Projet Java : Clustering

Instructions

1. Faire une archive (`.zip` ou `.tar`) contenant les fichiers suivants et l'envoyer à `omar.fawzi@ens-lyon.fr` avant le vendredi 9 janvier 2015 à 23h59 :
   (a) Tous les fichiers sources `.java`
   (b) Un fichier README qui indique comment compiler et executer votre programme
   (c) Des fichiers ou code que vous avez utilisés pour tester votre programme
   (d) Le rapport (fichier pdf) fait en LaTeX
   (e) (Optionnel) Un fichier `.jar` qui permet de lancer votre programme

   Merci d'inclure dans le sujet `[Proj1] [Projet]` et de nommer votre archive `NomPrenom.zip`.
2. Chacun devra faire une courte présentation (6 mins) sur son projet. Les présentations la semaine du 12 janvier.

Evaluation du projet

— Correction et performance (le code doit compiler et faire ce qui est demandé)
— Propreté et lisibilité du code (facile à compiler/comprendre/rajouter une fonctionalité, modularité, conventions de nommage respectées, bien commenté)
— Modélisation objet (programme découpé en plusieurs classes, bon usage de l'encapsulation e.g., attributs internes privés, bon usage de l'héritage, etc...)
— Le rapport (qualité de la rédaction, qualité de l'analyse). Voir la page du cours de l'an dernier pour une introduction à LaTeX et les recommendations pour le rapport : http://perso.ens-lyon.fr/maxime.senot/Projet2013.html.
— La soutenance (clarté, respect du temps). Voir aussi la page de l'an dernier pour les instructions.

1 Introduction

Lorsque nous avons une grande quantité de données, une question de base est de classifier les données en plusieurs groupes ou clusters. Par exemple, on peut classifier des articles de recherche par le thème ou des photos par le nom des personnes représentées. Comment effectuer cette tâche de regroupement de manière automatique ?
À ce niveau, ce n'est pas encore un problème algorithmique vu que l'objectif n'est pas clairement défini. Il faut d'abord modéliser les données. Une donnée est en general représentée par un vecteur dans $\mathbb{R}^d$ pour un certain d (en général grand). Pour une image, ce vecteur peut être la valeur de la couleur de chaque pixel par exemple. Une fois que nous avons fixé la représentation des données, il faut essayer de préciser l'objectif qu'on essaie d'atteindre de manière quantitative (et ceci peut dépendre du type de données auxquelles on a affaire). Ensuite, on cherche un algorithme pour optimiser cet objectif.

2 Première partie : Clustering de points dans le plan

Dans cette section, nos données sont des éléments $x_1, \ldots, x_n$ de $\mathbb{R}^2$.

2.1 Algorithme des k-moyennes

Un des algorithmes les plus connus pour le clustering est l'algorithme des k-moyennes. L'objectif que tente de réaliser cet algorithme est de minimiser la somme des carrés des distances des points au centre du cluster :

$$\sum_{i=1}^{k} \sum_{j \in S_i} \|x_j - \mu_i\|_2^2 \ , \tag{1}$$

où S_i est l'ensemble des points x_j qui sont dans le cluster j, et μ_i est la moyenne des points dans S_i. La somme des carrés des distances du centre d'un cluster aux points du cluster quantifie l'étendue du cluster, quantité que l'on souhaite bien sûr minimiser.
L'algorithme des k-moyennes est simple. On commence avec k centres $\mu_1, \ldots, \mu_k$. Pour chaque point $x_1, \ldots, x_n$, on assigne le centre le plus proche (distance euclidienne). Ceci nous permet de définir les k groupes :

$$S_i = \{j : \|x_j - \mu_i\|_2 \leq \|x_j - \mu_{i'}\|_2, \text{ pour tout } i'\}. \tag{2}$$

Pour améliorer le clustering, on va mettre à jour les centres :

$$\mu_i \leftarrow \frac{1}{|S_i|} \sum_{j \in S_i} x_j. \tag{3}$$

L'algorithme alterne les étapes (2) et (3) jusqu'à ce que les centres μ_i cessent de bouger (on pourra vérifier que $|\mu_i - \frac{1}{|S_i|} \sum_{j \in S_i} x_j| \leq c$ pour une petit constante c).

