SNE data and the cluster labels
df_tsne = pd.DataFrame(data=df_tsne, columns=['Component 1', 'Component
    2'])
df_tsne['Cluster'] = dbscan.labels_

# Visualize the clusters
plt.figure(figsize=(8,8))
colors = ['r', 'g', 'b']
for cluster, color in zip(df['cluster'].unique(), colors):
    indicesToKeep = df_tsne['Cluster'] == cluster
    plt.scatter(df_tsne.loc[indicesToKeep, 'Component 1']
               , df_tsne.loc[indicesToKeep, 'Component 2']
               , c = color
               , s = 50)
plt.legend(df['cluster'].unique())
plt.grid()
```

Excerto de Código 2.15: Implementação do Algoritmo *T-SNE*

2.4 *Script* SOM

Nesta secção será exposto e explicada as partes de maior relevância do *script* que permite fazer o *clustering* com o SOM a partir do *dataset* do problema. De referir que as partes comuns (leitura, tratamento dos dados, descoberta do número optimal `'k'` através do *elbow method*) deste *script* ao *script K-Means* não serão referidas nesta secção porque já foram referenciadas em 2.2.

Bibliotecas Necessárias

No trecho de código ilustrado em 2.16, são importadas as seguintes bibliotecas:

- `minisom`: Para Mapas Auto-Organizáveis (SOM).
- `numpy`: Para operações numéricas.
- `StandardScaler`: Do `sklearn` para padronizar características.

- `KMeans`: Do `sklearn` para *clustering K-Means*.
- `pandas`: Para manipulação e análise de dados.
- `matplotlib.pyplot`: Para visualização de dados.
- `silhouette_score`: Do `sklearn` para avaliar a qualidade de *clusters*.
- `euclidean_distances`: Do `sklearn` para calcular distâncias entre pontos de dados.
- `bone, pcolor, colorbar, plot, show`: Para dar suporte na visualização de métricas da avaliação do *clustering*.

```
from minisom import MiniSom
import numpy as np
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.metrics import silhouette_score
from sklearn.metrics.pairwise import euclidean_distances
from matplotlib.pylab import bone, pcolor, colorbar, plot, show
```

Excerto de Código 2.16: Bibliotecas necessárias SOM

Inicialização e Treino

Breve noção do que é como funciona o SOM:

É um tipo de Rede Neural Artificial treinada através de aprendizagem não supervisionado para produzir uma representação discretizada de baixa dimensão (tipicamente bidimensional) do espaço de entrada das amostras de treino, chamada de mapa. Este mapa preserva a topologia, o que significa que as posições dos nós no mapa refletem as relações inerentes nos dados de entrada.

- **Inicialização:** Os pesos dos neurónios são inicializados com pequenos valores aleatórios.
- **Competição:** Para cada vetor de entrada, o neurónio com os pesos mais próximos do vetor de entrada é identificado como a Unidade de Correspondência Melhor (*Best Matching Unit* (BMU)).
- **Cooperação:** O raio da vizinhança da BMU é calculado. Este raio determina o número de neurónios que também serão atualizados quando os pesos forem ajustados.

- **Adaptação:** Os pesos dos neurónios dentro da vizinhança da BMU são ajustados para torná-los mais semelhantes ao vetor de entrada. Os pesos dos neurónios são puxados para mais perto do vetor de entrada, com neurónios mais próximos da BMU sendo movidos mais.

- **Continuação:** Os passos 2 a 4 são repetidos por um certo número de iterações. Com o tempo, o mapa começará a tomar forma, com áreas do mapa sendo associadas a determinados vetores de entrada.

O resultado é um mapa onde entradas similares são mapeadas para áreas próximas. Os SOMs são úteis para visualização e análise exploratória de dados. No trecho de código da figura 2.17, o SOM é inicializado com um tamanho e número de características especificados. Em seguida, treina o SOM utilizando os dados padronizados.

Neste trecho de código, ilustrado em 2.17, `som = MiniSom(x_size, y_size, input_len, sigma=1.0, learning_rate=0.5)` cria um novo SOM. Os parâmetros são:

- **x_size e y_size:** Definem as dimensões da grelha na qual o SOM será treinado. A escolha de 10x10 é arbitrária e pode ser ajustada com base nos requisitos específicos do conjunto de dados e do problema.

