CentraleSupélec

Big Data : Informatique pour les données et calculs massifs

13 – Algorithmes de *clustering*

Stéphane Vialle

CentraleSupélec université PARIS-SACLAY – École doctorale Sciences et technologies de l'information et de la communication (STIC) – RISEGrid – Grand Est

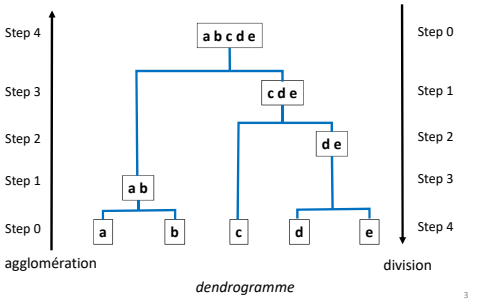
Stephane.Vialle@centralesupelec.fr
http://www.metz.supelec.fr/~vialle

Algorithmes de *clustering*

1. **Clustering hiérarchique**
2. Clustering non-hiérarchique par partitionnement
3. Clustering non-hiérarchique large échelle
4. Clustering hiérarchique non-sphérique et large échelle
5. Autres solutions

Clustering hiérarchique

Clustering ascendant — Clustering descendant

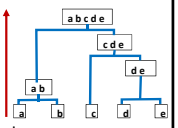


dendrogramme

Clustering hiérarchique

Clustering ascendant (on veut *k* clusters)

Etat initial : *n* clusters de *1* donnée chacun



A - 1ère étape

- Calcul des distances entre chaque couple de clusters (*n.(n-1)/2* distances)
- Identification de la distance d(C_i, C_j) minimale
- Regroupement des clusters C_i etC_j

B - Etapes suivantes

- Calcul des distances entre le nouveau cluster et les autres clusters (les autres distances restent inchangées)
- Identification de la distance minimale
- Regroupement des clusters associés

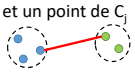
Si nbr de clusters > k : retour au point B
Sinon : fin

Clustering hiérarchique

Clustering ascendant (on veut *k* clusters)

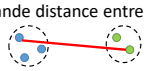
Difficultés :

1. **Définir une distance entre deux clusters**
   - Plus petite distance entre un point de C_i et un point de C_j



→ *Tend à construire des arbres aplatis : composés de chaînes de clusters*

   - Plus grande distance entre un point de C_i et un point de C_j



→ *Tend à construire des clusters bien séparés*

   - Distance moyenne entre les point de C_i et les points de C_j

Clustering hiérarchique

Clustering ascendant (on veut *k* clusters)

Difficultés :

2. **Combien de clusters créer ?**
   Quel est le nombre de clusters le plus pertinent ?
   - Le nbr de clusters peut être imposé « coute que coute »
   - **Sinon on peut :**
     - Appliquer un critère de distance maximale entre clusters pour autoriser une agrégation
     - Pratiquer une analyse visuelle de l'arbre d'agglomération (ex : rechercher un arbre équilibré…)
     - Rechercher une solution qui à du sens métier : s'appuyer sur la signification des données et des regroupements

Algorithmes de *clustering*

1. Clustering hiérarchique
2. **Clustering non-hiérarchique par partitionnement**
   - **K-Means**
   - K-Medoids
3. Clustering non-hiérarchique large échelle
4. Clustering hiérarchique non-sphérique et large échelle
5. Autres solutions

7

Clustering par K-Means

Principe de l'algorithme des K-Means

- partitionner un ensemble de points *S* en *k* sous-ensembles $\{S_1, S_2, \ldots S_k\}$
- en minimisant la distance entre les points à l'intérieur de chaque partition

→ On cherche la partition de *S* en *k* sous-ensembles qui minimise globalement :

$$\sum_{i=1}^{k} \sum_{x_j \in S_i} \|x_j - m_i\|^2$$

où m_i est le barycentre de la partition S_i

Dans le cas général : pb NP complet…

→ trop couteux de rechercher le minimum exact

→ on applique une **heuristique** qui recherche une solution approchée en temps raisonnable

8

Clustering par K-Means

Heuristique des K-Means

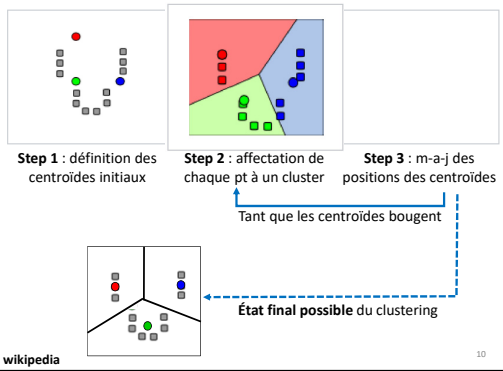
1. Placer aléatoirement *k* points dans l'espace des données
   → les *centroïdes* initiaux des *k* clusters
2. Calculer la distance de chaque point à chaque *centroïde*, et affecter chaque point au cluster dont il est le plus proche
3. Quand tous les points ont été affectés, recalculer les positions des *k centroïdes*
4. Répéter les étapes 2 et *3 jusqu'à ce que les centroïdes ne bougent plus*.

couteux !