2.2 Diagramme de Voronoi

Maintenant que nous avons trouvé les clusters, nous voulons représenter ces clusters de manière plus générale pour pouvoir classifier tout nouveau point dans le plan. Pour cela, un choix naturel pour les clusters serait

$$V_i = \{y \in \mathbb{R}^2 : \|y - \mu_i\|_2 \leq \|y - \mu_{i'}\|_2, \text{ pour tout } i'\}. \tag{4}$$

L'ensemble des V_i pour $i = 1, \ldots, k$ couvre tout le plan $\mathbb{R}^2$ (en pratique, pour votre programme le plan a des limites finis que vous devrez spécifier dans votre programme). Ce recouvrement s'appelle diagramme de Voronoi. V_i est appelé cellule de Voronoi pour le point μ_i. Notre objectif est maintenant de trouver une méthode pour représenter et calculer les cellules de Voronoi pour l'ensemble de points $\mu_1, \ldots, \mu_k$. Si on définit $h(i, i') = \{y \in \mathbb{R}^2 : \|y - \mu_i\|_2 \leq \|y - \mu_{i'}\|_2\}$, on a par définition

$$V_i = \bigcap_{i' \neq i} h(i, i'). \tag{5}$$

Les demi-plans $h(i, i')$ sont assez facile à décrire étant donné les coordonnées des points $\mu_i, \mu_{i'}$. On pourra representer un demi-plan par un élément (a, b, c) de $\mathbb{R}^3$ où $h = \{(x, y) \in \mathbb{R}^2 : ax + by \leq c\}$ (on ne doit pas avoir $a = b = 0$). Nous avons donc directement une représentation de V_i en terme d'une intersection de demi-plans (5). En utilisant cette représentation, on peut facilement déterminer si un point est dans V_i ou pas, il suffit de tester que le point se trouve dans chacun des demi-plans. Notons que V_i est un polygone convexe. Pour pouvoir colorier la cellule V_i, on va transformer cette représentation en une représentation plus adaptée : une liste ordonnée de segments du polygone, ou encore une liste ordonnée de demi-plans.

2.3 Fonctionalités de votre programme

La première partie du projet est la même pour tous : C'est une interface graphique qui permet d'afficher des points ainsi que des centres, appliquer l'algorithme des k-moyennes et un algorithme pour le calcul du diagramme de Voronoi des centres. Votre programme doit avoir les fonctionalités suivantes.

1. Votre fenêtre devra contenir un panneau qui sert à afficher des points, et quelques boutons de contrôle
2. Un bouton qui permet de charger et d'afficher une liste de points à partir d'un ficher texte
3. Un textfield qui permet de prendre en entrée la valeur de k
4. Un bouton qui permet d'initialiser les k centres choisis aléatoirement parmi les points existants. Penser à afficher les centres de manière un peu différente des points qui représentent les données
5. Un bouton qui permet de faire tourner l'algorithme des k-moyennes pour la valeur de k spécifiée dans le textfield
6. Un bouton qui permet de tracer le diagramme de Voronoi des k centres, en coloriant chaque case de Voronoi d'une couleur différente. Les données initiales doivent rester visible quand même. Vous pourrez utiliser l'algorithme qui est décrit dans le document en annexe, ou tout autre algorithme plus efficace.

Important : Pour le samedi 13 décembre, vous devrez rendre un programme qui réalise toutes les fonctionalités ci-dessus excepté la dernière. Ceci constitue le TP3, merci d'inclure `[Proj1] [TP3]` dans le sujet du mail.

2.3.1 Interface graphique en Java

Nous allons utiliser le package `javax.swing` pour créer des fenêtres, boutons, etc... Vous pourrez utiliser les ressources suivantes :

— http://fr.openclassrooms.com/informatique/cours/apprenez-a-programmer-en-java/notre-premiere-fenetre
— http://docs.oracle.com/javase/tutorial/uiswing/index.html

Les classes les plus importantes sont : `JFrame` pour une fenêtre, `JPanel` pour un panneau d'affichage (on peut avoir plusieurs panneaux dans un fenêtre), `JButton` pour un bouton et `JTextField` pour prendre du texte en entrée.
Pour détecter les événements qui surviennent dans un composant, on utilise un `ActionListener`. On rajoute un objet de type `ActionListener` à un composant en utilisant la fonction `addActionListener(listener)`, où listener doit être un objet qui implémente l'interface `ActionListener`. Cet objet peut par exemple être le composant lui même à condition que l'on définisse une fonction `actionPerformed(ActionEvent e)`. Une interface est une classe dont les méthodes ne sont pas implémentées. Pour implémenter une interface on utilise le mot clé `implements`.

