

Greedy K-Means: uma abordagem aproximativa para o problema de clusterização

Lucas Paulo Martins Mariz

¹Universidade Federal de Minas Gerais

²Departamento de Ciência da Computação – UFMG

lucaspaulom@hotmail.com

Abstract. *This article describes the challenges encountered in an approximate approach to the K-means clustering algorithm. For this, several data sets were used that, with the appropriate metrics and treatments applied, were able to answer several interesting questions. In addition, topics such as efficiency, complexity and limitations will be extensively explored in all possible areas.*

Resumo. *Esse artigo descreve os desafios encontrados em uma abordagem aproximativa do algoritmo de clusterização K-means. Para tal, foram utilizados diversos conjuntos de dados que, com as devidas métricas e tratamentos aplicados, foram capazes de responder diversos questionamentos interessantes. Além disso, temas como eficiência, complexidade e limitações serão amplamente aprofundados em todos os âmbitos possíveis.*

1. Introdução

O problema dos K-centros é amplamente conhecido na matemática e computação. Em sua abordagem mais geral, pode-se dizer que ele tem como principal função o agrupamento de dados em relação as características do conjunto como um todo. Em outras palavras, dado um conjunto de dados, o algoritmo definirá K pontos centrais (tais pontos representam os centros de cada cluster). Um ponto qualquer do conjunto de dados pertencerá ao cluster comandado pelo ponto central mais próximo a ele.

A definição do número K é um desafio na área de machine learning: diversos métodos são utilizados para ajustar esse parâmetro da melhor forma possível. No entanto, a proposta desse artigo leva em consideração grupos de dados previamente classificados, ou seja, o número de centros a serem encontrados é definido pela quantidade de classes do conjunto verificado. Dessa forma, a temática está concentrada em avaliar a qualidade dos clusters bem como analisar outras métricas que informam a peculiaridade do agrupamento.

Por fim, por se tratar de um problema NP, será abordado um algoritmo guloso 2-aproximativo, ou seja, que retornará um raio até 2x maior que o raio ótimo. Dessa forma, obteremos uma abordagem com complexidade $O(nK)$, onde n é o número de pontos analisados e K é o número de centros a serem definidos.

2. Procedimentos e Métodos

O algoritmo 2-aproximativo a ser implementado foi apresentado no decorrer da disciplina. Sua ideia consiste no seguinte: primeiramente é avaliado se o número de centros a serem buscados é maior que a quantidade de dados disponibilizados. Caso positivo,

retornamos o próprio conjunto e encerramos. Caso contrário, escolhemos um ponto, de maneira arbitrária, para ser o primeiro centro. Após isso, inicia-se um loop que busca os outros $K-1$ centros, de forma que um ponto será selecionado somente se ele maximiza a distância entre ele e os outros pontos centrais já definidos.

Dado a facilidade e poder providos pela linguagem python no que tange à análise e manipulação de dados, ela foi escolhida como o principal recurso de implementação. Com ela, bibliotecas como a sklearn puderam ser exploradas, fator que contribuiu positivamente para as análises seguintes. Em especial, algumas métricas foram utilizadas, como o índice de Rand ajustado e a silhueta. Todo o código desenvolvido pode ser encontrado no repositório do Github (<https://github.com/LucasPMM/greedy-kmeans>).

Além disso, para cada conjunto de dados, foram realizadas 30 execuções. Isso ocorre pois a escolha inicial do centros é aleatória, fator que pode influenciar significativamente na qualidade dos resultados. Por outro lado, 10 conjuntos de dados diferentes foram avaliados, pois, dessa forma, pode-se classificar a performance do algoritmo em diferentes situações. Assim, com o intuito de obter uma média melhor e evitar resultados extrapolados, tais medidas foram tomadas.

3. Implementação

A implementação foi subdividida em algumas etapas para facilitar a compreensão de todo o processo. Serão detalhados a seguir os passos seguidos.