- Calculer alors la métrique à optimiser/minimiser
  Ex : la somme des carrés des distances de chaque point au centre de son cluster : $\sum_{i=1}^{k} \sum_{x_j \in S_i} \|x_j - m_i\|^2$

9

Clustering par K-Means



Src : wikipedia

10

Clustering par K-Means

Heuristique des K-Means

Robustesse de l'heuristique :

- On peut prouver que l'heuristique termine toujours, si on utilise une distance Euclidienne.

Sensibilité de l'heuristique :

- Sensibilité au choix des *k* centroïdes initiaux
  Différentes configurations de départ peuvent mener à des clustering différents
  → Tenter de prendre de bons points de départ…
- Sensibilité au choix de la métrique pour exprimer $\|x_j - m_i\|$
  La distance euclidienne est souvent utilisée, mais d'autres distances vont donner d'autres clusters
  → Choisir une distance adaptée au problème…

11

Clustering par K-Means

Démarches possibles pour l'étape initiale :

- ***Forgy method*** :
  - choisir aléatoirement k points dans S, qui deviennent les k centroïdes initiaux
- ***Random Partition method*** :
  - affecter aléatoirement chaque point de *S* à un des *k* clusters,
  - puis reprendre au point 3 : calculer les nouveaux centroïdes à partir des points de chaque cluster

12

Clustering par K-Means

Démarches possibles pour l'étape initiale :

- ***Bradley & Fayyad & Cori approach (Microsoft) :***
  → **pour de GROS ensembles de données**
  - appliquer l'algorithme sur ***q* sous problèmes échantillonnés** aléatoirement *(q petits pbs, à calculs rapides)*
  - pour chacun des *q* clustering obtenu : calculer la métrique d'évaluation sur l'ensemble des données :
    - Partitionner l'ensemble des données par rapport aux *k* centroïds
    - Calculer la métrique sur le partitionnement complet
  - retenir le meilleur clustering comme configuration **initiale**

Rmq : Sur de gros ensembles, si on échantillonne à 1%, cette pré-étape initiale est négligeable

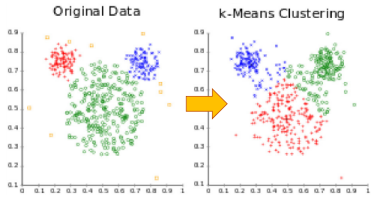
Clustering par K-Means

Points faibles du clustering par K-means

Le K-means fonctionne bien si les **clusters** à identifier sont **hypersphériques** et de **tailles similaires**

Problèmes dans les autres cas :

- Ex. si les clusters sont hypersphériques mais de tailles différentes :



Src : *Wikipedia*

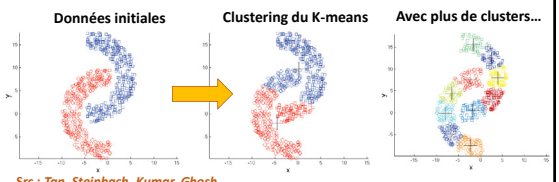
Clustering par K-Means

Points faibles du clustering par K-means

Le K-means fonctionne bien si les **clusters** à identifier sont **hypersphériques** et de **tailles similaires**

Problèmes dans les autres cas :

- Ex. si les clusters ne sont **pas** hypersphériques :



Src : *Tan, Steinbach, Kumar, Ghosh*

Clustering par K-Means

Points faibles du clustering par K-means

Le K-means est **sensible aux points aberrants** car :
il calcule les centroïdes des clusters par la moyenne de leurs points (et la moyenne est sensible aux points aberrants)

Le K-means **reste couteux car :**

- Le pb exact est NP-complet
- l'heuristique classique continue à calculer *n.k* distances à chaque itération !

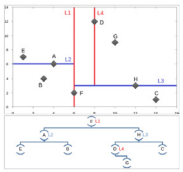
Clustering par K-Means

Stratégies d'optimisation en temps de calcul

- **Elagage du nombre de calculs à réaliser par "kd-tree" :**
  - structure de données arborescente (le *kd-tree*)
  - qui range les points en réalisant un découpage géométrique de l'espace

  → rapide à interroger

  → permet d'élaguer les calculs en éliminant les points trop éloignés du point en cours de traitement pour être des voisins intéressants…



Arbre 2-d correspondant à la partition de l'espace (à dimension 2) au-dessus.

