O objetivo deste trabalho é implementar o algoritmo de Llyod para realizar um *partitional clustering* (tal como K-mean) com diversas métricas e representantes. Vamos considerar apenas dados de tipo numéricos mesmo se o algoritmo suporta dados binários ou nominativos.

1 Implementação das funções

O ficheiro `xclara.csv` é constituído por 2 atributos numéricos e notamos por $\mathcal{A} = A_1 \times A_2$ o espaço dos atributos enquanto $D = (x^n)_{n=1,\cdots,N}$ representa o conjunto de dados, $x_i^n \in A_i = \mathbb{R}$. Por outro lado $\mathcal{M} = \{m^1, m^2, \cdots, m^K\}$ é o conjunto de representantes da partição $\mathcal{C} = \{C^1, C^2, \cdots, C^K\}$. Nota que, as variáveis dos scripts associadas podem ter nomes diferentes

A matriz `D` de N linhas e d colunas contém os dados. Notamos por K o número de clusters, a matriz `rep` de K linhas e d coluna contém os representantes. Finalmente, o vetor `clusters` de dimensão N identifica os clusters, a saber, `clusters[n]` $\in \{0, 1, \cdots, K-1\}$ é o número do cluster do elemento x^n. A tabela `rep` implementa o conjunto $\mathcal{M}$ enquanto `clusters` é uma implementação da partição $\mathcal{C}$.

Um script inicial `partitional_initial.py` é disponibilisado no CoCal e contém a estrutura do programa, as tabelas para realizar a gestão dos clusters assim como a designação das funções para implementar.

1.1 Inicialização e distância

A função `my_distance` define a métrica que usamos na construção dos clusters.
A routina `initialise_representative(D,K)` retorna uma tabela dos representantes para inicializar o ciclo de Llyod. Várias possibilidades para inicializar os representantes:

- Usar os K primeiros elementos do conjunto de dados D.
- Escolher aleatoriaremente K elementos em D.
- Escolher aleatoriaremente um primeiro elemento m^1. Depois escolher um segundo elemento m^2 tal que $dist(m^1, m^2)$ seja máxima. A seguir. Para k representantes, determinamos o elemento m^{k+1} tal que a quantidade

$$\sum_{\ell=1}^{k} dist(m^\ell, m^{k+1})$$

  seja máxima.

1.2 Implementação da função `assigment`

O operador de *assigment* determina o cluster de cada elemento x^n em função do representante. Matematicamente, trata-se da operação $\mathcal{M} \to \mathcal{C}$. No script, temos de construir a tabela `clusters` a partir da lista dos representantes `rep`.

Recordamos que o evento x^n pertence ao cluster C^k se

$$dist(x^n, m^k) \leq dist(x^n, m^\ell), \quad \forall \ell \in \{0, 1, \cdots, K-1\}.$$

Ao nível do script, identificamos o valor do k associado ao elemento x^n e definimos `clusters[n]=k`.
A função retorna `clusters`.

1.3 Implementação da função `centroid_mean`

O operador de *representative* determine o representante de cada cluster. Matematicamente, trata se da operação $\mathcal{C} \to \mathcal{M}$. No script, temos de construir a lista dos representantes `rep` a partir da tabela `clusters`. Numa primeira implementação, a função corresponde à construção das médias (centróide) definidas por

$$m^k = \frac{1}{|C^k|} \sum_{x^n \in C^k} x^n.$$

Ao nível do script, identificamos os elementos de C^k com os elementos x^n tal como `clusters[n]=k`. A função retorna `rep`.

1.4 Experimentação e visualização

Antes implementar o ciclo de Llyod, temos de verificar que as duas funções estão corretas e precisamos visualizar o resultado. O `plot` dos clusters está realisado com `plt.scatter`, como indicado no script.
Experimentar com um pequeno número de dados. Assegurar que os centróides estão corretos e que o `assigment` produz os bons clusters.

2 Algoritmo de Lloyd e validação dos clusters

Nesta segunda secção, implementamos o procedimento de clustering $K - mean$ assim como algumas ferramentas para analizar a qualidade dos clusters.

