Quentin JODELET, Alexis LECHAT

TP K-MEANS

RAPPORT VERSION FINALE

MAJ INF 413 – ALGORITHMIQUE AVANCEE

PRINCIPE DE L'ALGORITHME K-MEANS

Le but de l'algorithme est de répondre à un problème de classification et ainsi de regrouper des données ayant des propriétés semblables en différentes classes.

Dans notre cas, la classification s'effectue par la projection des données dans un espace métrique.

On choisit un entier k désignant le nombre de classes,

Les éléments sont alors classés en fonction de leur distance par rapport à k centres.

Dans un premier temps, ces k centres sont choisis aléatoirement parmi les données en entrée, puis ils sont recalculés à chaque itération en fonction des regroupements intermédiaires.

FONCTIONS UTILISEES

Notre implémentation de l'algorithme k-means se décompose avec les fonctions suivantes :

- distance : calcule la distance entre deux vecteurs.
- barycenter : calcule le barycentre d'un ensemble de vecteurs.
- distToCenter : calcule toutes les distances entre les données et les centres.
- groupByCenters : groupe les données en classe selon leur distance aux centres.
- randomIndexCenters : selectionne k éléments aléatoirement pour initialiser les centres.
- kmeans : algorithme k-means

CALCUL DE LA DISTANCE

```
Fonction distance(x , y)
    dist = 0
    Pour i allant de 0 à longueur(x)
        dist += (x[i] – y[i])^2
    Fin Pour
    Renvoyer sqrt(dist)
```

La fonction distance sert à calculer la distance entre deux vecteurs d'un même espace. Dans notre programme, nous utilisons la 2-distance :

Soit x, y deux vecteurs de $\mathbb{R}^n$, $\mathrm{d}(\mathrm{x},\mathrm{y}) = \sqrt{\sum_{\mathrm{i}=1}^{\mathrm{n}}(\mathrm{x_i} - \mathrm{y_i})^2}$

Cette distance étant la plus intuitive, notamment pour les représentations graphiques des données en dimension 2 et 3.

L'utilisation d'une autre distance est possible en modifiant cette fonction.

BARYCENTRE

```
Fonction barycenter(X)
    bCentre = vecteur nul de taille longueur(X[0])
    Pour i allant de 0 à longueur(X[0])-1
        Pour j allant de 0 à longueur(X)
            bCentre[i] += X[j][j+1]
        Fin Pour
        bCentre[i] /= longueur(X)
    Fin Pour
    Renvoyer bCentre
```

La fonction calcule le barycentre d'une famille de vecteurs X. Soit un espace de dimension n, X contenant p vecteurs de cet espace, on réalise la moyenne de chaque coordonnée des vecteurs de X. Ainsi, le barycentre aura pour coordonnée d'indice i, la moyenne de toutes les coordonnées d'indice i des vecteurs de X.

DISTANCE AU CENTRES

```
Fonction distToCenters(X, centers)
    distMatrix = matrice nulle de taille (longueur(X) * longueur(centers)
    Pour i allant de 0 à longueur(X)-1
        Pour j allant de 0 à longueur(centers)-1
            distMatrix[i][j] = distance(X[i], centers[j])
        Fin Pour
    Fin Pour
    Renvoyer distMatrix
```

Soit X une famille de vecteurs, centers une famille de vecteurs désignés comme centres. La fonction distToCenters permet de calculer toutes les distances entre les vecteurs de X et les centres. Les résultats de ces

calculs de distances sont stockés dans une matrice où une élément de coordonnées (i,j) correspond à la distance entre le vecteur de X d'indice i et le centre d'indice j.

CLASSIFICATION PAR RAPPORT AUX CENTRES

```
Fonction groupByCenters(X, centers)
    distMatrix = distToCenters(X, centers)
    vectGroup=[ [] pour k allant de 0 à longueur(centers)-1]
    Pour i allant de 0 àlongueur(X)-1
        minIndice = distMatrix[i].indice(min(distMatrix[i]))
        vectGroup[minIndice].concatener(X[i])
    Fin Pour
    Renvoyer vectGroup
```

Prenant l'ensemble des données et l'ensemble des centres en entrées, cette fonction répartie ensuite les données dans une liste contenant une liste pour chaque centre. On lit la matrice des distances aux centres, et on place une donnée dans la liste correspondant au centre dont elle est le plus proche.