Src : *Wikipedia*

Très efficace avec un petit et moyen nombre de dimensions…

Clustering par K-Means

Stratégies d'optimisation en temps de calcul

- **Elagage du nombre de calculs à réaliser par :**
  - utilisation de propriétés sur les distances
    ex : $d(a,b) \leq d(a,c) + d(c,b)$
  - stockage de bornes inférieurs et supérieures, de diverses distances…
    ex : d_{min}(centroide,centroide), d_{max}(pt,cluster)…

Plus efficace que les kd-trees quand le nombre de dimensions augmente…

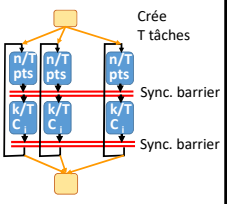
Clustering par K-Means

Stratégies d'optimisation en temps de calcul

- **Parallélisation des calculs**

1. Placer aléatoirement *k* points dans l'espace des données
   → les *centroïdes* initiaux des *k* clusters
2. Calculer la distance de chaque point à chaque *centroïde*, et affecter chaque point au cluster dont il est le plus proche
3. Quand tous les points ont été affectés, recalculer les positions des *k centroïdes*
4. Répéter les étapes 2 et *3 jusqu'à ce que les centroïdes ne bougent plus.*

Parallélisation « SPMD »



Rmq : parallélisation simple,
mais beaucoup d'accès mémoires concurrents (pb « sans localité spatiale des données ») : src de perte d'accélération…

Clustering par K-Means

Stratégies d'optimisation en temps de cacul

- **Réduction des calculs par échantillonnage (*sampling*) des données à classer**

1. Travail sur un ou plusieurs sous-problèmes pour trouver une bonne initialisation au problème complet

Bradley, Fayyad, Cori approach

2. Travail complet sur un sous-problème, avec un échantillon **bien représentatif**
   - Solution de qualité presque identique
   - Temps de traitement fortement réduit
   - Attention à vérifier que la solution est de bonne qualité
     → plusieurs échantillonnages et plusieurs traitements

Clustering par K-Means

Stratégies d'optimisation de la qualité du clustering

- **Considérer *k* (le nbr de clusters) comme un paramètre d'entrée du problème**

1. Réaliser un clustering complet avec k_0 clusters
2. Evaluer la qualité globale
   - par une métrique de la similarité intra-cluster : somme de distances à minimiser
   - et/ou par une métrique de dissimilarité inter-clusters : autre somme de distance à maximiser
3. Recommencer en 1 avec une autre valeur de *k*, jusqu'à essai de suffisamment de valeurs de *k*
4. Conserver le clustering ayant la meilleure qualité globale (ou semblant le meilleur du point de vue métier)

→ Très couteux !
→ Mais hautement parallélisable

Algorithmes de *clustering*

1. Clustering hiérarchique
2. **Clustering non-hiérarchique par partitionnement**
   - K-Means
   - **K-Medoids**
3. Clustering non-hiérarchique large échelle
4. Clustering hiérarchique non-sphérique et large échelle
5. Autres solutions

Clustering par K-Medoids

Principes et avantages

- Même approche que celle du K-means, mais les centres des clusters (les *medoids*) sont pris parmi les points à classer

- Fonctionne avec une mesure de dissimilarité : norme L1, L2 ou … (n'exige pas une distance Euclidienne)
- Moins sensible aux valeurs aberrantes : on ne calcule pas de moyennes (qui sont sensibles aux valeurs aberrantes).
- **Toutes les *n.(n-1)* distances possibles entre points peuvent être pré-calculées** sur CPU définitivement au début (et en parallèle)

Rmq : dans le cas d'implantations sur GPU il peut être plus efficace de refaire des calculs que de stocker et relire trop de données…

Clustering par K-Medoids

Algo PAM : *Partitioning Around Medoids*

- Kaufman & Rousseeuw, 1990 (Vrije Univ. & Antwerp Univ, Belgique)
- **2 variantes** existent dans la littérature

Algo PAM – V1 : remplacement de medoids sans contrainte

1. Sélectionner aléatoirement *k* points pour être les Medoids (formation des centres des clusters initiaux)
2. Associer chaque point au Medoid dont il est le plus proche (formation des clusters initiaux)
3. Calculer la dissimilarité globale du système :

$$DG = \sum_{i=1}^{k} \sum_{x \in C_i} d(x, m_i)$$

avec $d(x, m_i)$ calculé à partir de la norme L1, ou L2, ou …

Clustering par K-Medoids

Algo PAM : *Partitioning Around Medoids*
- Kaufman & Rousseeuw, 1990 (Vrije Univ. & Antwerp Univ, Belgique)
- **2 variantes** existent dans la littérature

Algo PAM – V1 : remplacement de medoids sans contrainte

4. **Répeter :**
   5. Pour chaque medoid m_i
      6. Pour chaque point non-medoid O
         - Echanger m_i et O (O devient le medoid du cluster C_i)
         - Réaffecter chaque point non-medoid à son medoid le plus proche dans tous le système
         - Recalculer la dissimilarité globale (DG)
         - Si DG à chuté : valider la permutation de m_i et O
           Sinon : annuler cette permutation

