Relatório Trabalho Pratico 2

Ivan Gomes da Cruz[1], **Gabriel Arcanjo Campelo Fadoul**[2]

[1]Aluno de Matemática Computacional – Universidade Federal de Minas Gerais (UFMG)
Belo Horizonte – MG – Brazil

[2]Aluno de Ciência da Computação – Universidade Federal de Minas Gerais (UFMG)
Belo Horizonte – MG – Brazil

ivangomes.trabalho@gmail.com, gabrielarcanjo642@gmail.com

Abstract. *This report explores the implementation of approximation algorithms for the k-center problem, which is essential in clustering techniques within machine learning. We focus on comparing a 2-approximation algorithm with the traditional K-Means algorithm, analyzing both the efficiency and the quality of the clustering solutions offered. The algorithms were tested on synthetic datasets generated through the Scikit-learn library and data generated through the multivariate normal distribution, and performance was evaluated using metrics such as average time, Rand index, and silhouette score. The results highlight the advantages and disadvantages of each algorithm in terms of accuracy and computational speed.*

Resumo. *Este relatório explora a implementação de algoritmos aproximativos para o problema dos k-centros, essencial nas técnicas de agrupamento em aprendizado de máquina. Focamos na comparação entre um algoritmo 2-aproximado e o tradicional K-Means, analisando tanto a eficiência quanto a qualidade das soluções de agrupamento oferecidas. Os algoritmos foram testados em conjuntos de dados sintéticos gerados através da biblioteca Scikit-learn e dados gerados através da distribuição normal multivariada, e o desempenho foi avaliado com métricas como tempo médio, índice de Rand e pontuação de silhueta. Os resultados destacam as vantagens e desvantagens de cada algoritmo em termos de precisão e velocidade computacional.*

1. Introdução

Trabalho realizado para a disciplina de Algoritmos 2, no primeiro semestre de 2024

1.1. Objetivos

O objetivo deste trabalho é explorar e implementar algoritmos aproximativos para o problema dos k-centros, que é fundamental para técnicas de agrupamento em aprendizado de máquinas. Vamos focar em implementar o algoritmo 2-aproximado e compará-lo com o clássico K-Means, analisando tanto a eficiência quanto a qualidade das soluções que cada um oferece.

1.2. O Problema dos K-centros

O problema dos k-centros busca escolher k centros em um conjunto de pontos para que a maior distância entre qualquer ponto e o seu centro mais próximo seja minimizada.

É um problema importante em *clustering*, onde queremos encontrar uma boa solução que agrupe os dados de forma eficiente. Esse problema é fundamental para diversas aplicações, como análise de dados e compressão de imagens, mas encontrar a solução exata pode ser muito complexo e demorado, especialmente para grandes volumes de dados.

Para lidar com essa complexidade, muitas vezes recorremos a algoritmos aproximativos. Esses algoritmos não garantem a solução perfeita, mas oferecem boas soluções de forma muito mais rápida. Entre esses algoritmos, os 2-aproximados são populares por garantir que a solução encontrada não seja mais do que o dobro da melhor solução possível.

Neste trabalho, vamos implementar e comparar dois desses algoritmos 2-aproximados com o algoritmo K-Means, que é uma abordagem clássica para o agrupamento. Vamos avaliar a qualidade das soluções e o tempo necessário para processar os dados, utilizando tanto conjuntos de dados reais do UCI Machine Learning Repository quanto conjuntos sintéticos gerados por métodos específicos.

2. Metodologias

2.1. Algoritmos Implementados

2.1.1. 2-aproximativo por Refinamento de Intervalo

Para resolver o problema dos k-centros com uma abordagem de 2-aproximação baseada em refinamento de intervalo, implementamos a função `refining_two_approximation`. Esta função ajusta iterativamente o intervalo onde o raio ótimo pode estar, buscando encontrar uma solução eficiente para o problema.

A função começa determinando a dimensão dos pontos no conjunto de dados S e calcula o valor máximo do raio (r_{max}) entre todos os pares de pontos, estabelecendo o intervalo inicial de 0 a r_{max}. Inicializamos dois limites, `low` e `high`, que representam o intervalo atual de possíveis valores para o raio ótimo.

Em cada iteração, calculamos o ponto médio do intervalo (`mid`) e aplicamos o algoritmo de aproximação para verificar se é possível encontrar uma solução com esse valor de raio. Dependendo do resultado, ajustamos o intervalo: se uma solução é encontrada, reduzimos o limite superior (`high`); se não, aumentamos o limite inferior (`low`). Registramos os valores de `high` e `low` em listas para monitorar a evolução do intervalo ao longo das iterações.

