Projeto 2 - Algoritmos gulosos (agrupamento)

SCC-5900 - Projeto de Algoritmos

Aluno: Afonso Matheus Sousa Lima

Nº USP: 11357812

1 Explicação da Implementação

A implementação é constituída de cinco arquivos escritos em Python3. Ao executarmos o arquivo **clustering_main.py**, passa-se como entrada o nome do arquivo que contém os dados, a quantidade k de *clusters* desejados e qual MST deseja-se usar. Utilizando o nome e localização do arquivo de dados, a classe presente em **graph_builder.py** irá retornar um grafo, representado por meio de um dicionário, onde cada vértice (chave) terá uma lista de distâncias (obtidas com a distância euclidiana) com todos os outros vértices. Também é retornado um dicionário, com os vértices atrelados a seus respectivos valores, e uma lista com todas as arestas formadas, para serem utilizadas no algoritmo de Kruskal.

O próximo passo dependerá da MST escolhida, que estão implementadas no arquivo **MST.py**. Escolhido o algoritmo de Prim, ele recebe como argumentos o grafo e o valor de k. Para gerar a MST com Prim, é escolhido um vértice inicial que será a raiz, para todo vértice, é mantido a informação do seu predecessor e do seu caminho mínimo até a raiz, inicialmente. Utilizando uma lista de prioridades (*heap*) implementada no arquivo **structs.py**, é inserido inicialmente todos os vértices adjacentes ao vértice inicial, priorizando pela menor distância entre eles. Enquanto a *heap* não estiver vazia, o primeiro elemento dela, que tem a menor distância, é retirado e o vértice associado é incluído em uma lista de vértices já visitados. Então, será verificado os adjacentes a esse vértice que não estão incluídos nessa lista, se a distância entre eles for menor que a distância associada atual, a distância mínima desse vértice adjacente será atualizada, bem como a sua posição na *heap* e o predecessor desse vértice adjacente. Na próxima interação, outro vértice de menor distância será escolhido até que não sobre mais vértices. Ao final, será retornado um dicionário onde cada vértice é associado ao seu predecessor, sendo então as arestas da MST. Com a MST, é possível localizar no grafo as distâncias de suas arestas, permitindo que elas sejam ordenadas decrescentemente

e assim são retiradas as k maiores arestas. O conjunto final de arestas é retornado posteriormente.

Escolhido o algoritmo de Kruskal, ele recebe como argumentos o grafo, a lista de arestas e o valor de k. Para gerar a MST com Kruskal, primeiramente, as arestas são ordenadas crescentemente e para cada vértice do grafo, é criado um *set* que será utilizado na estrutura *Union Find*, presente no arquivo **structs.py**. Então, todas as arestas serão percorridas e os vértices associados são colocados na mesma componente utilizando a estrutura *Union Find* e adicionadas ao conjunto de arestas que formam a MST. Se os vértices de uma aresta já estiverem na mesma componente, a adição daquela aresta formará um ciclo, então ela é ignorada. Ao final, é retornado a lista de arestas que formam a MST. Com essa lista, é possível retirar as k últimas arestas, desconectando a MST e retornando uma nova lista com essas arestas.

Ambos os algoritmos retornam arestas que podem ser usadas para formar um grafo que terá k componentes conexas, uma vez que qualquer aresta removida de uma MST desconecta o grafo e são removidas k - 1 arestas. Portanto, cada componente representa um *cluster* e uma classe de 1 até k pode ser associado aos vértices que pertencem a cada um deles. Com isso, temos k *clusters* formados que pode-se comparar com o resultado da clusterização apresentada na descrição do projeto utilizando o *Rand index*. A parte de formação dos *clusters* e atribuição das classes é feita pela classe do arquivo **plot_clusters.py**.

2 Executando o Algoritmo

O algoritmo pode ser executado utilizando o comando:

```
python3 clustering_main.py <nome-do-arquivo> <k> <mst>
```

Onde **k** é um valor maior que 0 e **mst** pode ser *"kruskal"* ou *"prim"*. É retornado a concordância pelo *Rand index* entre o resultado do algoritmo e o resultado do trabalho relacionado apresentado na descrição do projeto. Também é exportada uma figura com os *clusters* formados pelo algoritmo, sendo salva na pasta **out/**. Pode ser necessário instalar os seguintes módulos:

```
pip3 install seaborn
sudo apt-get install python3-tk
```

Códigos e imagens disponíveis nesse repositório do GitHub.