Tant que : la dissimilarité globale (DG) décroit

25

Clustering par K-Medoids

Algo PAM : *Partitioning Around Medoids*
- Kaufman & Rousseeuw, 1990 (Vrije Univ. & Antwerp Univ, Belgique)
- **2 variantes** existent dans la littérature

Algo PAM – V1 : remplacement de medoids sans contrainte

4. **Répeter :**
   5. Pour chaque medoid m_i
      6. Pour chaque point non-medoid O
         - Echanger m_i et O (O devient le medoid du cluster C_i)
         - **Étape parallélisable** Réaffecter chaque point non-medoid à son medoid le plus proche dans tous le système
         - **Étape parallélisable** Recalculer la dissimilarité globale (DG)
         - Si DG à chuté : valider la permutation de m_i et O
           Sinon : annuler cette permutation

Tant que : la dissimilarité globale (DG) décroit

26

Clustering par K-Medoids

Algo PAM : *Partitioning Around Medoids*
- Kaufman & Rousseeuw, 1990 (Vrije Univ. & Antwerp Univ, Belgique)
- **2 variantes** existent dans la littérature

Algo PAM – V1 : remplacement de medoids sans contrainte

4. **Répeter :**
   5. Pour chaque medoid m_i
      6. Pour chaque point non-medoid O
         - Echanger m_i et O (O devient le medoid du cluster C_i)
         - **Étape parallélisable** Réaffecter chaque point non-medoid à son medoid le plus proche dans tous le système
         - **Étape parallélisable** Recalculer la dissimilarité globale (DG)
         - Si DG à chuté : valider la permutation de m_i et O
           Sinon : annuler cette permutation

Tant que : la dissimilarité globale (DG) décroit

27

Clustering par K-Medoids

Algo PAM : *Partitioning Around Medoids*
- Kaufman & Rousseeuw, 1990 (Vrije Univ. & Antwerp Univ, Belgique)
- **2 variantes** existent dans la littérature

Algo PAM – V1 : remplacement de medoids sans contrainte

4. **Répeter :**
   5. Pour chaque medoid m_i
      6. Pour chaque point non-medoid O

Bloque le parallélisme supérieur : boucles 5 et 6 à faire dans l'ordre

         - Echanger m_i et O (O devient le medoid du cluster C_i)
         - **Étape parallélisable** Réaffecter chaque point non-medoid à son medoid le plus proche dans tous le système
         - **Étape parallélisable** Recalculer la dissimilarité globale (DG)
         - Si DG à chuté : valider la permutation de m_i et O
           Sinon : annuler cette permutation

Tant que : la dissimilarité globale (DG) décroit

28

Clustering par K-Medoids

Algo PAM – V2 : remplacement de medoids au sein de leurs clusters

1. Sélectionner aléatoirement k points pour être les Medoids
2. Associer chaque point au Medoid dont il est le plus proche
3. Calculer la dissimilarité de chaque cluster: $\{\sum_{x \in C_i} d(x, m_i)\}_{1 \le i \le k}$

4. **Répeter :**
   5. Pour chaque cluster C_i :
      6. Pour chaque point non-medoid O dans C_i :
         - Echanger m_i et O (O devient le medoid du cluster C_i)
         - Calculer la dissimilarité du cluster C_i
      7. Remplacer m_i par le point O_h qui a produit la plus petite dissimilarité du cluster C_i
      8. Réaffecter chaque point non-medoid à son medoid le plus proche dans tous le système

Tant que : une dissimilarité de cluster décroit

29

Clustering par K-Medoids

Algo PAM – V2 : remplacement de medoids au sein de leurs clusters

1. Sélectionner aléatoirement k points pour être les Medoids
2. Associer chaque point au Medoid dont il est le plus proche
3. Calculer la dissimilarité de chaque cluster: $\{\sum_{x \in C_i} d(x, m_i)\}_{1 \le i \le k}$

4. **Répeter :**
   5. Pour chaque cluster C_i :
      Étape 6 parallélisable 6. Pour chaque point non-medoid O dans C_i :
         - Echanger m_i et O (O devient le medoid du cluster C_i)
         - Calculer la dissimilarité du cluster C_i
      7. Remplacer m_i par le point O_h qui a produit la plus petite dissimilarité du cluster C_i
      Étape 8 parallélisable 8. Réaffecter chaque point non-medoid à son medoid le plus proche dans tous le système

Tant que : une dissimilarité de cluster décroit

30

Clustering par K-Medoids

Algo PAM – V2 : remplacement de medoids au sein de leurs clusters

1. Sélectionner aléatoirement k points pour être les Medoids
2. Associer chaque point au Medoid dont il est le plus proche
3. Calculer la dissimilarité de chaque cluster: $\{\sum_{x \in C_i} d(x, m_i)\}_{1 \le i \le k}$