- **input_len:** Este é o número de características no conjunto de dados. É derivado dos próprios dados (`data_scaled.shape[1]`).

- **sigma:** Este é o espalhamento da função de vizinhança e determina quantos neurónios serão atualizados quando um neurónio específico é atualizado. A escolha de 1.0 é um ponto de partida comum, mas pode ser ajustada com base nos requisitos específicos do problema.

- **learning_rate:** Controla a quantidade pela qual os pesos são atualizados durante cada iteração. A escolha de 0.5 é um equilíbrio entre fazer grandes atualizações (que podem ultrapassar a solução ótima) e fazer atualizações pequenas (que podem ser muito lentas).

Em `som.random_weights_init(data_scaled)` inicializam-se os pesos do SOM aleatoriamente, usando os dados normalizados.

Em `som.train_random(data_scaled, 100)` treina-se o SOM usando os dados normalizados por 100 iterações. A escolha de 100 iterações é arbitrária e pode ser ajustada com base nos requisitos específicos do problema. O treino é feito de forma aleatória, o que significa que, para cada iteração, um ponto

de dados aleatório é selecionado e o SOM é atualizado com base nesse ponto de dados.

```
# Initialize and train the SOM
print("Initializing and training the SOM...")
x_size = 10  # size of the SOM
y_size = 10
input_len = data_scaled.shape[1]  # number of features in the dataset
print('Number of Features:', input_len)
som = MiniSom(x_size, y_size, input_len, sigma=1.0, learning_rate=0.5)
som.random_weights_init(data_scaled)
som.train_random(data_scaled, 100)  # number of iterations
```

Excerto de Código 2.17: SOM inicialização e treino

Obtenção das Coordenadas Vencedoras

A parte do código, ilustrada em 2.18, obtém as coordenadas vencedoras para cada ponto de dados nos dados normalizados. As coordenadas vencedoras referem-se à posição do neurónio no SOM que tem os pesos mais próximos de um dado ponto de dados.

A linha de código `winning_coordinates = np.array([som.winner(x) for x in data_scaled]).astype(float)` chama o método `winner` do objeto `som` para cada ponto de dados em `data_scaled`. O método `winner` retorna as coordenadas (x, y) do neurónio vencedor para o dado ponto de dados. O resultado é uma lista de pares de coordenadas, que é então convertida para um *array* e o tipo de dados é definido como *float*. O *array* resultante `winning_coordinates` contém as coordenadas vencedoras para cada ponto de dados nos dados normalizados.

```
# Initialize and train the SOM
print("Initializing and training the SOM...")
x_size = 10  # size of the SOM
y_size = 10
input_len = data_scaled.shape[1]  # number of features in the dataset
print('Number of Features:', input_len)
som = MiniSom(x_size, y_size, input_len, sigma=1.0, learning_rate=0.5)
som.random_weights_init(data_scaled)
som.train_random(data_scaled, 100)  # number of iterations
```

Excerto de Código 2.18: Obtenção das coordenas vencedoras

Determinação do Número Optimal `'k'` recorrendo ao *Elbow Method*

Esta secção tem o objetivo de determinar o número optimal `'k'` recorrendo ao *Elbow Method*. Esta parte do código já foi referenciada em 2.2, sendo retirado que `'k'` tem como valor 4.

K-Means Clustering nas Coordenadas Vencedoras

Esta parte do código, ilustrada em 2.19, realiza *clustering K-Means* nas coordenadas vencedoras obtidas do SOM e, em seguida, atribui os rótulos de *cluster* resultantes aos dados originais.

- `kmeans = KMeans(n_clusters=optimal_k, n_init=10)`: Esta linha inicializa um modelo de *clustering K-Means* com o número ótimo de *clusters* (`optimal_k`) determinado anteriormente. O parâmetro `n_init` está definido como 10, o que significa que a *clustering K-Means* será executada 10 vezes com diferentes sementes de centróide, e o modelo final será aquele com melhor desempenho.