3 Resultados

Ao observarmos os gráficos da Figura 1, percebe-se que há maior concordância entre os resultados dos algoritmos de clusterização com MST e do algoritmo apresentado no projeto quando k é igual 5.

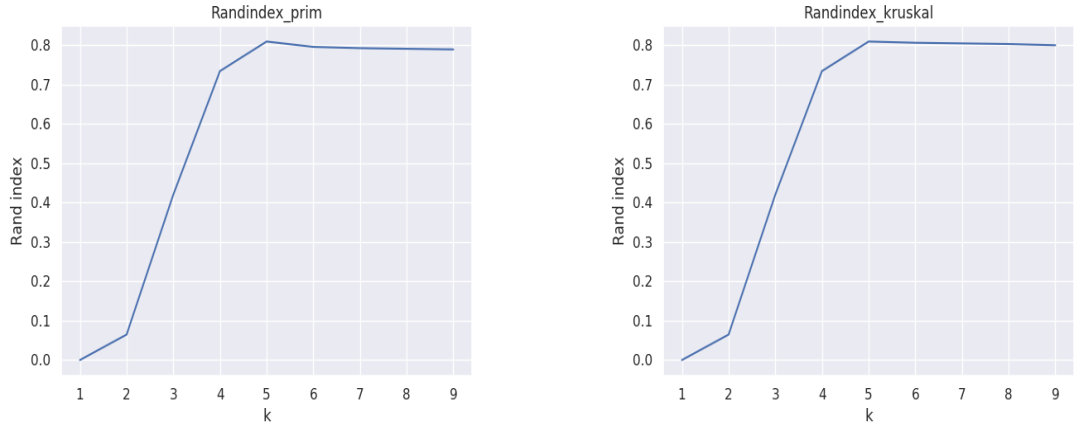

Figure 1: Rand Index com k variando de 1 até 9

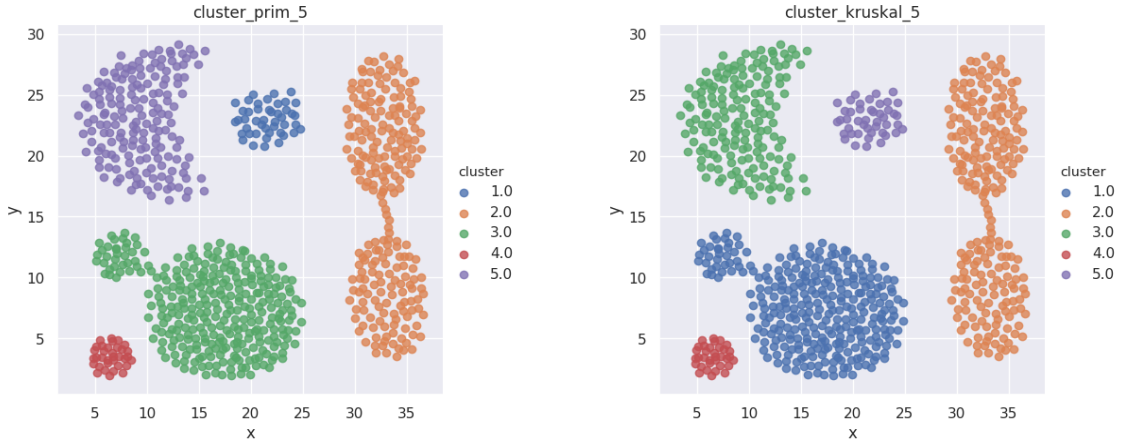

Figure 2: *Clusters* formados com k igual a 5

A Figura 2 mostra a clusterização de ambos algoritmos com MST com k igual a 5.

A partir de 5, a concordância tende a cair. Isso ocorre por causa da natureza dos algoritmos com MST, como estamos removendo as arestas de maior peso, conseguimos separar grupos que estão mais distantes e assim conclui-se que eles pertencem a um *cluster* diferente. No entanto, ele não conseguira identificar *clusters* que que estiverem interligados entre si por um conjunto de pontos. É justamente o que ocorre em ambos os casos, se observarmos o *cluster* laranja, há uma linha de pontos muito próximos interligando dois agrupamentos diferentes visíveis. Como o algoritmo remove as arestas mais distantes para criar os *clusters*, as arestas que formam essa linha de pontos dificilmente será retirada.