INITIALISATION ALEATOIRE DES CENTRES

```
Fonction randomIndexCenters(nbData, k)
    index = liste contenant tous les entiers de 0 à nbData-1
    indexCenters = []
    Pour i allant de 0 à k-1
        randomInt = random entier entre 0 et longueur(index)-1
        indexCenters.concatener(index[randomInt])
        index.retirer(randomInt)
    Fin Pour
    Renvoyer indexCenters
```

L'initialisation de k centres aléatoires se fait en sélectionnant k indices parmi la liste index qui contient tous les entiers entre 0 et le nombre de données. Une fois qu'un indice est sélectionné au hasard, on le supprime pour ainsi assurer le fait que les centres soient distincts. On obtient à la fin une liste de k indices, dans la famille des données X, les vecteurs correspondant à ces indices seront les centres à l'initialisation de l'algorithme k-means.

K MEANS

```
Fonction kmeans(data, k)
    nbData = longueur(data)
    centers = [data[j] pour j dans randomIndexCenters(nbData, k)]
    groups = groupByCenters(data, centers)
    formerCenters=[]
    Tant que formerCenters != centers
        formerCenters = centers
        Pour i allant de 0 à k-1
            centers[i] = barycenter(groups[i])
        Fin Pour
        groups = groupByCenters(data, centers)
    Fin Tant que
    Renvoyer groups, centers
```

La fonction k-means est le cœur de l'algorithme, elle utilise les fonctions précédemment définies. Les centres sont initialisés aléatoirement, les données sont groupées autour de ces centres puis de nouveaux centres sont calculés et la boucle de classification se répète ainsi.

La condition d'arrêt du programme est « Tant que les centres changent d'une itération à l'autre, continuer le programme ». Ainsi le programme ne s'arrête que lorsque aucun centre ne s'est déplacé et donc que les groupes de données n'ont pas changé lors de la dernière itération. Grâce à cette condition, on est sûr d'avoir atteint la classification optimale par rapport à la valeur de k donnée en entrée.

TESTS DE L'ALGORITHME

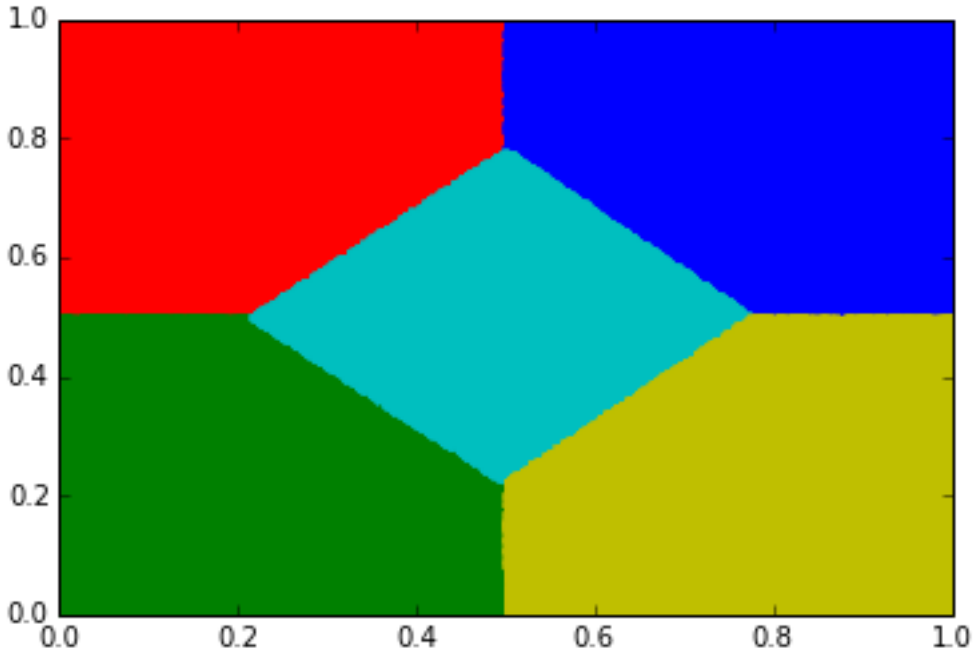

Figure 1 – Algorithme k-means (k=5) appliqués à 200 000 données de dimension 2 générées aléatoirement