- `clusters = kmeans.fit_predict(winning_coordinates)`: Esta linha ajusta o modelo K-Means às coordenadas vencedoras do SOM e prevê o cluster para cada ponto de dados. O array resultante `clusters` contém as atribuições de cluster para cada ponto de dados.

- `df['cluster'] = clusters`: Esta linha adiciona as atribuições de cluster ao dataframe original `df` como uma nova coluna chamada 'cluster'. Isso permite uma visualização fácil e análise adicional dos resultados da *clustering* em relação aos dados originais.

O objetivo deste código é agrupar pontos de dados semelhantes com base nas coordenadas vencedoras do SOM.

```
# Perform KMeans clustering on the winning coordinates with the optimal
    number of clusters
kmeans = KMeans(n_clusters=optimal_k, n_init=10)
clusters = kmeans.fit_predict(winning_coordinates)

# Now, 'clusters' contains the cluster assignments for each data point
df['cluster'] = clusters
```

Excerto de Código 2.19: *K-Means clustering* nas coordenas vencedoras

Métricas de Avaliação da *clustering* e do Desempenho do SOM

Cálculo do *Silhouette Score* para o Número Optimal

Neste trecho de código, ilustrado em 2.20 é avaliado a qualidade da *clustering* para `'k'`.

```
# Calculate Silhouette Score
score = silhouette_score(winning_coordinates, clusters)

print("Silhouette Score: ", score)
```

Excerto de Código 2.20: Cálculo do *silhouette score* para o número optimal

Visualização do SOM

Neste trecho de código, ilustrado em 2.21 é criado uma visualização 2D do mapa SOM, onde a cor de cada célula representa a distância média para os vizinhos dessa célula, e os marcadores representam os diferentes clusters.

```
bone()
pcolor(som.distance_map().T)
colorbar()
markers = ['o', 's', 'D', 'v', '^', '<', '>', '8', 'p']
colors = ['r', 'g', 'b', 'c', 'm', 'y', 'k', '#eeefff', '#15b01a']
for i, x in enumerate(data_scaled):
    w = som.winner(x)
    plot(w[0] + 0.5,
         w[1] + 0.5,
         markers[clusters[i]],
         markeredgecolor = colors[clusters[i]],
         markerfacecolor = 'None',
         markersize = 10,
         markeredgewidth = 2)
show()
```

Excerto de Código 2.21: Visualização SOM

Cálculo dos Erros de Quantização e Topográfico

Neste trecho de código, ilustrado em 2.22 é calculado:

- O erro de quantização que mede o erro médio entre cada dado e o neurónio vencedor.
- O erro topográfico que mede a proporção de todos os neurónios para os quais o segundo melhor neurónio vencedor não é um vizinho direto do primeiro melhor.

Valores menores para ambos indicam uma melhor qualidade de *clustering*.

```
# Calculate Quantization and Topographic errors
q_error = som.quantization_error(data_scaled)
t_error = som.topographic_error(data_scaled)
```

Excerto de Código 2.22: Cálculo dos erros de quantização e topográfico

2.5 Conclusão

Nesta secção foram apresentadas as partes de maior relevo no que toca aos *scripts* correspondentes a cada técnica de *clustering*. Foi também aproveitado para introduzir conceitos teóricos relativos a cada técnica e feita uma explicação e exposição das métricas utilizadas para cada técnica de *clustering*.

Capítulo 3

Exposição e Análise de Resultados

3.1 Introdução

O foco principal deste capitulo é fornecer *insights* claros sobre o desempenho de cada técnica de *clustering*, por meio de métricas que ajudam a perceber a eficácia do agrupamento proporcionado pelas mesmas.

3.2 Resultados do *K-Means*

Para esta técnica de *clustering*, foram utilizadas as seguintes métricas (já detalhadas em 2.2):

- Método do Cotovelo (*Elbow Method*)
- Pontuação de Silhueta

Método do Cotovelo (*Elbow Method*)

Nesta secção apresenta-se o gráfico deste método e o número ótimo (`'k'`) de *clusters* determinado pelo mesmo.

