Trabalho Prático II

Antônio C. N. Neto[1], **Raphael Aroldo C. Mendes**[1]

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

{antonioneto, raphaelmendes}@dcc.ufmg.br

1. Introdução

O presente relatório retrata a implementação de algoritmos aproximativos, para um problema clássico de aprendizado de máquina, o *k-centros*, bem como ilustra como diversificações acerca de especificações e eficácia se comportam em variados conjuntos de dados, para cada método.

O intuito é compreender como algoritmos aproximativos funcionam em conjuntos reais e simulados, alguns verdadeiramente desafiadores para os mesmos. Em sequência, como fins comparativos, também será executado para os mesmos conjuntos de dados o algoritmo clássico para o problema de agrupamentos, o K-Means.

As comparações tem como objetivo avaliar, tanto aspectos de demanda computacional, quanto da qualidade da solução por determinadas métricas, além de incluir análises de relação de fatores, como dimensionalidade, aumento do conjunto e característica do dado com a capacidade de agrupar os mesmos em *k-centros*.

2. Descrição do Problema

Contextualizando, o problema dos *k-centros* envolve encontrar centros os quais visam minimizar o raio máximo de cada cluster (denomina-se cluster como um centróide, isto é, o ponto central e todos os pontos que estejam mais próximos desse centro, do que dos outros).

Portanto, para cada ponto, a distância mínima de um cluster deve ser a menor possível (1), e a métrica de qualidade da solução é a maior distância de um ponto ao seu centro mais próximo (2), que define o raio da solução.

$$\text{dist}(S_i, C) = \min_j \{\text{dist}(S_i, C_j)\} \tag{1}$$

$$r(C) = \max_{i \in S} \{\text{dist}(S_i, C)\} \tag{2}$$

Logo, a solução do problema é um conjunto de centros C que minimize a equação 2. Outro fator importante também é que a cardinalidade desse conjunto é previamente definida, ou seja, $|C| = k$, sendo k dado como parâmetro. Em contextos práticos, o problema é muito comum, pois existem uma variedade de situações nas quais *k-centros* é aplicável.

- Separar clientes de uma empresa em grupos, entendendo cada um separadamente;

- Em redes de comunicação, como redes de telecomunicações ou redes de computadores, k centrais (hubs) precisam ser posicionadas para minimizar o tempo de transmissão ou a latência;
- Governos ou autoridades locais precisam decidir onde posicionar k estações de serviços de emergência (como bombeiros, ambulâncias, ou polícia) para que o tempo de resposta em emergências seja minimizado.

Portanto, seu estudo é extremamente importante, principalmente devido ao fato de não haver solução determinística conhecida que seja rápida de execução, sendo caracterizado como um problema NP.

3. Algoritmos

Os algoritmos utilizados no presente relatório buscam resolver problemas computacionalmente complexos, admitindo soluções aproximadas, isto é, distantes da solução ótima por um fator c, com $c = 2$, nesse caso. Será apresentado duas variações de aproximativos e um heurístico comumente utilizado pela comunidade de aprendizado de máquina.

3.1. K-centros

A presente seção tem como objetivo descrever a implementação das duas versões existentes para o algoritmo aproximativo do *k-centros*. Em ambas as versões do algoritmo, foi utilizado de forma auxiliar a matriz de distâncias, construída para cada uma das distâncias utilizadas, de modo a acessar de forma fácil as distâncias entre pontos p_i, p_j.

3.2. Aproximativo por busca intervalar

A primeira versão do k-centros implementada está descrita no Algoritmo (1).

A partir desse algoritmo, existe um teorema que afirma que para qualquer solução ótima C^* com até k centros, o raio ótimo é maior que o raio definido para o algoritmo aproximativo, $r(C^*) > r$.

Utilizando desse teorema, é possível utilizar o algoritmo anterior como indicativo se o raio r atual deve ser aumentado ou reduzido. Com isso, é possível refinar o intervalo que contém r, partindo de $(0, max(dist))$, até obter dois intervalos, sendo o limite esquerdo r_l e o limite direito r_r, formando (r_l, r_r). Observe a implementação da ideia acima no Algoritmo (2).

3.3. Aproximativo por maximização de distâncias

O segundo algoritmo aproximativo para o problema é um algoritmo guloso, que busca de forma iterativa sempre reduzir o raio atual. Para compreender, observe a definição do raio atual definida na Equação (2). Portanto, para reduzir o raio, basta fazer com que o ponto que define a maior $dist(S_i, C)$ se torne também um centro. Segue o algoritmo no Algoritmo (3).