4. Répeter :

   5. Pour chaque cluster C_i : **Boucle 5 à faire dans l'ordre**

   Étape 6 parallélisable

   6. Pour chaque point non-medoid O dans C_i :
      - Echanger m_i et O (O devient le medoid du cluster C_i)
      - Calculer la dissimilarité du cluster C_i
   7. Remplacer m_i par le point O_h qui a produit la plus petite dissimilarité du cluster C_i

   Étape 8 parallélisable

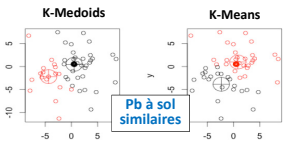
   8. Réaffecter chaque point non-medoid à son medoid le plus proche dans tous le système

Tant que : une dissimilarité de cluster décroit

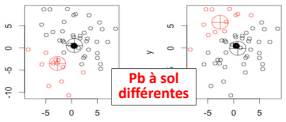
Clustering par K-Medoids

Exemple de classification K-Means et K-Medoïds

- Sur certains pbs : résultats proches de ceux du K-means,
- Sur d'autres pbs : résultats différents !



- **Même pb de sensibilité à l'initialisation des centres des K clusters** : faire plusieurs exécutions pour étudier la stabilité/variance du clustering….



Clustering par K-Medoids

Exemple de critère de bonne classification d'un point (*silhouette*)

Pour chaque point/donnée d'entrée on calcule sa *silhouette* : un critère qui mixe :

- la faible dissimilarité du point avec les autres points de son cluster,
- la forte dissimilarité du point avec les points des autres clusters

- **Une silhouette** : un critère unique pour décider si le point est bien ou mal classifié
- **La moyenne des silhouettes** : un critère unique pour décider si le clustering a mis en évidence une structure profonde dans les données, ou s'il faut changer l'algorithme ou le nombre de clusters.

Clustering par K-Medoids

Calcul de la *silhouette* d'un point

On considère le point *i* (la donnée d'entrée *i*)

1. Calcul de : *a* = la dissimilarité de *i* dans son cluster
   (sa dissimilarité vis-à-vis de tous les autres points de son cluster)
2. Calcul de : *b* = la dissimilarité de *i* vis-à-vis des autres clusters
   En fait : *b* = la dissimilarité de *i* vis-à-vis des points du cluster autre que le sien dont elle est le plus proche (son 2ème plus proche cluster)
3. Calcul de : s = la silhouette de *i*
   - **Si le cluster de i ne contient que i : s = 0**
   - **Si a < b : s = 1 – a/b**
   - **Si a > b : s = b/a – 1**
   - **Si a = b : s = 0**

Clustering par K-Medoids

Interprétation de la *silhouette* d'un point

- **s proche de 1 : le point est bien classifié**
  Plus de dissimilarité avec les autres clusters qu'avec son cluster
- **s proche de -1 : le point est mal classifié**
  Plus de dissimilarité avec son cluster qu'avec un autre cluster

Interprétation de la moyenne des silhouettes

- $\bar{s}$ indique si le clustering réalisé correspond bien à la *structure profonde* des données
- On peut calculer $\bar{s}$ pour différentes valeurs du nombre de clusters *k*, et identifier un nombre de clusters maximisant $\bar{s}$

Exemple : $0.71 \le \bar{s} \le 1.0$: une structure profonde a été identifiée
$0.51 \le \bar{s} \le 0.7$: une structure raisonnable à été identifiée
Sinon … structure faible ou pas de structure identifiée

Algorithmes de *clustering*

1. Clustering hiérarchique
2. Clustering non-hiérarchique par partitionnement
3. **Clustering non-hiérarchique large échelle**
   - **CLARA**
   - CLARANS
4. Clustering hiérarchique non-sphérique et large échelle
5. Autres solutions

CLARA : Clustering LARge Applications

Objectifs

Traiter des ensembles de données de **grandes tailles**

Principes

Echantillonner les données et travailler sur des sous-ensembles

1. Créer plusieurs sous-ensembles par échantillonnage
2. Exécuter un algo. PAM (classique) sur chaque sous-ensemble
3. Evaluer chaque clustering sur l'ensemble complet des données
4. Conserver la meilleure solution

Rmq : Ici on ne se sert pas de la meilleure solution pour initialiser une méthode sur les données complètes

Rmq : En pratique les différents sous-ensembles ne seront pas complètement indépendants (voir plus loin)

37

CLARA : Clustering LARge Applications

Algorithme CLARA

Pour i = 1 à 5 :

1. Si i == 1 : sélectionner aléatoirement *40+2.k* données
   Sinon : échantillonner aléatoirement *40+k* données et y ajouter les meilleurs *k* medoids trouvés précédemment
2. Exécuter l'algorithme PAM sur le sous-ensemble pour trouver *k* medoids — Calcul sur échantillon
3. Associer chaque point de l'ensemble complet des données au médoid dont il est le plus proche — Eval sur ensemble complet
4. Calculer la dissimilarité moyenne de l'ensemble complet des *n* données : $\sum_{i=1}^{k} \sum_{x \in C_i} \frac{d(x,m_i)}{n}$
5. Si cette dissimilarité est plus faible que la meilleure trouvée: considérer cette solution et ses *k* medoids comme la meilleure solution courante