Na figura 3.1 está ilustrado o gráfico do cotovelo onde o eixo x é o número de *clusters* e o eixo y é a inércia. À medida que o número de *clusters* aumenta, a inércia diminui, pois os pontos de dados estão mais próximos dos seus respetivos centróides.

No entanto, após um certo número de *clusters*, a diminuição da inércia torna-se muito menos acentuada. Este ponto é conhecido como o "cotovelo"e é geralmente considerado um bom indicador do número ótimo de *clusters*. Em outras palavras, adicionar mais *clusters* além desse ponto não resultará numa melhoria significativa da inércia.

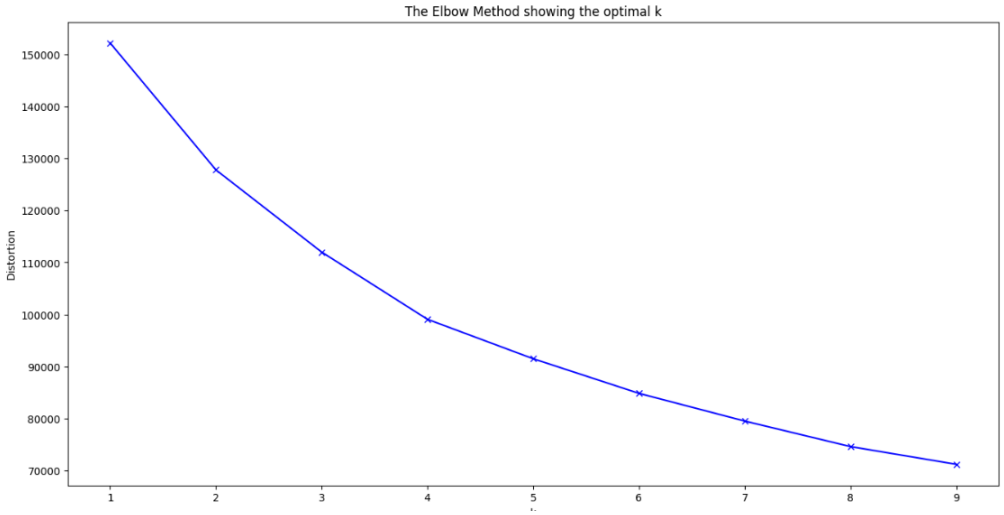

Figura 3.1: Gráfico do Cotovelo (*Elbow*)

Através do trecho de código ilustrado em 2.6, conseguiu-se obter que o número ótimo (`'k'`) de *clusters* é **3**.

Observando o gráfico, é de notar que o ponto o número ótimo (`'k'`) está entre 3 e 5, sensivelmente.

Pontuação de Silhueta (*Silhouette Score*)

Nesta secção apresenta-se o número ótimo (`'k'`) de *clusters* determinado pela Pontuação de Silhueta (através do trecho de código em 2.8).

Os resultados indicam que o número ótimo de *clusters*, determinado pelo método da pontuação de silhueta, é **3**. Isso significa que, de acordo com este método, 3 é o número de *clusters* que maximiza a pontuação de silhueta.

A pontuação de silhueta para este agrupamento é de aproximadamente `Silhouette Score: 0.24978243426308558`.

Embora o método da pontuação de silhueta sugira que 3 é o número ótimo de *clusters*, a pontuação de silhueta sugere que a qualidade do agrupamento pode não ser ideal.

3.3 Resultados do DBSCAN

Valores dos Hiperparâmetros Obtidos e Pontuação de Silhueta

Os valores obtidos no trecho de código 2.10 para o `eps` e para o `min_samples` foram, respetivamente, **2** e **6**. Isto significa que, para um ponto de dados ser

considerado parte de um *cluster*, deve haver pelo menos 6 outros pontos de dados dentro de uma distância de 2.0 (no espaço de características escalado).

A pontuação de silhueta neste método de *clustering* para estes hiperparâmetros foi de **0.480**, que é razoavelmente bom, o que sugere que os pontos de dados encaixam razoavelmente bem nos seus respetivos *clusters*.

Dados Obtidos sobre os Clusters Formados

Os valores obtidos no trecho de código 2.14 são espelhados nas seguintes tabelas.

