Trabalho Prático 2
Solução para problemas difíceis

Leandro Freitas de Souza[1]

[1]Departamento de Ciência da Computação – Universidade Federal de Minas Gerais
Belo Horizonte – MG – Brasil

1. Introdução

Neste trabalho, foi estudado o problema dos k-centros, que é classificado como NP-Completo, o que o torna inviável para instâncias muito grandes. Dessa maneira, para contornar esse problema, vemos algorítmos aproximativos que, embora não consigam chegar na solução ótima, chegam a uma distância aceitável e aproximada, além de ser calculado em tempo polinomial, o que o torna extremamente viável.

A entrada do problema consiste de um conjunto finito $S = \{s_1, \cdots, s_m\} \subset \mathbb{R}^n$ de pontos, com $m, n \in \mathbb{N}$. A partir dessa entrada, o objetivo é achar um conjunto $C = \{c_1, \cdots, c_k\} \subset \mathbb{R}^n$, com $k \in \mathbb{N}$, que minimize a distância máxima entre os elementos de S e C, ou seja, minimizar $r(C) = maxdist(s_i, C)$

Em outras palavras, o problema se baseia em agrupar os elementos de um conjunto S em k grupos (determinado pelo centro mais próximo) de forma que o raio desses grupos seja o mínimo possível.

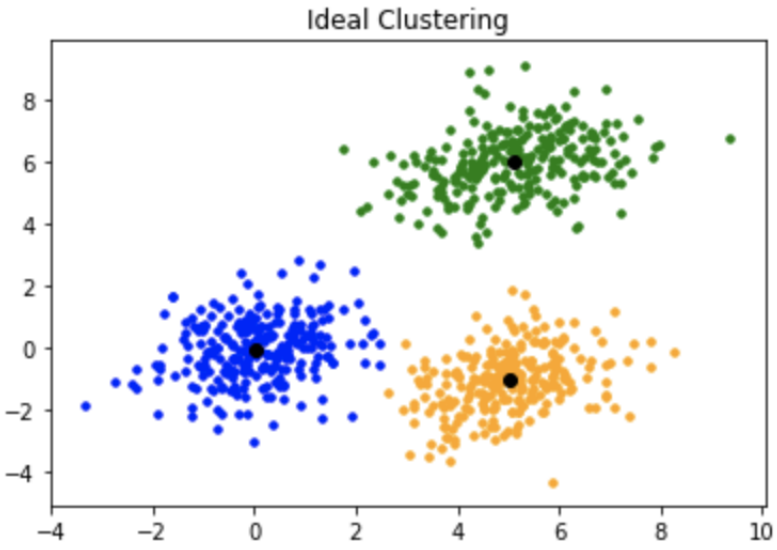

Figura 1. Ilustração dos K centros

2. Métodos e Métricas Usados

2.1. Métodos

Como afirmado na sessão anterior, o problema dos k-centros é NP-Completo, o que nos leva a utilizar algorítmos aproximativos para que consigamos algo em tempo polinomial.

Algorithm 1 K-Centros guloso 2-Aproximado

procedure KCENTERS(S, k)
  if $k \geq |S|$ **then return** S
  end if
  $C \leftarrow \emptyset$
  $s \leftarrow randint(1, |S|)$
  $C \leftarrow C \cup \{s\}$ ▷ C é inicializado com um elemento aleatório de S
  while $|C| < k$ **do**
    $s \leftarrow argmaxdist(S, C)$
    $C \leftarrow C \cup \{s\}$
  end while
  return C
end procedure

Como visto em aula, existe um algorítmo guloso 2-Aproximado para o problema, que funciona da seguinte maneira:

Esse algorítmo consiste em primeiramente selecionar um elemento arbitrário de C para incluir em S, e depois ir selecionando em incluindo em C os elementos de S mais distantes de C, até que $|C| = k$

Sabemos que esse algorítmo é 2-Aproximado, como provado no livro utilizado pela matéria [Kleinberg and Tardos 2006], o que significa que o raio máximo encontrado por meio desse algorítmo será no máximo duas vezes maior que o raio ótimo.

2.2. Métricas

Para calcular $Dist(S, C)$, foi utilizada uma métrica específica: a distância de minkowski. Essa métrica é definida como:

$$Dist(X, Y) = \left(\sum_{i=1}^{n} |x_i - y_i|^p \right)^{\frac{1}{p}}$$

Sendo que $p \in \mathbb{N}$ é um parâmetro passado na entrada do programa. Nos casos em que $p = 1$, $p = 2$ e $p \to \infty$ correspondem respectivamente às distâncias de Manhattan, Euclidiana e de Chebyshev.

Além disso, para calcular a precisão dos resultados encontrados, foram utilizados dois tipos de medidas clássicas para a avaliação: silhueta e índice de rand aleatório. Essas métricas não foram implementadas, e foram importadas de uma biblioteca do python. Enquanto a Silhueta mede o quão sepadrados os clusters encontrados estão, o índice de Rand aleatório mede a distância da solução à solução ideal.