Ao final, a função retorna a melhor solução encontrada, junto com os intervalos refinados.

2.1.2. 2-aproximativo utilizando a distância maxima (K_centers)

Para resolver o problema dos k-centros com essa abordagem, utilizamos três funções principais: `distance_matrix`, `max_dist_point` e `k_centers_max_dist`. Cada uma delas desempenha um papel crucial na seleção dos centros de agrupamento de forma eficiente.

A função `distance_matrix` é responsável por calcular a matriz de distâncias entre todos os pares de pontos em um conjunto de dados. Ela começa inicializando uma matriz $n \times n$ de zeros, onde n é o número total de pontos. Usando dois loops aninhados, a função calcula a distância entre todos os pares de pontos (i, j) usando a métrica de Minkowski, com o parâmetro p que define o tipo de distância (Manhattan para $p = 1$ e Euclidiana para $p = 2$). A distância calculada é atribuída às posições correspondentes da matriz, garantindo que a matriz seja simétrica. Após o cálculo, a função retorna a matriz de distâncias completa, essencial para os algoritmos subsequentes.

A função `max_dist_point` é utilizada para encontrar o ponto que possui a maior distância mínima em relação a um conjunto de centros já selecionados. Inicialmente, a função define `max_dist` como -1 e `max_index` como `None`. Para cada ponto que ainda não foi escolhido como centro, calcula-se a menor distância desse ponto para qualquer um dos centros já selecionados. A função então compara essa distância mínima com a maior distância mínima encontrada até o momento. Se a distância mínima atual for maior, `max_dist` e `max_index` são atualizados. Ao final da iteração, a função retorna o índice do ponto que possui a maior distância mínima, ajudando a garantir a melhor separação entre centros.

A função `k_centers_max_dist` seleciona k centros a partir de um conjunto de pontos, maximizando a distância entre os centros escolhidos. Se o número de centros k for maior ou igual ao número de pontos n, todos os pontos são retornados como centros. Caso contrário, a função começa selecionando aleatoriamente um ponto como o primeiro centro. Em seguida, enquanto o número de centros selecionados for menor que k, o próximo centro é escolhido com base na função `max_dist_point`, que determina o ponto com a maior distância mínima em relação aos centros existentes. Esse processo continua até que todos os k centros sejam selecionados. A função retorna os pontos correspondentes aos centros escolhidos, promovendo uma boa dispersão dos centros no espaço de dados.

2.2. Algoritmo classico para comparação

2.2.1. K-means

O algoritmo K-Means é amplamente utilizado em clustering e análise de dados. Ele divide um conjunto de dados em k grupos, onde cada grupo é representado por um centroide, que é a média dos pontos dentro do grupo. O processo começa com k centroides iniciais, escolhidos aleatoriamente. Em seguida, cada ponto é atribuído ao centroide mais próximo. Após essa atribuição, os centroides são recalculados como a média dos pontos atribuídos a cada grupo. Esse ciclo de atribuição e recalculo é repetido até que os centroides se estabilizem e as atribuições dos pontos não mudem mais.

2.3. Datasets

Para testar e comparar os algoritmos de clustering, criamos dois conjuntos de dados sintéticos usando abordagens diferentes. Cada uma foi pensada para simular cenários variados de distribuição e sobreposição entre grupos, permitindo uma análise mais completa dos resultados.

2.3.1. Dados do Scikit-learn

A primeira abordagem foi baseada nos exemplos fornecidos pela biblioteca Scikit-learn, Esses exemplos geram diferentes conjuntos de dados sintéticos que simulam diversas situações, como:

- **Blobs**: Pontos distribuídos em torno de centros bem definidos.
- **Moons**: Dois conjuntos de dados em forma de lua crescente, entrelaçados.
- **Circles**: Dados em formato circular, concentrados em anéis concêntricos.

2.3.2. Dados Usando Distribuição Normal Multivariada

Para a segunda abordagem, criamos dados sintéticos em duas dimensões usando uma distribuição normal multivariada. Desenvolvemos uma função específica que nos permitiu gerar grupos de pontos ao redor de centros definidos, com a possibilidade de ajustar o quanto esses grupos se sobrepõem.

O processo funciona da seguinte forma:

1. **Escolha dos Centros**: Primeiro, selecionamos alguns centros aleatórios que servem como os pontos principais em torno dos quais os clusters serão formados.
2. **Ajuste da Dispersão**: Para cada centro, definimos um desvio padrão que controla o espalhamento dos pontos ao redor do centro. Um desvio padrão maior faz com que os clusters fiquem mais espalhados e possivelmente se sobreponham, enquanto um desvio menor resulta em grupos mais compactos.
3. **Geração dos Pontos**: Usando a distribuição normal multivariada, geramos os pontos ao redor desses centros.

