TP n=°5 : K moyennes

Informatique tronc commun 2ère année

Le but de ce TP est d'implémenter et d'afficher l'algorithme des K-moyennes. On se placera dans le cas où les points étudiés font partie de $\mathbb{R}^2$. La fonction des K-moyennes est déjà préconstruite, et permettra d'afficher l'emplacement des représentants et des éléments composants une classe.

1) Exécuter l'instruction suivante `afficher_liste_points(li)`. La liste points est une liste de couples de flottants, et c'est cette liste qu'on va essayer de couper en différentes classe.

1 Fonction préalable

Une des étapes de l'algorithme des K-moyennes est le calcul d'un barycentre des classes temporaires trouvées. Une classe de points sera stockée sous la forme d'une liste de couples de flottants. Par exemple, la liste `C1 = [(1.5,2.3),(1.2,2.2)]` est une liste qui contient deux points.

Le barycentre $\vec{B_P}$ d'un ensemble de points P est :

$$\vec{B_P} = \frac{1}{|P|} \sum_{\vec{p} \in P} \vec{p}$$

2) Écrire une fonction `barycentre(liste_p)` qui prend en entrée une liste de couples de flottants nommé `liste_p` et qui renvoie un couple correspondant au barycentre des points de `liste_p`.

3) Écrire une fonction `barycentre_classe(liste_classe` qui prend en entrée une liste de liste de couples de flottants nommé `liste_classe` et renvoie la liste des barycentres de chaque sous-listes de `liste_classe`.

L'autre étape importante de l'algorithme des K-moyennes est d'associer chaque point au barycentre dont il est le plus proche.

4) Écrire une fonction `distance_euclidienne(p1,p2)` qui prend deux couples `p1` et `p2` et renvoie la distance euclidienne entre `p1` et `p2`.

5) Écrire une fonction `plus_proche_indice(liste_centre,point)` qui prend en entrée une liste de points `liste_centre` et un point `point`. La fonction renvoie l'indice du centre le plus proche du point donné en entrée.

Par exemple `plus_proche_indice([(1.0,0.0),(1.5,2.0),(-1.0,0.5),(2.0,-1.0)],(-1.2,0.7))` renvoie 2 car le point (-1.2,0.7) est le plus proche de (-1.0,0.5) en position 2.

2 Analyse de l'implémentation

6) Observer la boucle while dans la fonction préécrite `k_moyenne(liste,k)`. Quelle est le risque d'écrire `while True`? Grâce à quelle instruction les appels à la fonction `k_moyenne(liste,k)` vont terminer?

7) Faites un premier appel à la fonction `k_moyenne` avec la liste `li` et `k=4`. Est-ce que l'appel est satisfaisant?

Pour pouvoir un peu de variabilité dans les résultats, on va choisir les premiers centres de manière aléatoire.

8) Modifier la manière d'initialiser la liste `liste_centre` dans la fonction `k_moyenne(liste,k)` pour utiliser la fonction `choix_aléatoire(liste,k)`.

9) Décommenter la ligne `liste_centre = [(1.1,0.3),(1.6,6.8),(-2.7,1.8),(0.8,1.1)]`, et exécuter un appel de `k_moyenne(l,k)` avec k = 4. Que se passe t-il? Pourquoi?

10) Proposer une correction de la fonction `k_moyenne(liste,k)` et/ou des sous fonctions qui composent `k_moyenne(liste,k)` pour prendre en compte ce problème.

Dans le cours, pour savoir si un résultat était meilleur qu'un autre, on s'est proposé de calculer l'écart entre les points d'une classe et de son barycentre.

11) Écrire une fonction `eps_moyen(classe)` qui prend en entrée une liste de points `classe` et calcule la distance moyenne d'écart entre chaque point de la classe et le barycentre de la classe.

12) Écrire une fonction `eps_max(cls)` qui prend en entrée `cls` une liste de classe et renvoie la valeur du maximum de la fonction `eps_moyen(cl)` sur les classes `cl` de la liste `cls`.

Grâce à cet epsilon, on va pouvoir mettre en place un algorithme `k_moyenne_multiple(l,k,nb)`, qui prend en plus des entrées classiques de l'algorithme des k-moyennes un entier `nb`. Le principe de l'algorithme est le suivant :

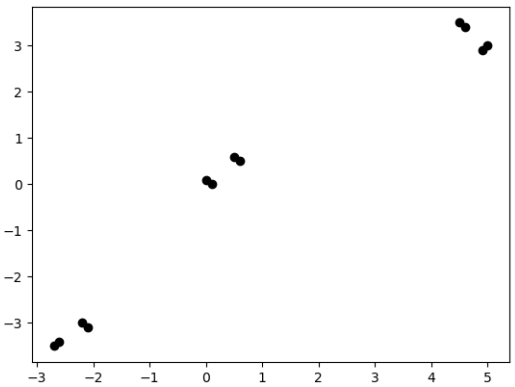
- On fait un premier appel à `k_moyenne(l,k)` et on stocke le résultat dans `cl`.
- On stocke aussi la valeur `eps` correspondant à l'appel de `eps_max(cl)` .
- Puis, on fait un nombre de fois correspondant à nb - 1 fois les instructions suivantes :
  - ** On stocke le résultat d'un appel à `k_moyenne(l,k)` dans cl2
  - ** On calcule la valeur `eps_max(cl2)` dans `eps2`
  - ** Si `eps2` est plus petit que `eps`, on stocke `eps2` dans `eps` et on stocke `cl2` dans `cl`
- On renvoie cl à la fin de la boucle

13) En utilisant la fonction `k_moyenne_sans_aff(liste,k)` qui est l'implémentation de l'algorithme des k-moyennes sans affichage, écrire une fonction `k_moyenne_multiple(liste,k,nb)` qui est l'implémentation de l'algorithme présenté précédemment.

La dernière étape est de trouver une méthode permettant de choisir le meilleur k pour notre algorithme, ainsi que le meilleur nb. Pour avoir une idée des critères permettant de choisir k, on va tracer l'erreur maximum de `k_moyenne_multiple(liste,k,nb)` en fonction de k.

14) Écrire une fonction `tracer_k(liste,nb,max_k)` prenant en entrée une liste `liste` de points et deux entiers `nb,max_k` et trace l'erreur moyenne renvoyée par `k_moyenne_multiple(liste,k,nb)` en fonction de `k` pour `k` variant de 2 à `max_k`.

15) Quelle valeur de k voudrait-on choisir pour la liste `li`? Quelle critère peut-on choisir sur le graphique tracé par l'appel de `tracer_k(li,50,10` pour trouver ce k?



À gauche se trouve une répartition de points, qui n'est pas stockée informatiquement.

16) Quelle serait la forme du graphique tracée par la fonction `tracer_k(liste,nb,max_k)` avec une répartition de points comme à gauche?

17) Écrire une fonction `tracer_nb(liste,nb_max,k,rep)` prenant en entrée une liste `liste` de points et trois entiers `nb_max,k,rep` et trace l'erreur moyenne renvoyée par rep appels de `k_moyenne_multiple(liste,k,nb)` en fonction de `nb` pour `nb` variant de 2 à `max_nb`.