3. Implementação

Esse trabalho foi implementado e compilado completamente em Python. Foram utilizados três arquivos para a execução do problema.

No primeiro arquivo, cujo nome é 'KCenters.py', é implementada a classe 'KCenters', que recebe a quantidade de centros k a ser calculada e o índice p. A partir disso,

ela possui a função $getMinkowskiDistances(S)$, que calcula a matriz de distâncias do conjunto S, a função $GreedyKCenters(S)$, que executa o algorítmo guloso e retorna os índices dos centros selecionados, e por fim a função $getLabels$, que retorna um vetor com os centros assignados a cada ponto.

O segundo arquivo possui a função $Evaluate(S, labelIndex, p)$, que filtra a coluna de rótulos do dataset e retorna um relatório que possui a média e o desvio padrão do tempo, índice Silhueta, índice Rand e raio máximo, englobando 30 execuções do algorítmo aproximativo. Além disso, o relatório contém os mesmos resultados do algorítmo implementado na biblioteca de python 'Sklearn' e o raio ótimo, como visto na figura 2

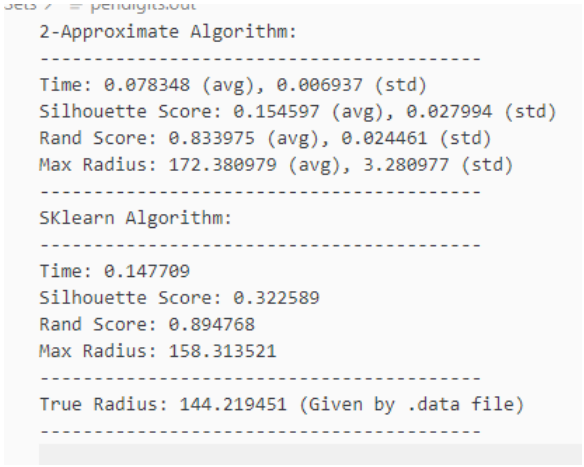

Figura 2. Saída do programa

Por fim, o terceiro arquivo é responsável pela leitura dos repositórios e pela chamada das funções.

Além disso, na chamada da função, podemos usar os argumentos '-p', que recebe o índice usado na distância de Minkowski, 'c', que recebe a coluna do repositório em que as classes ideais estão atribuídas e '-i', que recebe o nome do arquivo de entrada, como pode ser visto na figura 3

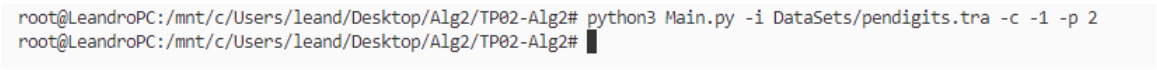

Figura 3. Chamada do programa

Referências

Kleinberg, J. and Tardos, E. (2006). *Algorithm Design*. Addison Wesley.

4. Experimentos

Foram feitos os seguintes experimentos em 10 datasets diferentes:

Nome do Arquivo	p_index	Tempo	Silhueta	Rand	Raio
CTG1.data	1	0.005473	0.416651	0.632899	521.236667
CTG1.data	2	0.005348	0.566821	0.643582	278.568826
CTG1.data	5	0.010752	0.729565	0.633759	244.005226
CTG2.data	1	0.019083	0.134163	0.631287	320.590000
CTG2.data	2	0.019269	0.228682	0.511607	135.489709
CTG2.data	5	0.034958	0.265496	0.492868	95.011997
letter-recognition.data	1	1.082332	0.067887	0.865533	38.766667
letter-recognition.data	2	3.381425	0.073623	0.862158	12.978237
letter-recognition.data	5	0.969378	0.058429	0.856721	7.905548
madelon_train.data	1	0.003310	0.021277	0.502218	15769.366667
madelon_train.data	2	0.003192	0.059779	0.501984	1194.220669
madelon_train.data	5	0.005262	0.141885	0.501416	537.064859
optdigits.data	1	0.043405	0.102101	0.846807	260.866667
optdigits.data	2	0.040287	0.096070	0.840624	50.976807
optdigits.data	5	0.068323	0.064266	0.828053	22.558111
pendigits.data	1	0.078524	0.153191	0.832894	526.700000
pendigits.data	2	0.081233	0.156222	0.834801	172.312815
pendigits.data	5	0.081882	0.160488	0.844864	104.860312
segmentation.data	1	0.012509	0.451757	0.425305	741.773067
segmentation.data	2	0.012855	0.427938	0.268536	358.753655
segmentation.data	5	0.015565	0.558002	0.154567	279.388587
UCI.data	1	0.019015	0.127844	0.525443	23.939128
UCI.data	2	0.019897	0.140495	0.522581	11.684957