Rapport de projet

Saidi Rayane – Master Informatique

Les algorithmes de recommandations sont des moyens de recommander un contenu pertinent à un certain utilisateur. Ceux-ci sont utilisés dans une multitude de domaines tel que l'e-commerce ou les plateformes de streaming. Il existe plusieurs de ces algorithmes dont celui qui nous intéresse dans ce projet : Annoy (Approximate Nearest Neighbor Oh Yeah). Crée par Spotify, il est aujourd'hui largement utilisé grâce à son efficacité en temps.

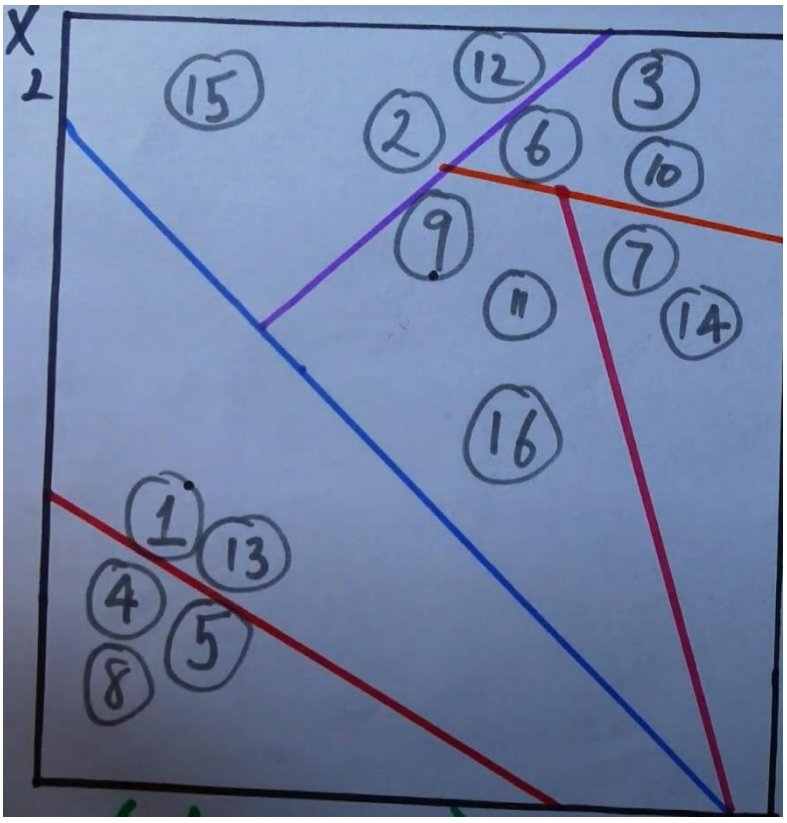
Annoy

Les données sont représentées à l'aide de vecteur à plusieurs dimensions. Le but est de connaitre les k-plus proches voisins de notre vecteur v.

Une approche naïve consisterait à calculer la norme de Frobenius entre v et tous les vecteurs restants. Puis on ne retient que les k-plus petits. Le problème étant que la complexité est de O(n) (car on utilise tous les vecteurs de notre DataSet).

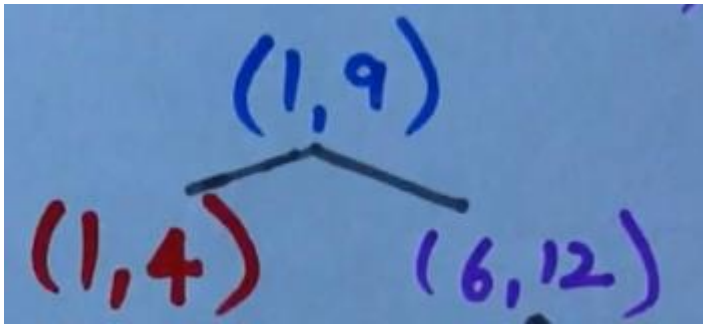
L'algorithme d'Annoy tente une autre approche en utilisant **un arbre binaire de recherche.**

Celui-ci nécessit l'utilisation d'un graphique pour mieux comprendre.