Cluster	BALANCE	BAL_FREQ	PURCHASES	ONEOFF_PUR
-1	4312.14	0.92	4550.00	2951.88
0	1355.71	0.87	733.72	413.17

Tabela 3.1: Centróides dos Clusters (Parte 1)

Cluster	INSTALL_PUR	CASH_ADV	PUR_FREQ	ONEOFF_FREQ	PUR_INST_FREQ
-1	1599.10	3543.45	0.70	0.45	0.54
0	320.80	784.02	0.47	0.18	0.35

Tabela 3.2: Centróides dos Clusters (Parte 2)

Cluster	CASH_ADV_FREQ	CASH_ADV_TRX	PUR_TRX	CREDIT_LIMIT	PAYMENTS
-1	0.30	11.21	51.36	9340.06	7291.97
0	0.12	2.64	11.93	4126.22	1310.79

Tabela 3.3: Centróides dos Clusters (Parte 3)

Cluster	MIN_PAYMENTS	PRC_FULL_PAY	TENURE
-1	3439.72	0.20	11.21
0	669.62	0.15	11.54

Tabela 3.4: Centróides dos Clusters (Parte 4)

Daqui retira-se que:

- **Tamanhos dos Clusters:** O algoritmo DBSCAN identificou dois *clusters*. O *Cluster* 0 contém 8318 pontos de dados, e o *cluster* -1 contém 632 pontos de dados. No DBSCAN, o *cluster* -1 representa o "ruído"– esses são pontos de dados que não se encaixaram bem em nenhum *cluster*.

- **Centróides dos Clusters:** Os centróides dos *clusters* representam os valores médios das características para cada *cluster*. Por exemplo, a média do `BALANCE` para o *cluster* -1 é aproximadamente 4312.14, enquanto para o *cluster* 0 é aproximadamente 1355.71. Isso sugere que os clientes no *cluster* -1 tendem a ter um saldo mais elevado do que aqueles no *cluster* 0. Da mesma forma, é possível comparar os centróides de outras características para entender como os *clusters* diferem.

A interpretação destes resultados depende do contexto específico e do significado dos seus dados. Por exemplo, como o `BALANCE` representa o montante de dinheiro na conta de um cliente, então o *cluster* -1 representa clientes "mais abastados", enquanto o *cluster* 0 representa clientes "menos abastados".

Resultado do t-SNE

O gráfico obtido a partir do trecho de código ilustrado na figura 2.15 é ilustrado em 3.2.

O gráfico t-SNE é uma representação visual dos dados de alta dimensão reduzidos para duas dimensões. Cada ponto no gráfico representa uma única amostra do conjunto de dados. O seu objetivo é posicionar os pontos no espaço 2D de forma que pontos próximos no espaço de alta dimensão também estejam próximos no espaço 2D, e pontos distantes no espaço de alta dimensão também estejam distantes no espaço 2D.

Analisando o gráfico da figura 3.2:

- Mostra 2 grupos distintos de pontos, que são potenciais *clusters* dos dados. Sendo o primeiro assinalado a vermelho, 0, e o segundo a verde, -1.

- Existe a presença de alguns *outliers*, pontos que estão longe de qualquer *cluster*. Neste caso, estes podem corresponder a pontos de "ruído"que o DBSCAN atribuiu ao *cluster* -1.

- Existe uma ligeira sobreposição de *clusters*, o que indica que estes *clusters* não estão bem separados no espaço de alta dimensão.