38

CLARA : Clustering LARge Applications

Algorithme CLARA

Pour i = 1 à 5 : **Boucle sur i à faire dans l'ordre**

1. Si i == 1 : sélectionner aléatoirement *40+2.k* données
   Sinon : échantillonner aléatoirement *40+k* données et y ajouter les meilleurs *k* medoids trouvés précédemment
2. **Étape 2 parallélisable** Exécuter l'algorithme PAM sur le sous-ensemble pour trouver *k* medoids
3. **Étape 3 parallélisable** Associer chaque point de l'ensemble complet des données au médoid dont il est le plus proche
4. **Étape 4 parallélisable** Calculer la dissimilarité moyenne de l'ensemble complet des *n* données : $\sum_{i=1}^{k} \sum_{x \in C_i} \frac{d(x,m_i)}{n}$
5. Si cette dissimilarité est plus faible que la meilleure trouvée: considérer cette solution et ses *k* medoids comme la meilleure solution courante

39

CLARA : Clustering LARge Applications

Bilan de l'algorithme CLARA

- PAM : $O(k.(n-k)^2)$
  CLARA : $O(k.(40+k)^2 + k.(n-k))$: **beaucoup mieux !!**
- **Mais un échantillonnage sévère peut fausser le clustering**
  → Réduire l'impact de l'échantillonage → « CLARANS »

40

Algorithmes de *clustering*

1. Clustering hiérarchique
2. Clustering non-hiérarchique par partitionnement
3. **Clustering non-hiérarchique large échelle**
   - CLARA
   - **CLARANS**
4. Clustering hiérarchique non-sphérique et large échelle
5. Autres solutions

41

CLARANS : Clustering Large Application based on RANdomized Search

Principe

Comme pour CLARA :

- On expérimente X configurations aléatoires,
- qu'on évalue par une dissimilarité moyenne sur l'ensemble complet des données,
- et on garde la meilleure des X solutions

MAIS :

- Chaque configuration aléatoire est seulement un pt de départ,
- et on cherche à l'améliorer en remplaçant un médoïde par un point non-medoïde pris dans l'ensemble complet des données (pas seulement dans un sous-ensemble échantillonné)

La recherche d'une meilleure solution n'est pas limitée à un sous-ensemble échantillonné, ce-dernier n'est qu'un point de départ de l'exploration.

42

CLARANS : Clustering Large Application based on RANdomized Search

Graphe des configurations

Notre problème se ramène à :

choisir *k* médoïdes dans un graphe de *n* points, où chaque point non médoïde est rattaché au médoïde dont il est le plus proche.

- Soit $G_{n,k}$ le graphe de toutes les configurations possibles, où deux nœuds sont voisins si leurs ensembles de *k* médoïdes ne diffèrent que d'un point (c-a-d s'ils ont *k-1* médoïdes communs)
- Chaque nœud du graphe possède *k.(n-k)* voisins
- Une configuration aléatoire constitue un nœud de ce graphe

→ Un des nœuds du graphe conduit à la mesure de dissimilarité minimale : $\min(\sum_{i=1}^{k} \sum_{x \in C_i} \frac{d(x,m_i)}{n})$ …. mais lequel ??

43

CLARANS : Clustering Large Application based on RANdomized Search

Algorithme CLARANS

Pour *i = 1* à *NbExplorations*

1. Choisir aléatoirement une configuration de *k* medoids (choix aléatoire d'un nœud de départ dans $G_{n,k}$)
2. Calculer sa mesure de dissimilarité sur l'ensemble complet des n données d'entrée
3. *nv = 1*, démarrer une exploration à partir de la config courante
4. Remplacer aléatoirement un medoid avec un pt non-medoid (choix aléatoire d'un voisin *V* du Nœud Courant dans $G_{n,k}$)
5. Evaluer sa dissimilarité sur l'ensemble complet des données d'entrée
6. Si la dissimilarité à diminué : valider le remplacement et
   repartir au point 3 (Nœud Courant = V)
   Sinon : *nv = nv+1*
   Si *nv* < ***MaxVoisinsExplorés*** :
   retour au point 4
7. Remplacer Meilleure Config par le Nœud Courant si meilleure dissimilarité

44

CLARANS : Clustering Large Application based on RANdomized Search

Algorithme CLARANS

Pour *i = 1* à *NbExplorations* **Boucle sur i parallélisable**

1. Choisir aléatoirement une configuration de *k* medoids (choix aléatoire d'un nœud de départ dans $G_{n,k}$)
2. Calculer sa mesure de dissimilarité sur l'ensemble complet des n données d'entrée
3. *nv = 1*, démarrer une exploration à partir de la config courante
4. Remplacer aléatoirement un medoid avec un pt non-medoid (choix aléatoire d'un voisin *V* du Nœud Courant dans $G_{n,k}$)
5. Evaluer sa dissimilarité sur l'ensemble complet des données d'entrée
6. Si la dissimilarité à diminué : valider le remplacement et
   repartir au point 3 (Nœud Courant = V)
   Sinon : *nv = nv+1*
   Si *nv* < ***MaxVoisinsExplorés*** :
   retour au point 4
7. **Sauver la config du Nœud Courant et sa dissimilarité**