3.4. K-means

O K-means é um dos algoritmos de clustering mais populares devido à sua simplicidade e eficiência. Como um algoritmo heurístico, é necessário determinadas parametrizações, sendo a principal delas a condição de convergência do algoritmo. A seguir, o pseudocódigo para o algoritmo do K-means.

Algorithm 1: K-centros Base

Function $k - centros1(S, k, r, dist)$**:**
- $S' \leftarrow S$
- $C \leftarrow \emptyset$
- **while** $S' \neq \emptyset$ **do**
  - $p^r \leftarrow random(s \in S')$ `/* Random point select. */`
  - $remove\ p^r\ from\ S'$
  - **forall** $p' \in S'$ **do**
    - **if** $dist(p', p^r) \leq 2r$ **then**
      - $remove\ p'\ from\ S'$
    - **end**
  - **end**
  - $C \leftarrow p^r$
- **end**
- **if** $C \leq k$ **then**
  - **return** C
- **else**
  - **return** $\emptyset$
- **end**

end

Algorithm 2: Refinamento de Intervalos

Function $RefinedInterval(S, k, dist, w^p)$**:**
- $M \leftarrow max(dist)$
- $r_l \leftarrow 0$ `/* Left limit */`
- $r_r \leftarrow M$ `/* Right limit */`
- $C \leftarrow k - centros1(S, k, \frac{r_l + r_r}{2}, dist)$
- **while** $((r_r - r_l) > Mw^p)\ or\ (C = \emptyset)$ **do**
  - **if** $C \neq \emptyset$ **then**
    - $r_r \leftarrow \frac{r_l + r_r}{2}$
  - **else**
    - $r_l \leftarrow \frac{r_l + r_r}{2}$
  - **end**
  - $C \leftarrow k - centros1(S, k, \frac{r_l + r_r}{2}, dist)$
- **end**
- **return** $(r_l + r_r), C$

end

1. **Inicialização:** Escolha k e selecione k centróides iniciais aleatoriamente.
2. **Atribuição:** Atribua cada ponto ao centróide mais próximo.
3. **Recalcular Centroides:** Atualize cada centróide como a média dos pontos do cluster.
4. **Iteração:** Repita os passos de atribuição e recalculação até convergir.
5. **Finalização:** Retorne os clusters finais e seus centróides.

Algorithm 3: *Greedy* K-centros

Function $GreedyKcenters(S, k, dist)$**:**
- $S' \leftarrow S$
- **if** $k > |S'|$ **then**
  - **return** S `/* All points are centers. */`
- **else**
  - $p^r \leftarrow random(s \in S')$ `/* Random point select. */`
  - $remove\ p^r\ from\ S'$
  - $C \leftarrow p^r$
- **end**
- **while** $|C| < k$ **do**
  - $D_{min} \leftarrow \emptyset$
  - **forall** $p \in S'$ **do**
    - $D_{min} \leftarrow \min(dist(p, C), p)$
  - **end**
  - $p \leftarrow (\max(D_{min})).p$
  - $remove\ p\ from\ S'$
  - $C \leftarrow p$
- **end**
- **return** C

end

Nota-se que o principal aspecto do K-means é os centróides estarem na média do grupo os quais pertence, com atualização constante até certa convergência parametrizada.

4. Geração dos dados

Para fins de experimentação e testes dos algoritmos, é necessário dados de diversas distribuições. Nessa conjuntura, a cumprir com as determinações do trabalho, foi consultado dados de problemas reais, provenientes do *UCI Machine Learning Repository*, juntamente com a geração de dados sintéticos, a partir das formulações descritas nas subseções 4.2 e 4.3.

Com o intuito de uniformidade dos dados, foi determinado que o número de *samples* por conjunto será de 2000 ao máximo, isto é, 2000 pontos. Nos casos de geração simulada sempre será de 2000 instâncias de pontos e aos dados reais, no mínimo 700.

4.1. UCI Machine Learning Repository

O UCI Machine Learning Repository é uma coleção de conjuntos de dados para aprendizado de máquina, criada pela Universidade da Califórnia, Irvine. Lançado em 1987, oferece dados para experimentar e avaliar algoritmos de aprendizado de máquina. Ele cobre várias áreas, como reconhecimento de padrões e análise de séries temporais. É amplamente usado na pesquisa e desenvolvimento para testar e comparar métodos. O repositório é gratuito e atualizado regularmente.