3.1. Tratamento dos dados

Inicialmente, foram buscadas bases de dados que atendessem o objetivo do trabalho. Tal tarefa se apresentou bastante árdua dado o fato da quase inexistência de grupos de dados devidamente formatados e padronizados. Assim, foi necessário, em alguns casos, organizar as colunas, inserir os nomes dos atributos no topo do arquivo, entre outros. Além disso, foi necessário realizar um pré processamento dos dados, utilizando a biblioteca pandas, de forma a ajustar mais imperfeições. Em suma, como foram utilizados datasets numericos, as colunas não numericas foram removidas e as colunas sem dados foram preenchidas com zeros.

3.2. Matriz de distâncias

O cálculo da matriz de distâncias entre os pontos dos datasets foi uma das etapas mais decisivas no que se refere ao tempo de execução final do código. Tal matriz é calculada somente 1 vez entre as 30 execuções, pois seus dados não mudam, mas seu impacto no tempo de execução é avassalador. Conjuntos com mais de 10.000 instâncias e muitos atributos demoraram horas para serem processados, então foi definido um limite superior não muito maior do que 5.000 instâncias e 30 atributos.

Em um primeiro momento, toda a matriz era calculada. Entretanto, um fato importante foi notado: a parte triangular superior era idêntica à triangular inferior e a diagonal principal era nula. A resposta para a diagonal é imediata: a distância entre o elemento $A[i][i]$ e ele mesmo é zero. Por outro lado, também é possível perceber que a distância entre os elementos X e Y é igual a distância entre Y e X . Assim sendo, foi alterada a lógica inicial de forma que somente a matriz triangular superior fosse calculada, diminuindo consideravelmente o tempo de execução dessa etapa.

```

def matrix_of_distances(p = 1):
    distances = np.zeros((n_samples, n_samples))

    for diag in range(0, n_samples):
        for row in range(0, n_samples - diag):
            col = row + diag
            if row == col:
                distances[row][col] = 0.0
            else:
                sample = rows[row]
                target = rows[col]
                distances[row][col] = minkowski_distance(sample, target, p)

    return distance

```

Figura 1. Implementação do cálculo de distâncias.

3.3. Algoritmo guloso

A lógica do algoritmo guloso implantado, discutida previamente na seção anterior, será mais detalhada agora. Novamente, é avaliado se temos mais centros do que pontos disponíveis e, caso positivo, retornamos o próprio conjunto de dados. Seguimos com a escolha aleatória do centro inicial e iniciamos um loop que só para quando os K centros forem encontrados. Iteramos todos os pontos e, caso ele já não tenha sido escolhido para ser um centro, pegamos o seu centro mais próximo. Ao final do loop interno, é escolhido o ponto de maior distância e o adicionamos ao conjunto de centros.

```

def K_Means(distances):
    if K > n_samples:
        return data
    centroids = np.random.choice(n_samples, 1).tolist()
    while len(centroids) < K:
        candidates = []
        for idx_row in range(n_samples):
            s_is_centroid = [idx_row == centroid for centroid in centroids]
            if not any(s_is_centroid):
                options = []
                for centroid in centroids:
                    indexes = select_index(idx_row, centroid)
                    options.append(distances[indexes['row']][indexes['column']])
                closest_centroid = min(options)
                candidates.append({'distance': closest_centroid, 'index': idx_row})

        if len(candidates) > 0:
            max_item = max(candidates, key=itemgetter('distance'))
            s = max_item['index']
            centroids.append(s)

    return centroids

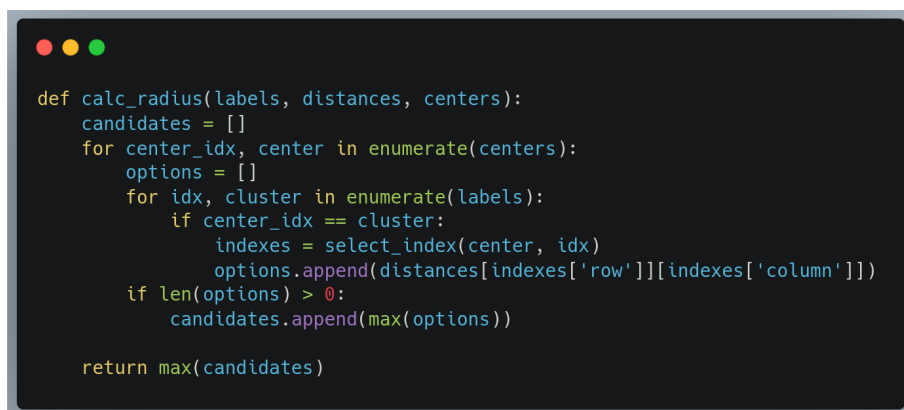
```

