Porteboeuf Benoit
Robin Morgan
FIG2A – Télécom Bretagne
INF413 – Automne 2015

TP K-MEANS

Rapport

Introduction

L'algorithme des k-means est un algorithme assez simple de clustering (regroupement, classification).

L'idée est la suivante :

- créer des groupes de points
- calculer leurs barycentres
- rééquilibrer les groupes en affectant les points au groupe dont le centre est le plus proche
- recalculer les barycentres
- réitérer jusqu'à ce que le système soit stable

Pour plus d'informations, nous vous invitons à lire l'article *An Efficient k-Means Clustering Algorithm:Analysis and Implementation* de Tapas Kanungo, David M. Mount, Nathan S. Netanyahu, Christine D. Piatko, Ruth Silverman, et Angela Y. Wu.

Implémentation

Cet algorithme a été implémenté en python 3 avec une compatibilité pour python 2.

Le fichier est découpé en six parties :
- la définition des classes point et group
- la définition de différentes normes
- l'initialisation
- les instructions à répéter
- deux méthodes de test
- le programme principal (main)

Afin de manipuler au mieux les objets, nous avons décidé de créer deux classes point et group.

Un point est défini par

- un identifiant (numéro),
- une liste de coordonnées dans un espace de dimension n (n étant un entier naturel non nul)
- une chaîne de caractères caractéristique

Cette classe point possède 3 méthodes :

- un constructeur, ayant pour paramètres le nom du point, ses coordonnées et son espèce
- une méthode __str__ qui retourne sous forme d'une chaîne de caractères les éléments d'un point (nom, coordonnées et espèce)
- une méthode __eq__ chargée de vérifier l'égalité entre deux points (égalité au sens de l'identifiant)

Un groupe est défini par :

- son nom
- son centre
- la liste des points qu'il contient

Cette classe possède 5 méthodes :

- un constructeur qui prend en argument le nom du groupe, et qui initialise le groupe avec le nom passé en argument, un centre étant nul et un ensemble de points étant une liste vide
- center qui prend en argument une instance de la classe point et qui sert à définir le centre du groupe
- add qui permet d'ajouter un point à la liste des points du groupe
- remove qui permet de supprimer un point du groupe
- __str__ qui renvoie sous forme d'une chaîne de caractères le nom du groupe, suivi de son center et de la liste des points qui le constitue

Différentes normes ont été codées : la norme euclidienne (norme 2), la norme dite de Manhattan (norme 1) ou encore la norme dite sup' (norme infinie). La possibilité de choisir la norme peut permettre de comparer les performances en terme de temps de calcul et de composition des groupes.

Le programme principal (main) est en fait assez simple

1. on commence par charger le fichier 'data.txt' qui contient des données sur la taille de différentes parties de fleurs d'iris au sein d'un massif
2. un point est créé pour chaque fleur présente dans le fichier
3. on demande le nombre de groupes à créer à l'utilisateur. Si l'utilisateur rentre un nombre qui n'est pas un entier compris entre 1 et le nombre de fleurs, la question est de nouveau posée
4. l'initialisation commence et on crée le nombre demandé de groupes en prenant des centres au hasard parmi les points existants. Les groupes ne sont alors définis que par leur identifiant et leur centre
5. on demande à l'utilisateur la norme qu'il souhaite utiliser. Si l'utilisateur ne rentre ni 'Euclidean', ni 'Manhattan', ni 'Sup', la question est de nouveau posée
6. une fois la norme choisie, le processus itératif débute

Tant que le système n'est pas stable, c'est-à-dire tant que la constitution des groupes n'est pas fixe (donc tant que les centres ne sont pas constants), on va chercher à optimiser les groupes.

Pour cela, pour chaque point, on calcule sa distance avec chaque centre des groupes (à l'aide de la norme choisie par l'utilisateur). Chaque point est alors affecté au groupe du centre dont il est le plus proche et est retiré du groupe précédent si c'est nécessaire.

Une fois la composition des groupes changée, il convient de recalculer les centres. On utilise pour cela un simple calcul d'isobarycentre, puis nous attribuons comme centre à chaque groupe le point du groupe le plus proche du barycentre.

Ces deux opérations sont répétées tant que le système n'est pas stable, donc tant que le calcul des centres produit un résultat différent du résultat précédent.

Enfin, les groupes et leur composition sont affichés et deux tests sont réalisés :

- le premier vérifie que toutes les fleurs au sein d'un même groupe sont bien de la même espèce
- le deuxième vérifie que les normes définies plus haut se comportent de manière cohérentes (résultat numérique, axiome de séparation, positivité et inégalité triangulaire sur quelques exemples)

Résultats

Le programme ainsi implémenté génère rapidement l'ensemble des groupes voulus.

Les différentes normes produisent des groupes à la composition un peu différentes, mais ne produisent pas de différences de temps de calcul notables. Il n'y a donc pas de raison particulière de préférer une norme à une autre. On peut cependant supposer que la norme euclidienne est la norme la plus apropriée des trois car elle tient assez bien compte de l'écartement d'un point au centre par rapport à l'ensemble des composantes (ce qui n'est pas du tout le cas de la norme sup' par exemple).

Nous avons en revanche remarqué une légère variation dans la composition des groupes lorsque le programme est lancé plusieurs fois d'affilé. Cela est probablement dû au choix des centres initiaux et à la condition d'arrêt qui est peut-être un peu trop permissive (en effet, il suffit que les centres soient stables après une optimisation pour que la boucle s'arrête, or nous pourrions très bien avoir un système dans lequel le changement de l'ensemble des points des groupes se compensent eux-même, de sorte que les centres restent identiques).

Par ailleurs, tous les groupes ne comportent pas que des fleurs de la même espèce. En effet, les iris versicolor et virginica se mélangent de manière assez conséquente. Cela est peut-être tout simplement dû à une similarité des espèces et non au programme. Cependant, n'ayant pas de connaissances avancées en botanique, nous n'avons pû trancher.

Conclusion

Cet algorithme basique de clustering et son implémentation en python donne globalement de bons résultats avec une complexité en o(nxm) avec n le nombre de points et m le nombre de groupes.

Ces résultats pourraient cependant être améliorés, notamment en choisissant intelligemment les premiers centres. On pourrait par exemple imaginer une pondération de chaque point en fonction de ses voisins et en déduire les meilleurs candidats pour être des centres de groupes, limitant ainsi drastiquement les itérations du programme.

Une autre amélioration possible serait de déterminer automatiquement le nombre optimal de groupes à réaliser, à noter aussi que les résultats obtenus avec notre implémentation semblent suggérer que la condition d'arrêt est à rendre plus stricte.