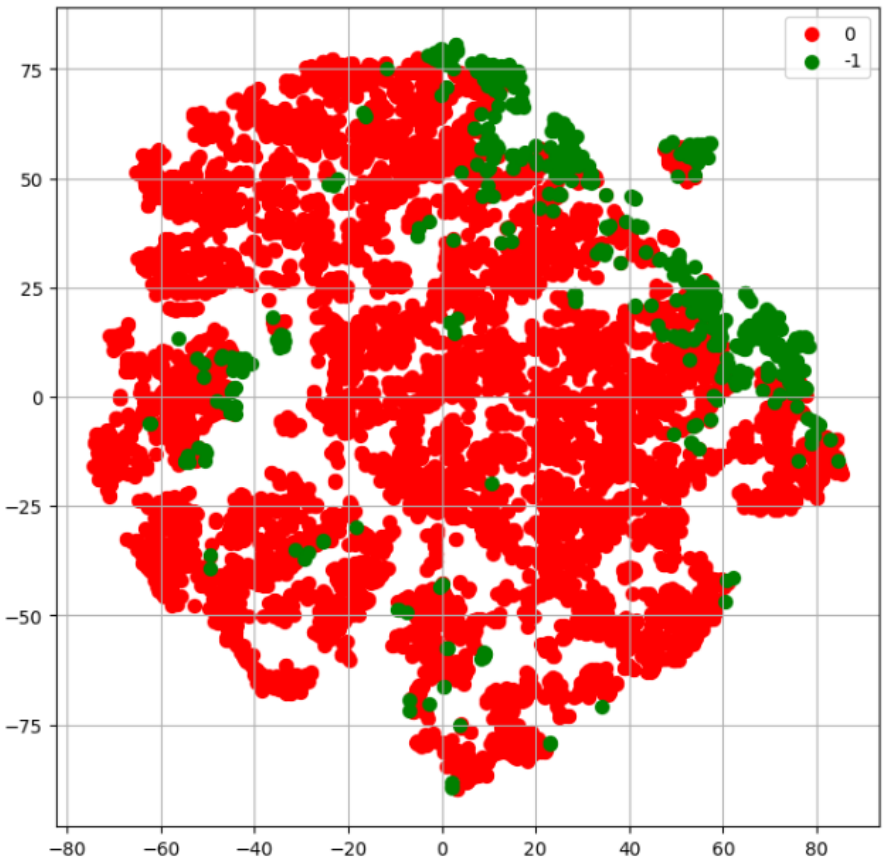

Figura 3.2: t-SNE DBSCAN

3.4 Resultados do SOM

Pontuação de Silhueta

Os resultados do trecho de código 2.8 foram que a pontuação de silhueta, para um `'k'` optimal de 3, foi:

- `Silhouette Score:  0.44651002217895697`

Visualização do SOM

A visualização 2D do mapa SOM, através do trecho de código 2.21 está ilustrada na figura 3.3.

- **Distâncias:** As cores no mapa representam a distância entre cada neurónio e os seus vizinhos. As cores mais escuras representam distâncias maiores. Os *clusters* aparecem como regiões de cores semelhantes cercadas por bordas de cores mais escuras.
- **Marcadores:** Os marcadores representam os diferentes *clusters* nos dados. Cada tipo de marcador corresponde a um *clusters* diferente. A posição de um marcador no mapa indica o neurónio vencedor para um dado de entrada, e, portanto, a posição desse dado no espaço de entrada.

No mapa é possível ver que existem regiões onde marcadores semelhantes estão agrupados juntos o que indica que os dados de entrada correspondentes são semelhantes entre si. Na disposição dos marcadores é possível ver que os marcadores estão agrupados em regiões distintas o que reflete a estrutura dos dados de entrada.