Figura 2. Implementação do K-means.

3.4. Funções auxiliares

Algumas funções auxiliares foram implementadas para suprir as necessidades e resultados esperados. Entre elas, a mais importante está relacionada ao cálculo final do

raio obtido. Já com os centros e distribuição de classes definidas em cálculos anteriores, inicia-se uma iteração por todos os centros. Para cada um deles, utilizamos a matriz de distâncias, anteriormente calculada, para encontrar o elemento mais distante que pertença ao cluster em questão. Dado o elemento mais distante do centro de cada um dos K clusters, retornamos o máximo deles, o qual representará o raio do maior cluster e, consequentemente, será o raio final do algoritmo.



```
def calc_radius(labels, distances, centers):
    candidates = []
    for center_idx, center in enumerate(centers):
        options = []
        for idx, cluster in enumerate(labels):
            if center_idx == cluster:
                indexes = select_index(center, idx)
                options.append(distances[indexes['row']][indexes['column']])
        if len(options) > 0:
            candidates.append(max(options))
    return max(candidates)
```

Figura 3. Cálculo do raio.

4. Experimentos

Uma das etapas mais importantes de todo esse projeto consiste na experimentação do algoritmo desenvolvido bem como na análise dos resultados obtidos. Além da implementação gulosa, foi utilizado a implementação do K-means pela biblioteca sklearn. Dessa forma, foi possível avaliar se o raio encontrado, por exemplo, era limitado superiormente por 2 vezes o raio retornado pela sklearn.

4.1. Bases utilizadas

Diversas bases foram avaliadas no decorrer da implementação. Contudo, nem todas atendiam as necessidades proposta pelo código desenvolvido. Após longos períodos de busca, uma configuração final foi atingida e os grupos utilizados podem ser encontrados no repositório do Github. Em suma, todas possuem pelo menos 700 instâncias (e no máximo 5500) e o número de classes varia entre 2 a 6.

4.2. Resultados obtidos

Com a execução do algoritmo em todas as bases selecionadas, mais de 70 tabelas foram montadas (disponíveis no Github: <https://github.com/LucasPMM/greedy-kmeans/tree/main/tables>). A partir delas, diversas conclusões podem ser tomadas. Vamos pegar uma amostra das médias de algumas métricas utilizadas.

É importante ressaltar a relevância da normalização dos dados para uma análise qualitativa. De maneira proposital, alguns datasets foram normalizados, enquanto outros não. Tal processo assegura que todos os pontos a serem processados estarão na mesma ordem de grandeza. Em conjuntos numéricos com dados entre 0 e 1, por exemplo, tal procedimento não apresenta grande impacto. Todavia, na medida que o valor absoluto dos

Dataset	K	Instances	Radius	Desvio padrão	Silhouette	Rand index	Time
#1	2	1151	571.06257	59.70561	0.45113	0.00103	0.00711
#2	3	1353	8.56667	0.4714	0.12593	0.19026	0.01323
#3	3	1473	22.6	1.82087	0.30133	0.00775	0.01378
#4	2	961	66.36667	7.98638	0.66726	0.00253	0.00533
#5	2	958	10.13333	0.57349	0.07166	-0.00121	0.00532
#6	3	4424	1935.72213	344.81894	0.44793	0.02072	0.0259
#7	6	4435	805.46667	35.61474	0.24918	0.24978	0.07009
#8	5	5473	93.57687	0.45691	0.0937	0.09695	0.06534
#9	3	4177	4.5948	0.10132	0.21514	0.11747	0.02454
#10	6	1599	33.17163	3.5963	0.03726	0.02443	0.04021