Prenons d'abord au hasard deux vecteurs aléatoires : numéro 1 et 9 (on les appellera directement 1 et 9 pour simplifier). Il faut dessiner une ligne perpendiculaire à la ligne connectant 1 et 9. On obtient donc deux Zones (Zones 1 et 9). On réitère l'opération dans chaque zone jusqu'à ce qu'une d'elle contienne au maximum k-éléments.

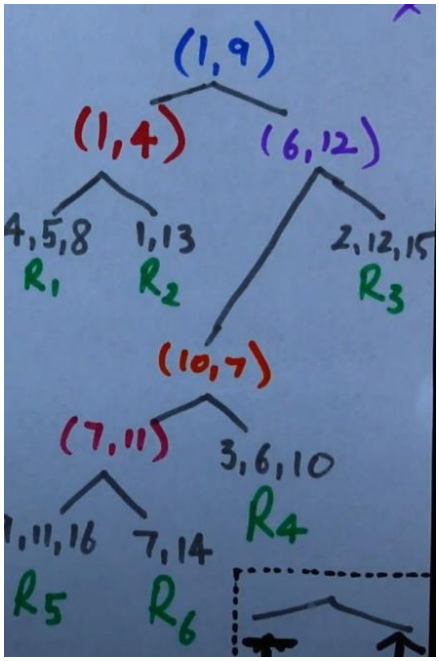
On peut représenter ces opérations grâce à un arbre binaire de recherche :



Tous les éléments à gauche du couple (1,9) appartiennent à la zone 1 (en bas du trait bleu dans notre graphique).

Tous les éléments à droite du couple (1,9) appartiennent à la zone 9 (en haut du trait bleu dans notre graphique).

Au final on obtient l'arbre suivant, où les feuilles représentent les zones finales :



Si l'on veut trouver les k-plus proches voisins d'un élément, il suffit de parcourir cet arbre jusqu'à trouver la bonne zone de notre élément. Quand on arrive dans une feuille, on obtient donc au maximum k-voisins (dans notre exemple 3), il suffit ensuite de calculer la norme de Frobenius pour ses 3 voisins et de faire le tri.

Note : Il faut faire une forêt d'arbre car il y'a des cas ou l'algorithme peut se tromper.

La complexité

Elle est de O(log(n)) car on parcourt un BST au lieu de parcourir tout le DataSet.

Implémentation

Un nettoyage du DataSet était d'abord nécessaire :

- Renommer les colonnes.
- Supprimer les 's' dans la colonne film_id.
- Transformer toutes les valeurs de cette colonne en float.

J'ai réalisé une représentation visuelle des vecteurs (en ne prenant que 100 vecteurs aléatoires).

La logique de ma fonction Annoy est la suivante :

- On regroupe les films pour chaque utilisateur. On aura donc user_id => [film 1, film 2 ...]
- On parcourt la structure précédente en créant pour chaque utilisateur un vecteur dont le premier élément sera l'id de celui-ci, tandis que le reste sera composé des films qu'il recommande.
- On définit un max de 40 films recommandés. Si un utilisateur en a moins, on remplit le vecteur de 0. S'il a trop de recommandation, on ne garde que les 40 premières.
- On donne chaque vecteur à la fonction Annoy de python.
- On crée une forêt de 10 arbres et on récolte les résultats

Le but de mon implémentation est de trouver **les utilisateurs** qui ont **des gouts** similaires. En appelant ma fonction, on obtiendra alors (d'abord des indices, puis faire le travail de correspondance) les k-utilisateurs les plus proches. Il suffit ensuite de prendre les premiers films de chaque utilisateur.

Problèmes

Je n'ai rencontré aucun problème particulier, tout était très bien documenté.

Annexe

Mes illustrations proviennent de cette vidéo : https://www.youtube.com/watch?v=DRbjpuqOsjk&t=428s&ab_channel=ritvikmath