Suites aux différents tests sur des données aléatoires en deux dimensions, on obtient des classifications qui semblent intuitivement correctes au sens d'une classification par la distance.

Appliqué aux données fournies en exemple (iris.data), on obtient les résultats suivants :

	Groupe 0	Groupe 1	Groupe 2
Iris virginica	36		14
Iris versicolor	2		48
Iris setosa	0	50	0

Si l'on considère que le groupe 0 est celui des virginica, le 1 celui des setosa et le 2 celui des versicolor, on obtient 16 erreurs de classifications, c'est-à-dire un taux d'erreurs de 10,7%.

Ce taux d'erreurs est acceptable, les erreurs viennent des spécimens dont les caractéristiques ne les rendent pas assez distinguables. Optimiser la correction de la classification peut éventuellement être réalisé en changeant la distance utilisée ou bien en augmentant le jeu de données. En effet, un plus grand nombre de spécimens permettra à l'algorithme de mieux distinguer les caractéristiques propres à chaque espèce.

ETUDE DE COMPLEXITE

Les tests présentés dans cette section ont été effectués sur un PC équipé d'un processeur Intel core I3-5010U 2.10GHz et de 4GO de RAM.

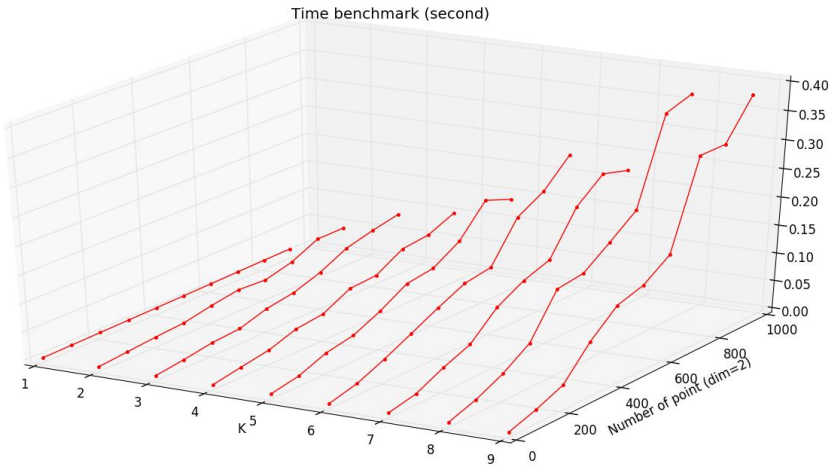

Figure 2 – Temps de calcul en fonction du nombre de données et de la valeur de k

La figure 2 permet d'avoir un premier aperçu du temps de calcul requis par l'algorithme. Ici, le nombre de données traité est faible (inférieur à 1000 et de dimension 2), on remarque cependant la nette augmentation du temps de calcul proportionnellement à la valeur de k. Ceci est cohérent avec l'algorthme. En effet, augmenter la valeur de k de 1 signifie un centre supplémentaire et donc un passage supplémentaire dans la boucle calculant la distance entre tous les points et comparant cette distance aux autres distances point-centre.