Para o uso do mesmo nas experimentações, a Tabela 1 descreve quais as bases de dados foram utilizadas, nas quais a partir do identificador único é possível identificar

no site do repositório. A escolha basesou em utilizar de conjuntos com poucas variáveis categóricas de formato de *strings* como espaço gerador das feature, dado que isso limita o problema e os algoritmos não serão eficientes. Logo, qualquer coluna que seja categórica em *strings* será desconsiderada no modelo. Colunas inteiras ainda pertencem para visualizar seus impactos no desempenho dos algoritmos. Diferentes dimensionalidades foram escolhidas para também entender sua relação com desempenho.

Tabela 1. Bases de dado do repositório UCI escolhidas

ID	N° Instâncias	Dimensão	Número de classes
545	2000	7	2
850	900	7	2
697	2000	36	3
350	2000	23	2
94	2000	57	2
80	2000	64	10
544	2000	16	7
159	2000	10	2
468	2000	17	2
59	2000	16	26

4.2. Toy Datasets

Para a geração do primeiro conjunto de dados sintéticos, utilizou-se de uma biblioteca em python que gera pontos $p_i = (x_i, y_i)$, configurados e distribuido em formatos pré-determinados ao longo do $\mathbb{R}^2$, bem como controlar qual a classificação de cada um desses pontos gerados.

Com isso, gerou-se 9 dados com as configurações acima, algumas com aplicação de matrizes de transformação e outras não, além de um exemplo separado uma distribuição aleatória de pontos, sem clusters, todos pertecentes a mesma classe, totalizando os 10 conjuntos para esse tópico.

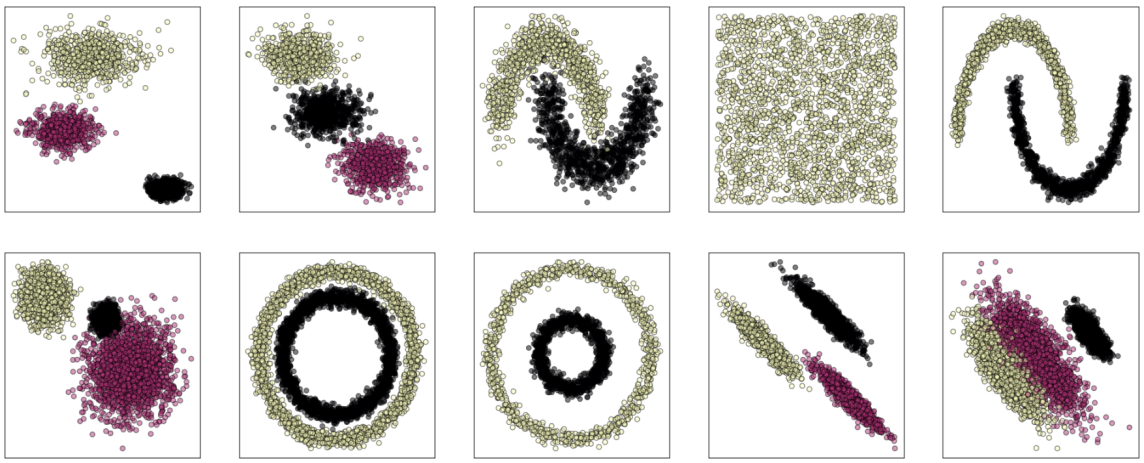

Figura 1. 10 Dados Sintéticos gerados com ToyDataset.

4.3. Distribuição Normal Multivariada

Prosseguindo na geração dos dados sintéticos, as últimas 10 amostras vem de uma distribuição normal multivariada em $\mathbb{R}^2$. Para essa geração de dados, existem alguns parâmetros que são produzidos aleatóriamente. Apesar disso, as 10 amostras utilizadas são iguais para os 3 algoritmos, apenas sua geração que é aleatória.

Para cada uma das amostras, inicialmente é gerado um número aleatório inteiro entre $(2, 6)$, que define a quantidade de clusters para cada uma das 10 amostras. A partir disso, para cada cluster de uma amostra, é gerado uma média aleatória em $(5, 15)$ para amostras ímpares e uma média aleatória $(20, 50)$ para amostras pares. A razão disso é que os clusters são gerados em torno de suas respectivas médias na distribuição, e a escolha acima faz com que as amostras ímpares tenham alta probabilidade de sobreposição de clusters, enquanto as amostras pares tenham baixa probabilidade de sobreposição de clusters.