Identifier et retenir la meilleure config des NbExplorations

45

CLARANS : Clustering Large Application based on RANdomized Search

Performances expérimentales de CLARANS

- Beaucoup plus rapide que PAM
- Plus rapide ou plus lent que CLARA !!
- Mais réalise un clustering de meilleure qualité (moins de dissimilarité) que CLARA

Rmq : expérimentations jusqu'à 100 points et 5 clusters….
… expérimentations pas très « large scale » !

Rmq : l'échantillonnage est une option intéressante mais qui mènent à de nombreuses variantes!

46

Algorithmes de *clustering*

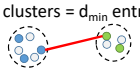
1. Clustering hiérarchique
2. Clustering non-hiérarchique par partitionnement
3. Clustering non-hiérarchique large échelle
4. **Clustering hiérarchique non-sphérique et large échelle**
5. Autres solutions

47

CURE : Clustering Using REpresentatives

Une solution qui mixe de nombreux concepts :

- Clustering hiérarchique ascendant
- Utilisation de structures de données en kd-tree et en pile pour stocker et gérer les données et les clusters
- Utilisation d'un sous ensemble des points de chaque cluster, mais **choisis** pour bien couvrir le cluster : les « *representatives* »

  → Distance entre clusters = d_{min} entre *representatives* de chaque cluster 
- Echantillonnage des données d'entrée, pour le « large échelle »
- Partitionnement des données et clustering en deux passes (clustering de clusters) pour accélérer les calculs

48

CURE : Clustering Using REpresentatives

Objectifs

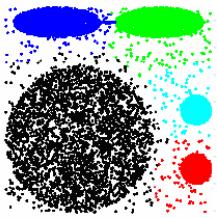
- Identifier des clusters non hyper-sphériques, grâce à :
  - une représentation des clusters par un sous-ensembles de points couvrant bien tout le cluster,
  - un mélange de clustering hiérarchique et de calcul de distance minimale entre *representatives* de deux clusters
- Des traitements rapides :
  - grâce aux structures en kd-tree de points et pile de clusters,
  - et en représentant les clusters par quelques points seulement (échantillonnage choisi)
- Des traitements de données large échelle, grâce à :
  - un échantillonnage aléatoire des données d'entrée,
  - un clustering en 2 passes sur un partitionnement des données

49

CURE : Clustering Using REpresentatives

CURE arrive à clusteriser des ensembles de points complexes

- Non-spheriques
- De rayons différents
- Pouvant se toucher
- Avec des points isolés entre les formes principales
- Avec beaucoup de points
- Rapidement



src : Guha, Rastogi, Shim

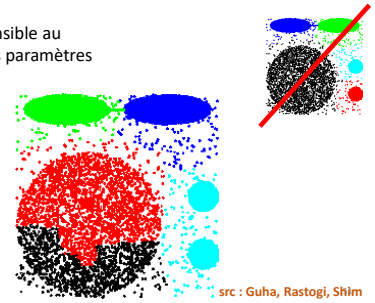
50

CURE : Clustering Using REpresentatives

CURE arrive à clusteriser des ensembles de points complexes

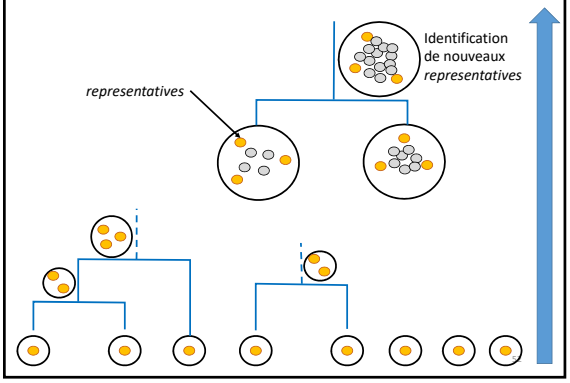
- Mais reste sensible au réglage de ses paramètres