Utilizando esta função criamos varios cenários diferentes tanto com clusters bem separados quanto com alta sobreposição.

2.4. Métricas de avaliação

Para este trabalho todas as métricas foram calculadas como a média após 30 execuções dos algoritmos

2.4.1. Tempo Médio (‘average_time‘)

O tempo médio é a média do tempo que um algoritmo leva para processar os dados. Usado para comparar a eficiência (quanto a velocidade) dos algoritmos.

2.4.2. Pontuação Rand Média (‘avg_rand_score‘)

A Pontuação Rand mede a qualidade do agrupamento ao comparar a clusterização obtida pelo algoritmo com uma clusterização de referência conhecida.

2.4.3. Pontuação de Silhueta Média (‘avg_silhouette_score‘)

A Pontuação de Silhueta avalia a qualidade dos clusters mostrando o quão bem cada ponto está agrupado em seu próprio cluster em comparação com outros clusters.

2.4.4.

3. Resultados

3.1. Dados do Scikit-learn

3.1.1. Visualização dos dados gerados

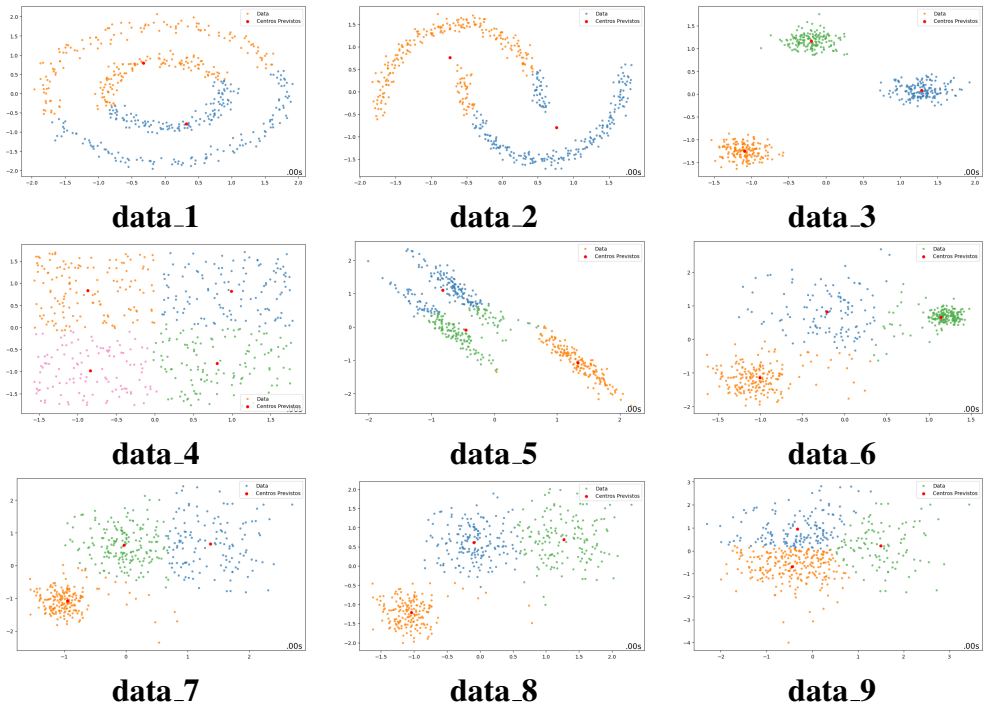

Figura 1. Imagens geradas usando a abordagem do Scikit-learn

3.1.2. Tabela de Métricas

	Métrica	data_1	data_2	data_3	data_4	data_5	data_6	data_7	data_8	data_9
K_centers	average_time	0,005	0,003	0,009	0,016	0,01	0,01	0,011	0,009	0,008
	avg_rand_score	5,036	6,817	9,998	0	6,667	7,837	6,959	8,169	4,474
	avg_silhouette_score	3,27	4,199	8,437	3,609	3,809	5,544	3,736	4,583	2,809
Kmeans	average_time	0	0	0	0	0	0	0	0	0
	avg_rand_score	4,99	7,269	2.1900	0	8,115	8,828	8,635	9,464	6,046
	avg_silhouette_score	3,5	4,962	8,439	4,131	5,032	6,394	4,999	5,548	3,479