Para definir a disposição dos pontos de um cluster, é gerado uma matriz aleatória 2x2 para cada cluster e aplicado um truque de álgebra linear, fazendo com que a matriz de covariância para seja $A^T A$ e sempre seja uma matriz de covariância válida. Para evitar que a matriz seja sempre positiva, as entradas $\forall_{i \neq j}\ A_{i,j}$, que indicam a covariância entre o eixo x e o eixo y, são multiplicadas por -1 aleatoriamente.

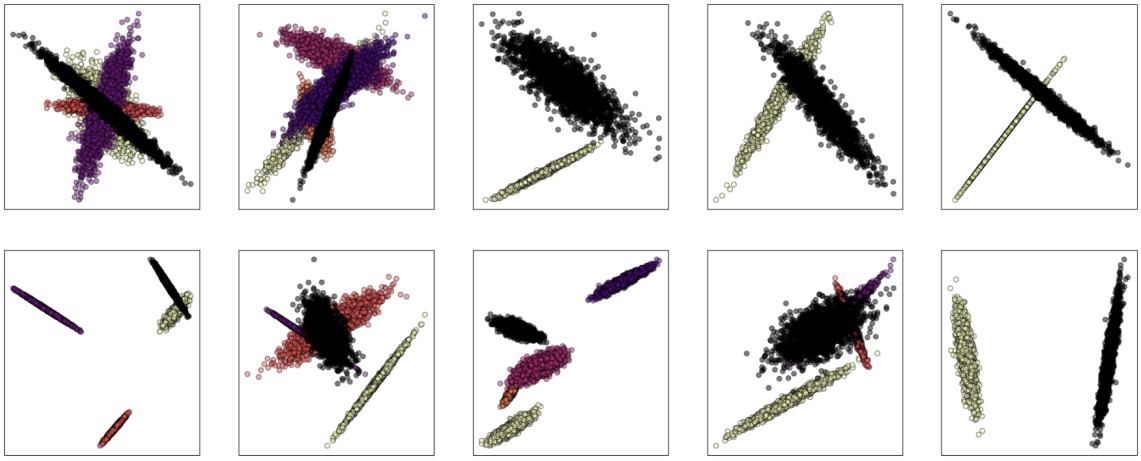

Figura 2. 10 Dados Sintéticos gerados com ToyDataset.

5. Experimentação

Neste tópico, será abordado a experimentação realizada com os algoritmos e os conjuntos de dados apresentados, descrevendo a metodologia e as métricas utilizadas.

5.1. Metodologia

Como decisão de projeto, cada parametrização possível foi executada no mínimo 5 vezes para conseguir ter uma amostra devidamente razoável para relevar possíveis variações randômicas intrísecas.

De tal forma, como cada algoritmo tem uma configuração diferente, cada um teve suas variações específicas. Importante destacar que para os algoritmos aproximativos, foi utilizado de duas distâncias, a de *Manhattan* (3) e a *Euclidiana* (4).

$$d_{Manhattan}(x, y) = \sum_{i=1}^{n} |x_i - y_i| \tag{3}$$

$$d_{Euclidean}(x, y) = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2} \tag{4}$$

Como variável de atribuição global, assume-se que o número de testes $N = 15$. Como o algoritmo de busca por refinamento possui 10 variações de parâmetros, pois são 5 intervalos possíveis, sendo escolhido um refinamento pelo conjunto 0.01, 0.05, 0.1, 0.2, 0.25 e também duas matrizes de distância, então para cada uma é executado 5 vezes, ou seja, $N/3$.

Para o algoritmo de maximização de distâncias, existem duas possibilidades somente, então cada configuração foi executada N vezes. Para o K-Means, como um algoritmo consolidado e que pela configuração padrão utilizada, executá-lo diversas vezes não impacta no resultado, como será visualizado no tópico 6.

Em resumo, cada dataset será executado 85 vezes.

5.2. Métricas

As métricas se baseiam em comparar o custo computacional em função do tempo para cada configuração e também em relação a otimalidade das soluções encontradas, isto é, qualidade do resultado aproximado obtido.

Para o cálculo do tempo, é determinado será somente da função que encontra o raio, sem considerar outras eventuais computações, como atribuição de labels, visando manter a uniformidade.

Para as métricas de qualidade da solução, foram feitas as duas indicadas nas especificações do trabalho prático: o *Silhouette Score* e *Adjusted Rand Index (ARI)*.