Influence du paramètre α (recentrage des *repres-entatives*) :



src : Guha, Rastogi, Shim

CURE : Clustering Using REpresentatives



CURE : Clustering Using REpresentatives

Routines principales de CURE

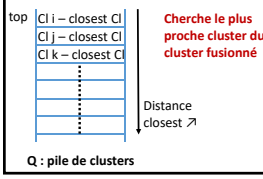
Identifie les *c* points les plus éloignées les uns des autres dans un cluster
→ Les *c representatives* du cluster

```
procedure merge(u, v)
begin
1.  w := u ∪ v
2.  w.mean := (|u|·u.mean + |v|·v.mean) / (|u| + |v|)
3.  tmpSet := ∅
4.  for i := 1 to c do {
5.      maxDist := 0
6.      foreach point p in cluster w do {
7.          if i = 1
8.              minDist := dist(p, w.mean)
9.          else
10.             minDist := min{dist(p, q) : q ∈ tmpSet}
11.         if (minDist ≥ maxDist){
12.             maxDist := minDist
13.             maxPoint := p
14.         }
15.     }
16.     tmpSet := tmpSet ∪ {maxPoint}
17. }
18. foreach point p in tmpSet do
19.     w.rep := w.rep ∪ {p + α*(w.mean-p) }
20. return w
end
```

CURE : Clustering Using REpresentatives

Routines principales de CURE

Vide et remplit Q, jusqu'à ce qu'elle ne contienne que *k* clusters




```
procedure cluster(S, k)
begin
1.  T := build_kd_tree(S)
2.  Q := build_heap(S)
3.  while size(Q) > k do {
4.      u := extract_min(Q)
5.      v := u.closest
6.      delete(Q, v)
7.      w := merge(u, v)
8.      delete_rep(T, u); delete_rep(T, v); insert_rep(T, w)
9.      w.closest := x /* x is an arbitrary cluster in Q */
10.     for each x ∈ Q do {
11.         if dist(w, x) < dist(w, w.closest)
12.             w.closest := x
13.         if x.closest is either u or v {
14.             if dist(x, x.closest) < dist(x, w)
15.                 x.closest := closest_cluster(T, x, dist(x, w))
16.             else
17.                 x.closest := w
18.             relocate(Q, x)
19.         }
20.         else if dist(x, x.closest) > dist(x, w) {
21.             x.closest := w
22.             relocate(Q, x)
23.         }
24.     }
25.     insert(Q, w)
26. }
end
```

Extrait les 2 plus proches clusters

Fusionne les 2 clusters

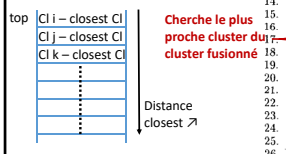
Insère le cluster fusionné dans la pile des clusters

CURE : Clustering Using REpresentatives

Routines principales de CURE

Vide et remplit Q, jusqu'à ce qu'elle ne contienne que *k* clusters

Rien de parallélisable ou avec bcp d'opérations de synchronisation !!!



Cherche le plus proche cluster du cluster fusionné

```
procedure cluster(S, k)
begin
1.  T := build_kd_tree(S)
2.  Q := build_heap(S)
3.  while size(Q) > k do {
4.    u := extract_min(Q)
5.    v := u.closest
6.    delete(Q, v)
7.    w := merge(u, v)
8.    delete_rep(T, u); delete_rep(T, v); insert_rep(T, w)
9.    w.closest := x /* x is an arbitrary cluster in Q */
10.   for each x ∈ Q do {
11.     if dist(w, x) < dist(w, w.closest)
12.       w.closest := x
13.     if x.closest is either u or v {
14.       if dist(x, x.closest) < dist(x, w)
15.         x.closest := closest_cluster(T, x, dist(x, w))
16.       else
17.         x.closest := w
18.       relocate(Q, x)
19.     }
20.     else if dist(x, x.closest) > dist(x, w) {
21.       x.closest := w
22.       relocate(Q, x)
23.     }
24.   }
25.   insert(Q, w)
26. }
end
```

Extrait les 2 plus proches clusters

Fusionne les 2 clusters

Insère le cluster fusionné dans la pile des clusters

CURE : Clustering Using REpresentatives

Algorithme de clustering hiérarchique ascendant très optimisé et NON-limité aux clusters hyper-sphériques

- Pratique deux échantillonnages :
  - « choisi » au sein d'un cluster avec ses points les plus représentatifs, pour accélérer sans dégrader les résultats
  - « aléatoire » sur les données d'entrée pour les applications large échelle
- + d'autres optimisations

Mais … semble très séquentiel !

Rmq : Les algorithmes très optimisés sont souvent plus difficiles à paralléliser/distribuer.

Algorithmes de *clustering*

1. Clustering hiérarchique
2. Clustering non-hiérarchique par partitionnement
3. Clustering non-hiérarchique large échelle
4. Clustering hiérarchique non-sphérique et large échelle
5. **Autres solutions**

57

Autres solutions de clustering

Démarche précédente :
complexifier (encore) les méthodes inspirées des K-means

Autre démarches :
utiliser d'autres familles de méthodes mathématiques et d'algorithmes pour réaliser du clustering

- Méthode basées sur la **détection de champs de densité de points**: rassemblent dans un cluster tous les points proches les uns des autres tant qu'il y a une forte densité de points (voir l'algorithme de DBSCAN)
- Méthodes **connexionnistes dynamiques**, qui créent des neurones pour coloniser un espace de points aux formes complexes
- …

58

Algorithmes de *clustering*

59