Figura 4. Média dos resultados das 30 execuções de cada dataset (considerando $P = 1$ na distância de Minkowski)

dados cresce, ou quando o conjunto em questão possui um subconjunto inúmeras vezes maior que todo o resto, a normalização se faz extremamente necessária. Um exemplo normalizado seria o conjunto #9 e um não normalizado seria o #6. É perceptível o impacto desse processo no valor do desvio padrão: dados com ordens de grandezas aproximadas obtiveram um desvio padrão, relacionado ao valor do raio, menor.

O cálculo do raio foi outro fator crucial em toda a análise. As 30 execuções se fazem necessárias devido ao fato do primeiro centro ser escolhido de forma aleatória. Assim, em determinadas instâncias, a escolha inicial pode gerar uma resposta ruim. Para avaliar a qualidade da resposta foi utilizada a biblioteca sklearn. A lógica utilizada nela difere da abordagem gulosa aproximada. A aleatoriedade ainda está presente, contudo, diversas iterações são realizadas de forma a provocar uma convergência dos centros, fator que aproxima muito o resultado do valor ótimo. Com tudo isso, observou-se que todos os valores dos raios obtidos foram estritamente maiores que o valor retornado pelo sklearn e menores do que 2 vezes o mesmo valor, comprovando o fator de aproximação 2 do algoritmo. De forma resumida, alguns grupos de dados analisados tiveram raios mais próximos do ótimo, enquanto outros caíram praticamente nos piores cenários.

Ainda sobre o raio, a natureza do conjunto analisado é determinante para a qualidade de muitas das métricas em questão. Um exemplo: determinar um agrupamento de espécies de flores dado o tamanho de suas pétalas é mais coerente do que determinar um agrupamento de cores de cabelos dadas as alturas das pessoas. Dessa forma, a análise do raio por si só é insuficiente para avaliar a real virtude do algoritmo, pois pode-se ter raios com valores dentro do esperado, mas clusters agrupando elementos que deveriam estar separados.

O tamanho do conjunto de dados também é outro fator determinante no tempo de execução. A matriz de distâncias foi calculada apenas uma vez, pois os valores dela não mudam, então foi considerado apenas o tempo de execução do cálculo dos centros e raios. Exemplificando: o conjunto #4, com 961 instâncias, executou cerca de 12 vezes mais rápido que o conjunto #8, o qual possui 5473 instâncias. Mais instâncias, auxiliadas por um K maior e numerosos atributos, significam mais iterações e cálculos a serem

processados. Além disso, a implementação do KMeans do sklearn executou muito mais rápido do que a versão gulosa (quando considerado o tempo de construção da matriz de distâncias), sendo que a primeira levou segundos e a última levou minutos.

No que tange ao cálculo das distâncias, o valor P escolhido para distância de Minkowski também altera os resultados obtidos. Supondo que a distância entre X e Y seja Z para $P = 1$, sabe-se que a mesma, para $P = 2$, será menor ou igual a Z. Assim, a "grandeza" dos resultados é inversamente proporcional ao valor absoluto de P. O desvio padrão, por exemplo, para o dataset #2 é aproximadamente 0.47 para $P = 1$ (Figura 4) e 0.13 para $P = 2$ (Figura 5).

Dataset	K	Instances	Radius	Desvio padrão	Silhouette	Rand index	Time
#1	2	1151	24.319	25.99697	0.81586	-0.00056	0.00655
#2	3	1353	3.84907	0.13513	0.12686	0.15081	0.01283
#3	3	1473	13.6761	1.12044	0.34789	0.0099	0.01404
#4	2	961	54.32767	4.20551	0.48334	0.02488	0.00496
#5	2	958	3.5476	0.1241	0.07873	-0.00266	0.00556
#6	3	4424	1399.25897	329.25572	0.45146	0.01775	0.02587
#7	6	4435	166.13853	7.12039	0.25617	0.2162	0.07002
#8	5	5473	72.00663	0.46992	0.08519	0.11774	0.06665
#9	3	4177	2.35913	0.04538	0.25691	0.10038	0.02439
#10	6	1599	14.9222	0.76049	0.06143	0.02783	0.04021