O *Silhouette Score* mede o quão bem cada ponto de um cluster está agrupado em relação aos outros clusters, com valores variando de -1 a 1. Um valor alto indica que os pontos estão bem agrupados dentro do cluster e bem separados dos outros clusters. Um valor próximo de 0 sugere que os pontos estão na fronteira entre dois clusters.

Já o *Adjusted Rand Index (ARI)* é uma métrica usada para avaliar a similaridade entre duas partições de dados. Ele ajusta o Rand Index para levar em conta a chance de agrupamentos aleatórios, com valores que variam de -1 a 1. Um ARI de 1 indica que as partições são idênticas, enquanto um valor próximo de 0 sugere que a similaridade é equivalente ao acaso. Valores negativos indicam pior do que o esperado por acaso. O ARI é útil para comparar a qualidade de diferentes agrupamentos ou validar resultados de clustering. Fato a ser considerado é que para diferentes configurações de label para os centróides, o *ARI* é o mesmo.

6. Resultados

Nesta seção, será visualizado os resultados feitos pela experimentação em forma tabular para algumas configurações, no conjunto de dados reais da UCI. Logo após, serão efetuadas análises específicas com visualizações gráficas nos dados sintéticos.

Tabela 2. Desempenho dos algoritmos no primeiro dataset UCI

Alg/Distância	Intervalo	Tempo	Silhouette	ARI	Raio
C1/Euclidian	**0.01**	0.879 ± 0.58	0.383 ± 0.0	0.0 ± 0.0	1.577 ± 0.13
	0.05	0.66 ± 0.45	0.383 ± 0.01	0.0 ± 0.0	1.654 ± 0.05
	0.10	0.478 ± 0.07	0.382 ± 0.01	0.0 ± 0.0	1.712 ± 0.12
	0.20	0.412 ± 0.09	-0.17 ± 0.94	0.0 ± 0.0	1.748 ± 0.33
	0.25	0.373 ± 0.14	0.382 ± 0.01	0.0 ± 0.0	1.734 ± 0.0
C1/Manhattan	**0.01**	1.149 ± 0.91	0.375 ± 0.01	0.0 ± 0.0	1.964 ± 0.21
	0.05	0.592 ± 0.12	0.379 ± 0.01	0.0 ± 0.0	1.995 ± 0.23
	0.10	0.764 ± 0.28	0.376 ± 0.01	0.001 ± 0.0	2.095 ± 0.18
	0.20	0.371 ± 0.06	0.372 ± 0.01	0.0 ± 0.0	2.086 ± 0.13
	0.25	0.263 ± 0.05	0.1 ± 0.76	-0.0 ± 0.0	2.44 ± 0.22
C2/Euclidian	**-**	0.011 ± 0.0	0.379 ± 0.0	0.001 ± 0.0	1.549 ± 0.06
C2/Manhattan	**-**	0.012 ± 0.0	0.377 ± 0.0	0.001 ± 0.0	1.88 ± 0.1
M/Euclidian	-	0.008 ± 0.0	0.388 ± 0.0	-0.0 ± 0.0	1.67 ± 0.02

A Tabela 2 mostra os resultados das métricas para cada configuração possível no primeiro dataset do *UCI*. Como intervalo de confiança, foi considerado 95%. Nota-se que para esse caso específico, o ARI foi 0 para praticamente todas as configurações, indicando um conjunto de difícil partição.

Além disso, nota-se como o raio vai piorando conforme o intervalo de convergência no refinamento é aumentado. Para a tabela, entende-se que o algoritmo C1 é o de refinamento, o algoritmo C2 é o de maximização e o M é o K-Means. No final da experimentação total, foram 2550 instâncias de configurações analisadas.

6.1. Análises

O conjunto de experimentos é vasto e possui diversas análises cabíveis de serem realizadas. Um exemplo é como o raio é impactado conforme o intervalo de busca do refinamento é aumentado.

Selecionando somente a distância euclidiana para a geração da visualização, a Figura 3 demonstra tal evolução, sendo uma normalização com os raios encontrados divididos pelo raio do parâmetro 0.01, utilizando somente um conjunto de cada estilo de dataset. Nota-se uma evolução do raio conforme o parâmetro de convergência aumenta.

6.1.1. Silhueta e ARI

Além disso, é possível verificar em relação aos índices o desempenho de referência, que é o K-Means, como os algoritmos aproximativos desempenham nos diferentes datasets.