2.3.2 Algorithme de calcul du diagramme de Voronoi

Il y a plusieurs algorithmes possibles pour le calcul du diagramme de Voronoi. Pour la première partie du projet, n'importe quel algorithme qui fonctionne en temps polynomial convient. Voici une piste pour un algorithme simple qui fonctionne en temps $O(k^3)$ où k est le nombre de points.

1. Définir une structure de donnée pour un demi plan, et une fonction qui calcule les demi-plan $h(i, j)$ pour tous points i, j.
2. Définir une fonction qui prend en entrée un polygone convexe (comme une liste ordonnée de segments ou de demi-plans) et un demi-plan et renvoie l'intersection dans le même format (en particulier, la liste doit rester ordonnée).

3. Pour déterminer la cellule correspondant au point i, il suffit de commencer avec le rectangle des limites du panneau par exemple, et de faire l'intersection pas à pas de chaque demi plan $h(i, j)$ pour $j \neq i$. Pour ce faire, on pourra définir une fonction qui prend en entrée un polygone convexe (comme une liste ordonnée de segments ou de demi-plans) et un demi-plan et renvoie l'intersection dans le même format (en particulier, la liste doit rester ordonnée).

Important : N'oubliez pas de tester vos fonctions au fur et à mesure. N'attendez pas d'avoir implémenter toutes les étapes pour finalement vous rendre compte qui votre programme n'affiche pas un diagramme de Voronoi.

3 Seconde partie

Vous devrez rajouter une fonctionnalité de votre choix à votre programme (à discuter avec le responsable). Voici quelques suggestions :

1. Outil pour permettre de générer un ensemble de points (par exemple clics de souris de l'utilisateur, aléatoirement selon une distribution donnée, etc...)
2. L'initialisation des centres peut beaucoup influencer la classification. Proposer d'autre méthodes pour prendre en entree la position des centres (par exemple clic de souris, fichier texte, etc...) Ou encore proposer une méthode pour une initialisation automatique qui soit plus efficace qu'un choix aléatoire.
3. L'algorithme pour le calcul du diagramme de Voronoi décrit plus haut $O(k^3)$. Il existe plusieurs algorithmes en temps $O(k \log k)$. Implémenter un de ces algorithmes.
4. Supposons que l'on ne connaisse pas le nombre de groupe dans nos données. Proposer une méthode pour essayer de deviner un nombre de clusters. Voir [1].
5. Proposer d'autres algorithmes pour le même problème et faire une comparaison dans votre rapport (par exemple changer de distance, ou utiliser un algorithme de clustering spectral [4]).
6. Pour la grande majorité des applications, on travaille en dimension $d \gg 2$. Par exemple, une image peut être représentée par un tableau contenant la valeur de chaque pixel. Un texte peut être représenté par une liste de mots avec leur fréquence dans le texte (on parle de "bag-of-words model"). Vous pourrez écrire un programme qui trouve des clusters pour un de ces types de données. Pour des textes, votre algorithme devra pouvoir classer des textes en fonction de la langue par exemple (assez facile) mais peut être aussi l'auteur, le sujet. Pour des images, on pourra soit regrouper des ensembles d'images soit regrouper des pixels au sein d'une même image (segmentation d'image, voir [3]). Une possibilité ici est d'implémenter un algorithme de réduction de dimension, par exemple analyse en composantes principales, et d'utiliser votre programme pour $d = 2$. Essayer de réutiliser votre code au maximum en utilisant la programmation orientée objet. Mais vous pourrez aussi optimiser votre programme pour votre type de données.

Références

[1] Wikipedia article on determining the number of clusters.

[2] Mark de Berg, Marc van Kreveld, Mark Overmars, and Otfried Schwarzkopf. *Computational geometry*. Springer Berlin Heidelberg, 1999.

[3] Christopher M. Bishop, *Pattern Recognition and Machine Learning*. Springer New York, 2006.

[4] Andrew Y. Ng, Michael I. Jordan, and Yair Weiss. "On spectral clustering : Analysis and an algorithm." Advances in neural information processing systems 2 (2002) : 849-856.