Figura 5. Média dos resultados das 30 execuções de cada dataset (considerando $P = 2$ na distância de Minkowski)

Como mencionado anteriormente, a relação entre os dados do conjunto é decisiva nos resultados. Assim, nem sempre o melhor agrupamento foi obtido, considerando ótimo o agrupamento em que itens da mesma classe ocupam o mesmo cluster. Não há como circundar tal limitação, entretanto, os resultados podem ser interpretados e explicados de acordo. Além disso erros de medida devem ser considerados pois aproximações e arredondamentos foram utilizados de forma a obter respostas com um número de casas decimais aceitável.

Por fim vamos analisar 2 métricas utilizadas para a avaliação de agrupamentos: o índice de Rand ajustado e a silhueta. O índice de Rand é uma medida que caracteriza a similaridade entre dois agrupamentos e sua versão ajustada lida com outliers (pontos fora da curva), traçando uma linha de base esperada para semelhança entre os agrupamentos. Ele varia de 0 a 1, sendo diretamente proporcional à otimalidade do agrupamento, contudo, pode apresentar valores negativos caso o índice seja menor do que o esperado. A silhueta varia de -1 a +1: a proximidade a +1 representa maior distância entre os clusters (clusters bem separados e definidos); mais próximo de 0 representa proximidade entre os clusters (dados "em dúvida" acerca de qual cluster pertencem); mais próximo a -1 representa maior "imperfeição" do agrupamento (possivelmente os dados estão no cluster errado).

A silhueta obtida, em média, foi bastante satisfatória, apresentando valores positivos e, em alguns casos, maiores do que +0.5, mostrando a boa definição dos clus-

ters. O índice de Rand ajustado, por sua vez, não performou tão bem. Pelos motivos de coesão e coerência anteriormente discutidos, tal métrica foi bastante impactada. No entanto, ao avaliar os valores retornados pelo KMeans da sklearn, é perceptível que ambas implementações sofrem das mesmas limitações.

Dataset	K	Instances	Radius	Silhouette	Rand index	Time
#1	2	1151	480.391	0.43638	-0.00149	0.88543
#2	3	1353	7.341	0.2209	0.23929	0.88405
#3	3	1473	18.943	0.44213	0.02604	0.90189
#4	2	961	59.559	0.54053	0.08807	0.14713
#5	2	958	7.011	0.09592	0.00073	0.13771
#6	3	4424	1492.486	0.62881	0.00277	1.14509
#7	6	4435	760.764	0.32337	0.53456	1.36935
#8	5	5473	73.165	0.51604	0.0979	1.33425
#9	3	4177	2.744	0.50991	0.17846	1.22751
#10	6	1599	18.771	0.19422	0.06206	0.91376

Figura 6. Resultados obtidos pela biblioteca SKLearn para os conjuntos de dados analisados (considerando $P = 1$ na distância de Minkowski)

5. Conclusão

A execução desse trabalho se apresentou como uma ótima oportunidade de aplicar diversos conceitos vistos em sala de aula de forma prática, além de instigar a curiosidade sobre os fatores que levam a um determinado resultado: saber os motivos de uma métrica divergir do valor esperado é fundamental.

Com o desenvolvimento de toda a análise apresentada na seção anterior, pode-se perceber que, apesar do raio estar dentro da aproximação desejada, nem sempre os agrupamentos formados são satisfatórios. Coerência entre os dados, ordem de grandeza, normalização, número de instâncias, número de centros, entre outros, são alguns dos fatores que, em conjunto, determinam o resultado e, na medida de sua importância, devem ser investigados.

Referências

- [1] Slides disponibilizados pelo professor para a ministração da disciplina
- [2] SKLearn documentation (<https://scikit-learn.org/stable/>)