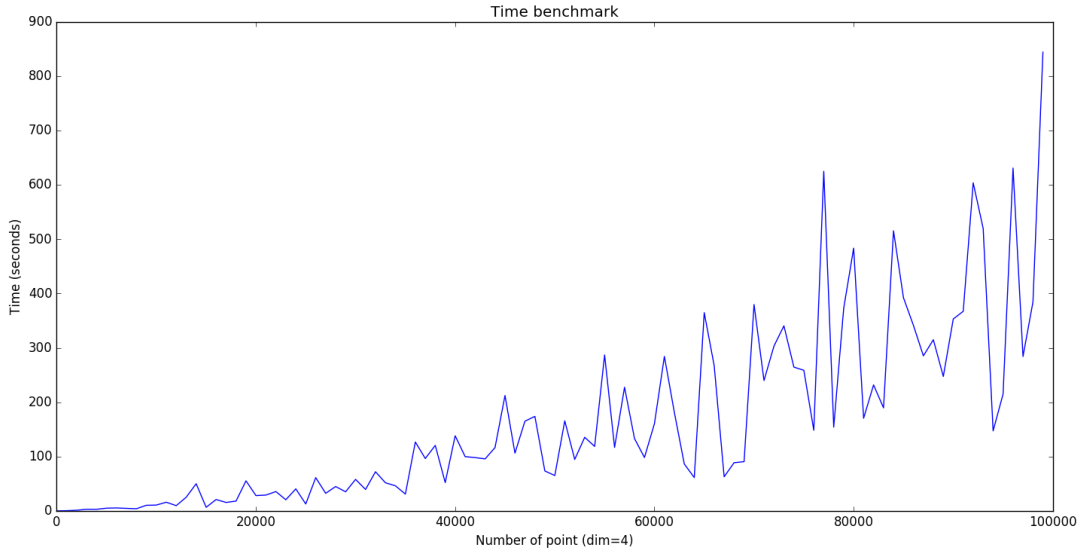

Figure 3 – Temps de calcul pour k = 5 et un nombre de données allant de 5 à 100 000

Pour la figure 3, aucun lissage n'a été réalisé. C'est-à-dire que pour chaque nombre de données, l'algorithme n'a été exécuté qu'une seule fois. On remarque dès lors la très nette irrégularité dans le temps de calcule. Ceci est dû principalement au nombre imprévisible d'itérations effectués par l'algorithme. En effet, en raison de l'initialisation aléatoire des centres, les centres peuvent être amenés à changer pendant plusieurs centaines d'itération avant de trouver une classification correcte, tout comme il est possible qu'ils se fixent après une dizaine d'itérations.

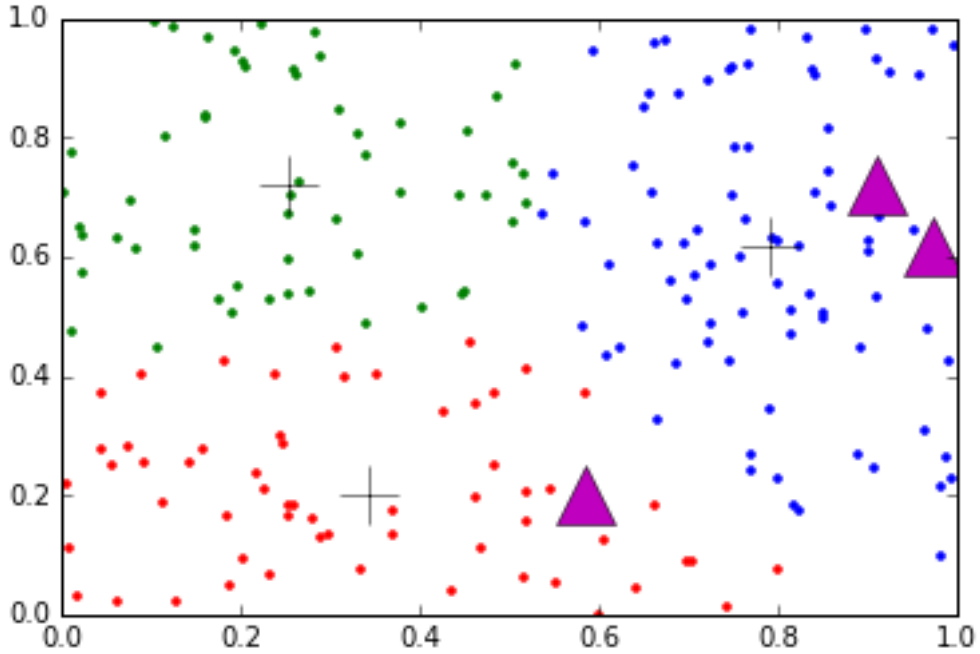

Figure 4 – Classification de 200 points pour k=3. Les triangles sont les centres initialisés aléatoirement et les croix les centres finaux