A Figura 4 mostra a diferença para o algoritmo de maximização, onde os datasets enumerados de 0 a 9 pertencem à UCI, de 10 a 19 pertencem ao Toy e de 20 a 29 pertencem à distribuição normal multivariada. A distância padrão utilizada é a euclidiana. É possível notar que o K-Means possui desempenho superior em ambos os índices, mas para o silhueta, no conjunto Toy, o algoritmo 2 é superior com frequência.

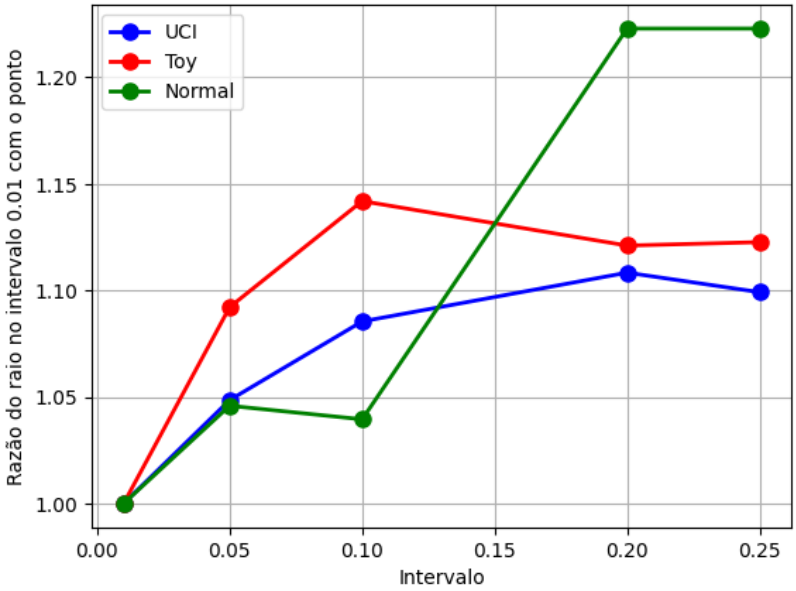

Figura 3. Desempenho do algoritmo de refinamento conforme o intervalo de convergência aumenta.

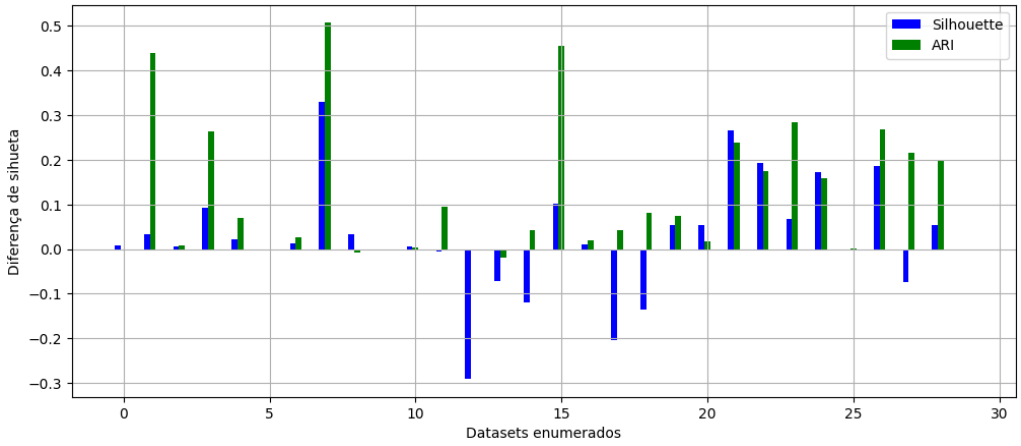

Figura 4. Diferença entre o desempenho do K-Means para o algoritmo 2 nos índices silhueta e ARI.

Já a Figura 5 mostra a diferença para o algoritmo de refinamento. O intervalo padrão foi 0.01. Induz que o K-Means tem desempenho inferior a ambos algoritmos aproximativos na maior parte dos conjuntos de dados Toy em relação à silhueta. Além disso, uma observação importante é seu desempenho inferior em relação ao algoritmo 2, pois sua diferença em relação ao K-Means é nitidamente maior.