Figura 2. Tabela Dados Scikit-learn

3.2. Dados Usando Distribuição Normal Multivariada

3.2.1. Visualização dos dados gerados

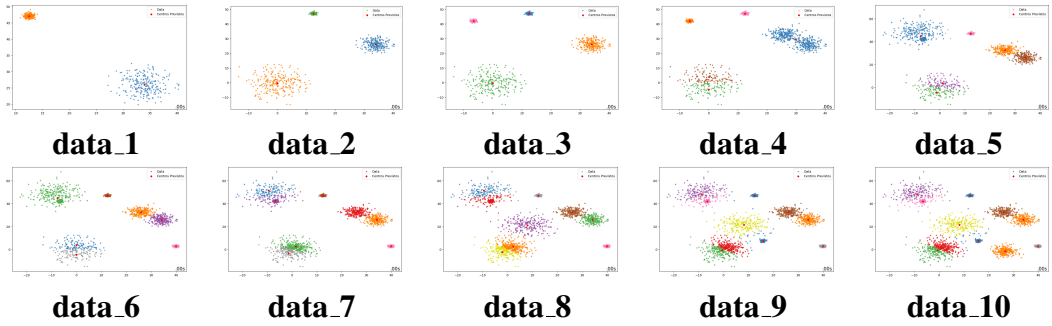

Figura 3. Imagens geradas sinteticamente

3.2.2. Tabela de Métricas

	Métrica	data_1	data_2	data_3	data_4	data_5	data_6	data_7	data_8	data_9	data_10
K_centers	average_time	0,023	0,017	0,023	0,031	0,042	0,045	0,047	0,054	0,049	0
	avg_rand_score	2.1900	9,907	2.1900	9,829	9,628	9,688	9,732	9,71	9,737	9,82
	avg_silhouette_score	9,13	8,602	8,86	7,658	6,616	7,009	6,432	6,232	6,384	6,311
Kmeans	average_time	0,003	0,015	0,034	0,062	0,106	0,165	0,237	0,345	0,712	1,029
	avg_rand_score	2.1900	2.1900	8,757	8,884	8,874	9,183	9,279	9,151	9,291	9,234
	avg_silhouette_score	9,13	8,72	6,946	6,375	5,638	5,946	5,44	4,618	5,015	4,136

Figura 4. Tabela DadosUsando Distribuição Normal Multivariada

4. Conclusão

4.1. Dados do Scikit

Ao analisarmos o formato dos agrupamentos formados pelos algoritmos do KMeans e do KCenters, é possível notar que há um resultado similar, inclusive ambos tendo dificuldade em agrupar corretamente dados que não são linearmente separáveis, como os conjuntos de dados `data_1` e `data_2`. Ao analisarmos as métricas das distâncias intra-clusters(silhueta) e o Rand, é possivel notar que há uma similiraridade no resultados entre os dois algoritmos em quase todas as bases, sendo a exceção mais notável os resultados da base de dados `data_3`, em que a métrica `avg_rand_score` entre o dois algoritmos são bem discrepantes.

4.2. Dados da Distribuição Normal Multivariada

Nestes conjuntos de dados foi possível já perceber uma diferença notável entre os algoritmo testados. Como o algoritmo aproximativo KCenters sempre dá preferência a pontos que maximizem a distância, seus centros sempre acabam tendendo às extremidades dos agrupamentos, um tanto distante das centroides de fato de onde os pontos foram gerados. Já o KMeans têm uma performance excelente, colocando suas centroides geralmente sobrepondo diretamente as originais. Nas métrica já é possivel notar que há ainda uma similaridade entre os dois algoritmos, porém o tempo de execução do KMeans parece

piorar ao aumentar a quantidade de centros e consequentemente de dados, enquanto o algoritmo aproximativo se mantém com um bom tempo de execução mesmo aumentando a quantidade de dados. É possível notar que há um *trade-off* na escolha entre os dois algoritmos, o KMeans parece ser mais preciso na escolha dos centros porém sua execução parece ficar mais lenta ao aumentar a quantidade de instâncias do conjunto de dados Enquanto o KCenters se mantêm estável ao aumentar a quantidade de dados, ele não têm a mesma precisão pelo método como os centros são escolhidos.

5. References

Referências

Micreiros (2024). Clusterização de dados: K-means na biblioteca scikit-learn. Acesso em: 13 agosto 2024.

Overflow, S. (2024). When using scikit-learn k-means clustering, how can you extract the centroids in python? Acesso em: 13 agosto 2024.

Scikit-learn (2024a). Cluster comparison example. Acesso em: 12 agosto 2024.

Scikit-learn (2024b). sklearn.metrics.rand_score. Acesso em: 12 agosto 2024.

Scikit-learn (2024c). sklearn.metrics.silhouette_score. Acesso em: 12 agosto 2024.