Pour pallier à ce problème, une solution est de modifier l'initialisation des centres, tout en gardant le facteur aléatoire (car il est impossible de prédire quels centres seront optimaux avant la réalisation de l'algorithme k-means). En prenant au hasard un certain pourcentage de données, en calculant le barycentre de ce sous-ensemble pour en faire un centre, et en répétant cette opération pour chaque centre à initialiser, on a ainsi plus de chance d'obtenir des centres aux endroits appropriés et d'identifier au préalable les zones de concentration possible.

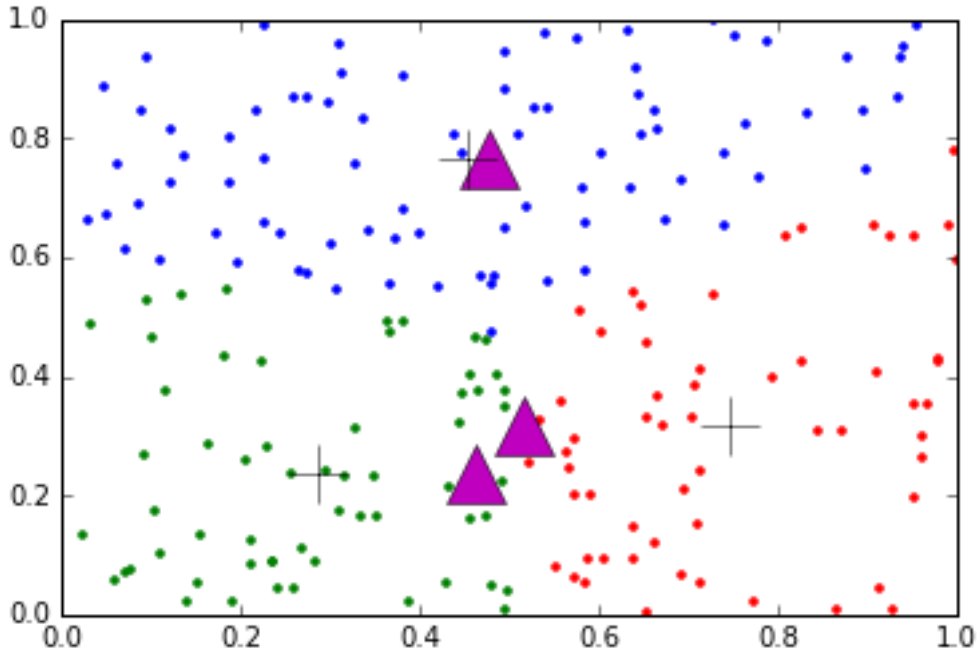

Figure 5 – Classification de 200 points pour k=3. Les triangles sont les centres dont l'initialisation a été optimisée et les croix les centres finaux

Pour réaliser la figure 5, lors de l'initialisation des centres, à trois reprise, un sous-ensemble de 50% des points a été sélectionné aléatoirement et le barycentre de ce sous-ensemble a été calculé pour donner un centre.

Cette méthode reste aléatoire, il existe toujours des pires cas possibles dans lesquels un grand nombre d'itérations sera nécessaire pour réaliser la classification. Il permet néanmoins d'avoir, en moyenne, une diminution du nombre d'itérations nécessaires. Sur l'exemple précédent, on remarque bien que cette méthode permet d'avoir des centres qui s'approchent de ceux à l'état final.

Outre le problème du nombre imprévisible d'itérations nécessaire, intéressons-nous maintenant au temps de calcul moyens. Pour cela, le test suivant a été répété 20 fois pour chaque taille de données. La figure 6 présente donc la moyenne du temps de calcul sur 20 expériences.