6.1.2. Tempo

Para o tempo, a Figura 6 possui o desempenho de cada algoritmo para 3 conjuntos de cada tipo de dataset. Analisando por um momento, é possivel observar como o algoritmo de refinamento demora mais que os outros, com o intervalo de convergência definido como 0.01, e métrica de distância euclideana. Além disso, para a distribuição

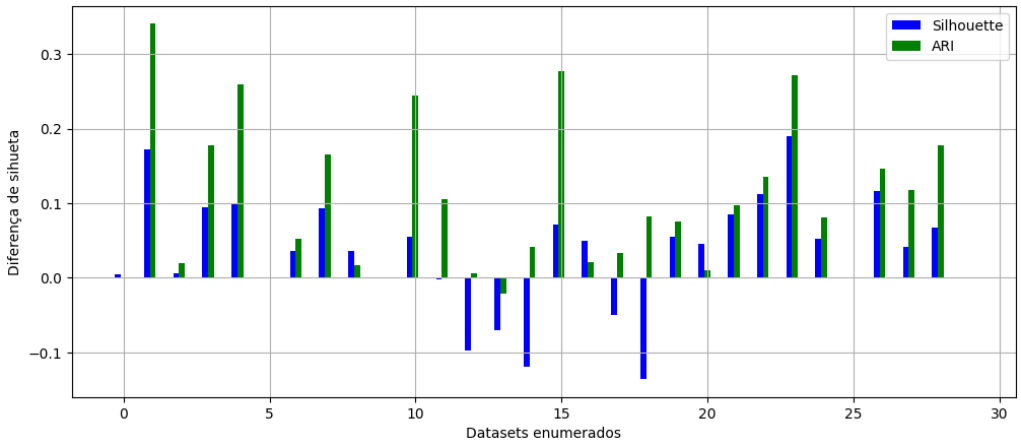

Figura 5. Diferença entre o desempenho do K-Means para o algoritmo 1 no índice de silhueta e ARI.

normal multivariada o tempo de refinamento é muito maior que os demais conjuntos.

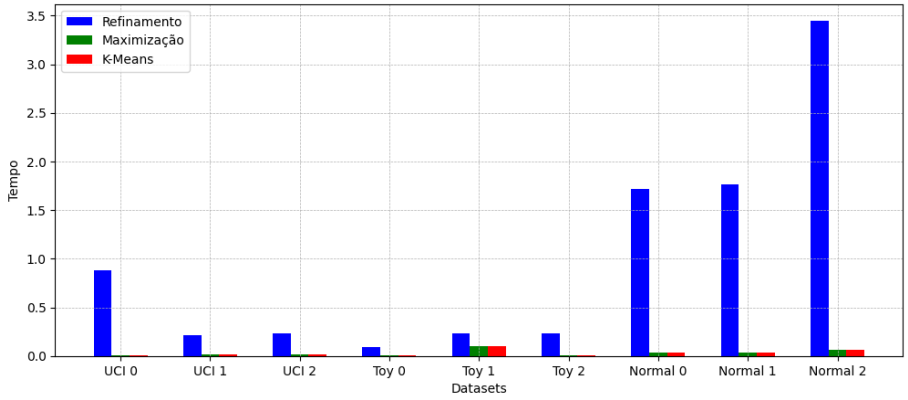

Figura 6. Desempenho de tempo dos algoritmos em diferentes datasets

6.1.3. Distância

Para fins comparativos, a Tabela 3 mostra o desempenho do algoritmo de refinamento com intervalo 0.01 para encontrar o raio para as duas distâncias utilizadas: *Manhattan* e *Euclidiana*. Observe que cada configuração foi testada no mínimo 5 vezes e o valor das tabelas são a média.

Por sua vez, Tabela 4 mostra o desempenho do algoritmo guloso. Observe que em ambas as tabelas o raio encontrado a partir da distância de *Manhattan* é maior que o da *Euclidiana*.

6.2. Exemplos

Para visualização do funcionamento dos diferentes algoritmos, as Figuras 7 e 8 ilustram os pontos pertences aos diferentes centróides encontrados por cada estratégia, visualizados pelas cores, para o conjunto do Toy Datasets e Distribuição Normal Multivariada, respectivamente.

Tabela 3. Comparativo de raio encontrado pelo algoritmo de refinamento.

Conjunto	Manhattan	Euclidiana
UCI 0	1.964	1.577
UCI 1	1.440	1.060
Toy 0	8543.29	5903.92
Toy 1	252.59	49.503
Normal 0	31.853	24.534
Normal 1	77.697	61.298

Tabela 4. Comparativo de raio encontrado pelo algoritmo de maximização.