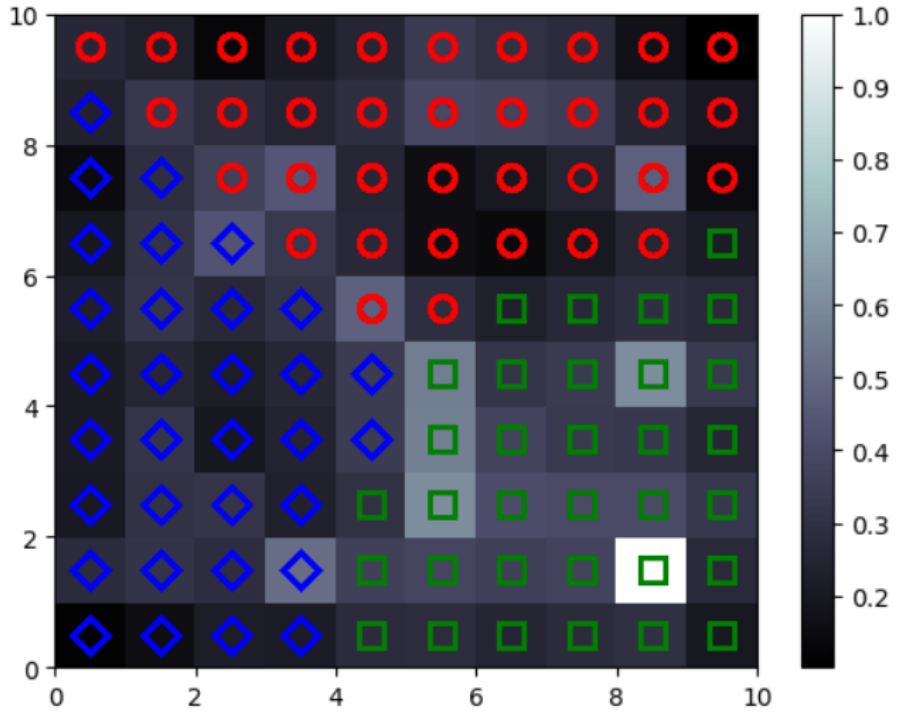

Figura 3.3: Mapa 2D SOM

Erros de Quantização e Topográfico

Os resultados do trecho de código, ilustrado em 2.22 foram:

- `Quantization Error: 1.8185325520268483` Indica a média das distâncias euclidianas entre cada dado de entrada e o neurónio vencedor correspondente no mapa SOM. Valores menores de erro de quantização indicam que o SOM está a fazer uma boa aproximação dos dados de entrada. Portanto, um valor de 1.82 é relativamente alto, sugerindo que este pode não estar a representar os dados de entrada de maneira muito precisa.

- `Topographic Error: 0.8367597765363128` Indica a proporção de todos os neurónios para os quais o segundo melhor neurónio vencedor não é um vizinho direto do primeiro melhor. Valores menores de erro topográfico indicam que o SOM está a preservar bem as relações topológicas dos dados de entrada. Portanto, um valor de 0.84 é bastante alto, sugerindo que o SOM pode não estar a preservar as relações topológicas dos dados de entrada de maneira eficaz.

Estes valores sugerem que a qualidade do SOM pode ser melhorada. Para isso será necessário ajustar parâmetros como:

1. Taxa de aprendizagem
2. Tamanho do mapa
3. Raio do vizinho

3.5 Conclusão

Nesta secção foram apresentados os resultados das métricas correspondentes a cada técnica de *clustering*. Sendo que a técnica que obteve uma maior pontuação de silhueta foi o DBSCAN, e a que obteve pior foi o *K-Means*.

Capítulo 4

Conclusão

4.1 Conclusões Principais

Ao analisar os resultados obtidos a partir dos algoritmos de *clustering K-Means*, DBSCAN e SOM pode-se concluir que:

Na métrica de *Silhouette Score*, o método DBSCAN foi o mais eficiente, com uma pontuação de aproximadamente 0.480 (`eps` = 2, `min_samples` = 6), enquanto o *K-Means* foi o menos eficiente, com uma pontuação de cerca de 0.2498 (`'k'` = 3). Isso sugere que o **DBSCAN foi capaz de produzir agrupamentos mais coesos e bem definidos do que o *K-Means***.

Além disso, a análise dos resultados do SOM mostraram que esta técnica pode servir para visualizar dados de alta dimensionalidade num espaço bidimensional, permitindo uma melhor compreensão da estrutura dos dados.

Em geral, podemos concluir que a escolha do algoritmo de *clustering* depende muito do conjunto de dados e dos objetivos específicos do projeto. No entanto, com base nos resultados obtidos neste trabalho, podemos afirmar que o DBSCAN é uma boa opção para a maioria dos casos, especialmente quando a métrica de *Silhouette Score* é usada como critério de avaliação.

4.2 Próximos Passos

Os próximos passos deste projeto poderão ser:

- Explorar outras métricas de avaliação de *clustering*, como a métrica de *Calinski-Harabasz* ou a métrica de *Davies-Bouldin*, para comparar os resultados obtidos pelos algoritmos de *clustering*.

- Investigar outras técnicas de *clustering*, como o *Hierarchical Clustering*, para avaliar a sua eficácia neste conjunto de dados e comparar com as técnicas já abordadas neste relatório.