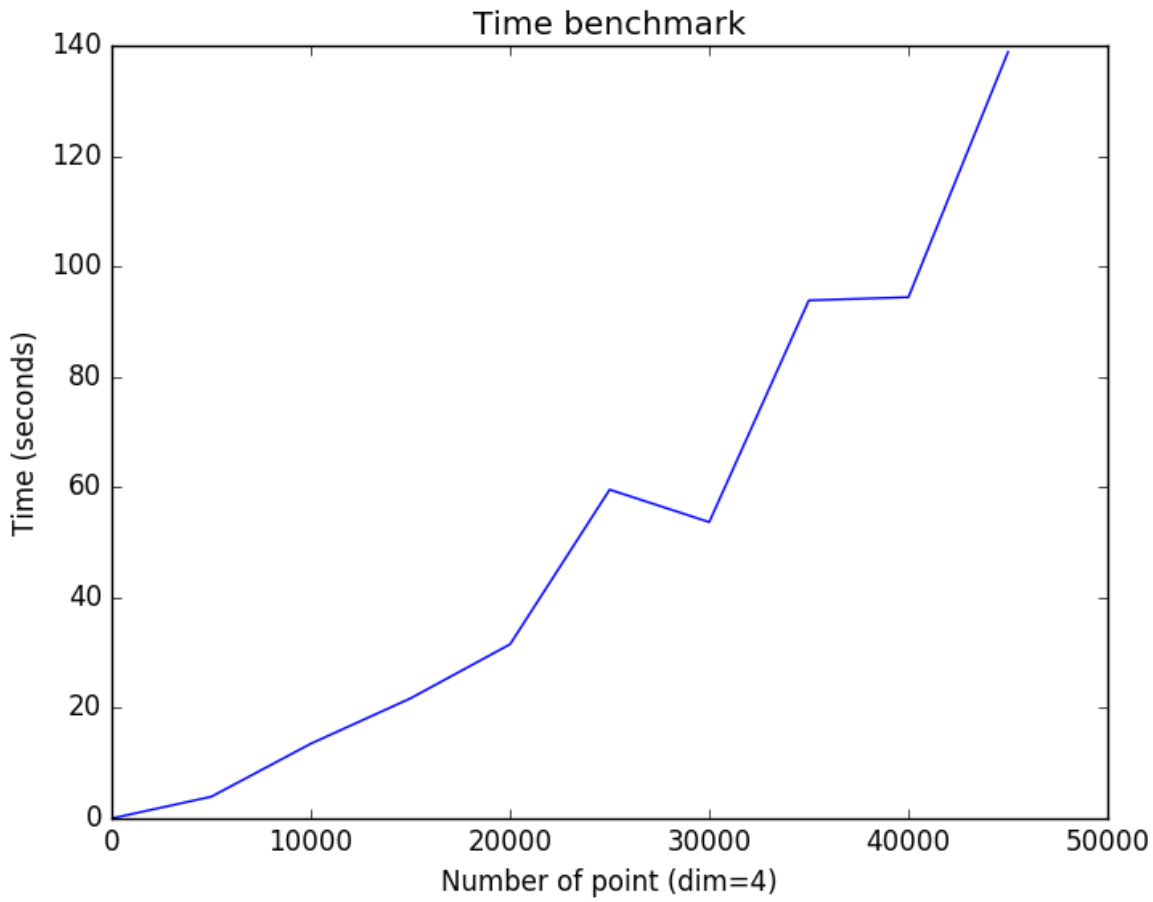

Figure 6 – Temps de calculs pour k=5

On remarque une croissance en temps qui semble polynomiale voire exponentielle. Ceci est logique en raison de notre implémentation qui à chaque itération, réalise une boucle sur toutes les données pour calculer toutes les distances aux centres, puis boucle de nouveau sur toutes les données pour réaliser la classification dans chaque groupe. Vient s'ajouter à cela le nombre d'itérations requis qui est en moyenne croissant.