Conjunto	Manhattan	Euclidiana
UCI 0	1.880	1.549
UCI 1	1.457	1.065
Toy 0	9196.19	6495.36
Toy 1	256.20	50.66
Normal 0	34.027	27.094
Normal 1	106.94	80.932

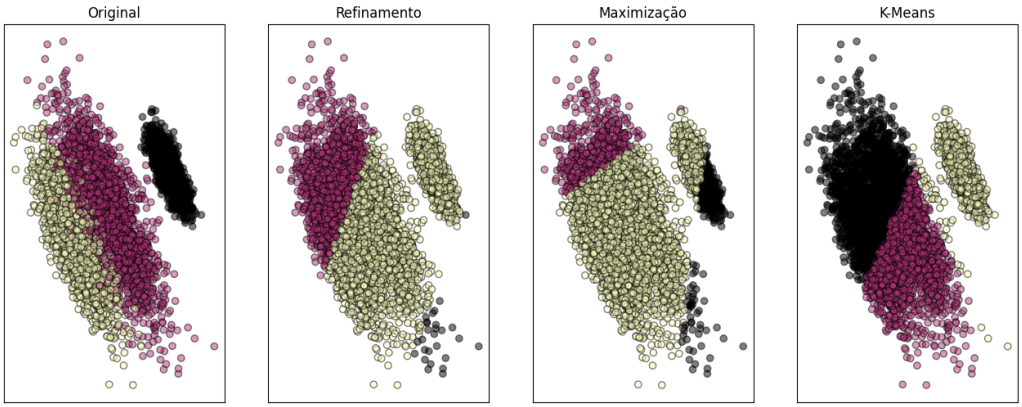

Figura 7. Classificação dos pontos, por algoritmo, no Toy Datasets.

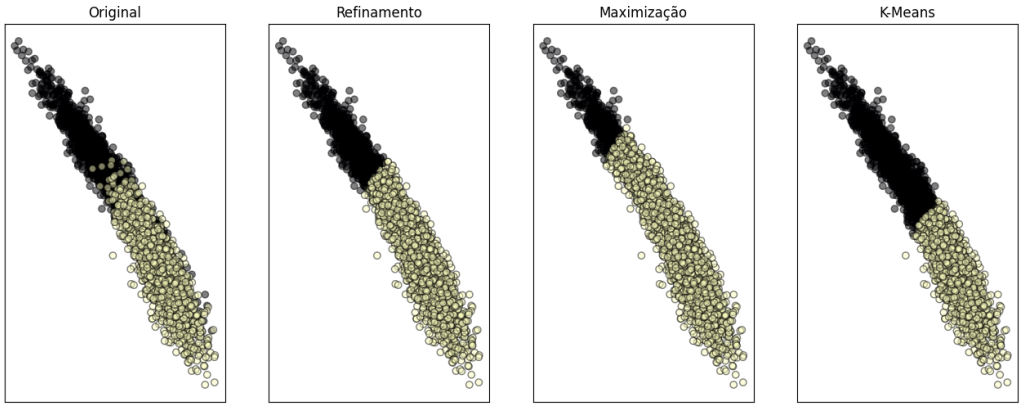

Figura 8. Classificação dos pontos, por algoritmo, na Distribuição.

7. Conclusão

No presente relatório, analisou-se o desempenho dos dois algoritmos aproximativos com diferentes estratégias e um heurístico em um problema clássico de machine learning, que é o agrupamento de dados.

As lógicas dos três algoritmos são relativamente simples, partindo de um viés guloso de construção de algoritmos, o que possibilita um baixo custo computacional para um problema difícil. Note que para alguns casos o algoritmo de refinamento pode ser ruim, devido à dificuldade de encontrar um possível raio ótimo, além da necessidade de um intervalo de convergência baixo para obtenção raios precisos.

Ademais, em relação aos índices, dificilmente os algoritmos aproximativos superam o K-Means, notado somente em conjuntos Toy. As distâncias também tem impacto no desempenho do algoritmo, visto que encontram diferentes raios, além da diferença notável nos conjuntos de pontos que pertencem aos seus respectivos centróides.

Em última análise, o trabalho possibilitou a compreensão do funcionamento dos algoritmos nas bases de dados requisitadas, e com o desenvolvimento a partindo de um viés científico e observatório, tornou os resultados confiáveis e comparáveis, sendo um estudo satisfatório com base na proposta inicial.

Referências

UCI Machine Learning Repository. Disponível em: https://archive.ics.uci.edu/datasets. Acesso em: 15 ago. 2024.

Scikit-learn: Machine Learning in Python. K-means clustering. Disponível em: https://scikit-learn.org/stable/modules/clustering.htmlk-means. Acesso em: 15 ago. 2024.