2.1 Algoritmo de Llyod

De forma sintética, o algorítmo de Llyod é o seguinte

```
1 while (go_on):
2     clusters=assigment(representative)    #M -> C
3     old_rep=rep                           #guardar em memoria
4     rep=representative(cluster)           # C -> M
5     error=differ_rep(old_rep,rep)         # C_old==C?
6     ncount++                              # count the nb of iter
7     go_on=(error>1E-6) and (ncount<n_MAX) # go_on condition
```

Podemos visualizar a evolução dos representantes em função das iterações (gráficos) e também a evolução do erro entre os representantes sucessivos.

2.2 Quantificações

Vamos definir duas ferramentas para quantificar a qualidade do clustering.

- A função `within_clusters` vai calcular o desvio entre os pontos e o representante de um mesmo cluster C^k (within cluster error or intra-cluster error) com a fórmula

  $$errW_k = \frac{1}{|C^k|} \sum_{x^n \in C^k} dist(x^n, m).$$

  A função retorna um vetor de dimensão K correspondente aos *within cluster errors* dos K clusters.

- A função `between_cluster` vai calcular o erro entre os representantes dos clusters C^k (between cluster error or inter-cluster error) com a fórmula

  $$errB_k = \min_{\ell \neq k} dist(m^k, m^l).$$

A função retorna um vetor de dimensão K correspondente aos *between cluster errors* dos K clusters.

2.3 Clustering index

A principal dificuldade no *partitional clustering* é a avaliação do valor de K. Para determinar o número de cluster, cria-se um index em função de K cujo valor extremo determine a melhor qualidade do clustering. Neste trabalho, propomos o critério

$$idx(K) = \frac{\max_k(errW_k)}{\min_k(errB_k)}$$

onde os vetores de erros (intra e inter clusters) dependem do K. O mínimo deve corresponder ao cluster otimal. Experimentar com a base de dados o index para determinar o número ideal de clusters.

Experimentar um critério diferente tal como

$$idx(K) = \min_k \frac{errW_k}{errB_k}.$$

Procurar outras bases de dados para avaliar a capacidade do index em determinar o número de cluster correto.

3 Outras distâncias e representativos

Temos diversos graus de liberdade na definição dos clusters: a escolha da distância e a escolha do representante. O objetivo é analizar o impacto das diferentes variantes no procedimento de clustering.

3.1 Novos representantes

Implementar as funções `centroid_median` e `medoid`. Experimentar e avaliar as diferenças do clustering em função do representante com os dados de `xclara`.

3.2 Novas métricas

Experimentar novas métricas tais como Canberra ou cosine. Determinar o tipo de cluster que podemos formar combinando distâncias e representantes. Quais são as duas configurações mais distintas?

3.3 Para os alunos muito avançados

Supomos que a base de dados é completada com a classe, ou seja, temos ambos os atributos x^n mas também a classe associada $y^n \in \{cl_1, cl_2, \cdots, cl_{\overline{K}}\}$ onde cl_k são os $\overline{K}$ valores da classe.

Idealmente, temos $\overline{K}$ clusters $\overline{\mathcal{C}} = \{\overline{C^1}, \overline{C^2}, \cdots, \overline{C^{\overline{K}}}\}$ associado a cada classe. Por outro lado, o clustering (sem conhecimento dos labels) vai determinar o número de cluster K e a partição associada $\mathcal{C}$. A questão é saber se temos $\overline{K} = K$ e $\overline{\mathcal{C}} = \mathcal{C}$. Neste trabalho, vamos usar o ficheiro `iris.csv` (com 4 atributos e uma classe nominativa de 3 valores).

- Avaliar o K com os diferentes índices. Conseguimos encontrar o valor correto?

- Fixamos $K = \overline{K}$ e procedemos ao clustering. Comparamos o clustering exato $\overline{\mathcal{C}}$ com o clustering calculado $\mathcal{C}$ do modo seguinte.

  Seja $\overline{C^k}$, notamos k' o indice tal como

  $$k' = \arg\max_{\ell=1}^{K} |\overline{C^k} \cap C^\ell|,$$

  seja, o cluster calculado que tem o maior número de elementos que $\overline{C^k}$. O valor de *matching* associado é

  $$s_k = \frac{|\overline{C^k} \cap C^{k'}|}{|\overline{C^k} \cup C^{k'}|} \in [0, 1].$$

  Obtemos uma semelhança s de matching, calculando a média ou o máximo dos índices. Uma semelhança de $s = 1$ indica um clustering perfeito. Valores abaixo, indicam o nível de proximidade (em%).

Realizar este trabalho com o dataset `iris`.