UNIVERSITÉ DE LORRAINE

Les moutons noirs : Etude de l'atypisme

Membres du groupe :
Gautier Dran
Clément Spies

Encadrants :
Armelle Brun
Benjamin Gras

2016/2017

Remerciements

Nous remercions nos tuteurs Armelle Brun et Benjamin Gras pour leurs disponibilités, leurs précieux conseils ainsi que leur accompagnement tout au long du projet. Nous tenons à remercier aussi toutes les personnes qui nous ont conseillé et relu lors de la rédaction de ce rapport.

Table des matières

1- Introduction

Contexte Général

Ce projet est proposé par l'équipe de recherche Kiwi (Knowledge, Information & Web Intelligence) du Loria. Cette équipe a pour but d'améliorer la qualité du service rendu par les systèmes informatiques aux utilisateurs et de faciliter les interactions entre les utilisateurs et les systèmes de recherche. Pour ce faire ils utilisent des systèmes de recommandations.
Un système de recommandation permet de filtrer l'information pour récupérer ce qui va intéresser un utilisateur donné afin de lui proposer des choses (film, musique, objets, sites) qui pourraient intéresser l'utilisateur. L'idée d'un système de recommandation est de se baser sur les ressemblances entre les utilisateurs si l'utilisateur A et l'utilisateur B aiment l'objet C et que l'utilisateur B aime l'objet D il est possible que A aime D.

Présentation du sujet

Énoncé du sujet repris depuis la fiche de projet tutoré (voir référence 1)

La recommandation dite sociale se sert des avis des utilisateurs qui vous ressemblent, donc qui partagent des préférences avec vous, pour déduire votre avis sur une ressource (un film, une musique, un livre, un site web, …) qui vous sera éventuellement recommandée par la suite. Cependant, chaque être humain possède au moins une préférence qui le différencie des autres et les systèmes de recommandation ne prennent pas en compte de manière particulière ces spécificités. Alors qu'en est-il si vous ne partagez pas assez de caractéristiques avec les autres, que vous êtes trop différent ? Et si vous étiez un mouton noir ? Il s'agit là du problème des moutons noirs de la recommandation sociale.

Les systèmes recommandation actuels fournissent des recommandations de mauvaise qualité aux utilisateurs « moutons noirs ». Cependant, tout un chacun a le droit de recevoir des recommandations de qualité, quelles que soient ses spécificités. Il reste donc de nombreuses pistes d'amélioration quant à la détection des ces utilisateurs ainsi que les solutions proposées pour améliorer les recommandations qui leur sont fournies. Un mouton noir est-il quelqu'un qui possède des avis toujours/souvent/quelquefois différents des autres ? Comment définir ce seuil ? Comment calculer une différence d'opinion ? Comment la gérer pour améliorer la qualité des recommandations qui leurs sont fournies ?

De nombreuses solutions sont envisageables et nécessitent la mise en place de protocoles de test pour étudier leurs validités. Le choix des pistes (qu'elles soient d'inspiration mathématique, informatique, psychologique, etc.) à explorer et des protocoles de validation seront à l'initiative des étudiants et discutées et validées avec et par les encadrants.

Déroulement du Projet

Dans un premiers temps, nous nous sommes renseignés grâce aux documents fournis par nos tuteurs. Ces documents dont nous parlerons plus tard nous ont permis de savoir dans quel contexte nous allions travailler (celui de la recommandation de films, plus particulièrement dans celle pour les utilisateurs atypiques). Ensuite nous avons recherché quels étaient les différents algorithmes de clustering qui pourraient être intéressants dans notre cas. Ensuite nous avons commencé à développer les algorithmes de clustering. Et pour finir nous avons cherché des hypothèses que nous trouvions intéressantes à démontrer.

Plan du rapport

Nous allons donc vous présenter les diverses parties suivantes en commençant par l'état de l'art, puis suivra une description du travail réalisé, une démonstration des divers résultats, ensuite une ouverture par rapport à ces résultats et enfin nous ferons une conclusion sur le projet en général.

2- Etat de l’art

Dans ce chapitre nous allons vous présenter ce qu’est un système de recommandation, ensuite nous présenterons ce qu’est le clustering, et enfin nous présenterons les différentes méthodes de clustering qui ont attirées notre attention.

Les systèmes de recommandations (SR)

Les systèmes de recommandations sont une forme spécifique de filtrage de l’information visant à présenter les éléments d’information qui pourraient intéresser les utilisateurs.*(voir 2)* Les cadres d’application de ces systèmes sont multiples : réseaux socio-numériques, marketing digital avec la relation client pour la vente en ligne ou services personnalisés liés à une offre culturelle. Les SR cherchent donc à prédire la valorisation qu’un utilisateur peut attribuer à un objet (livre, musique, film, etc) ou un élément social (personne, groupe, etc), c’est le cas pour les sites tels que Deezer (musiques), Amazon (livres, films), Facebook (personnes, groupes) et pleins d’autres. Les SR ont comme intérêt pour l’utilisateur de réduire le temps de recherche d’informations et découvrir des produits difficiles à trouver ou qui pourraient l’intéresser. Et ils ont un intérêt pour le fournisseur du service : celui d’orienter le client et d’augmenter ses bénéfices.

Le principe, à l’origine des premiers SR, s’appelle le filtrage collaboratif. L’idée est que : si Clément a des idées similaires à Gautier sur un sujet, alors il y a des chances pour que Clément partage son avis sur un autre sujet, plutôt que celui de quelqu’un pris au hasard. Lorsqu’un système veut proposer des objets intéressants à Clément, il a pour but de prédire ses opinions sur ces objets. Les objets sur Deezer, ce sont des musiques, sur Amazon ce sont des produits et sur Facebook ce sont les statuts et articles partagés ou aimés. Cet ensemble de notes constituent une représentation des centres d’intérêts de l’utilisateur. L’étape suivante consiste à rechercher les utilisateurs qui ont les avis les plus similaires à ceux de Clément, en se servant des notes de chacun. Le système détermine ensuite les objets que ces utilisateurs ont vus mais que Clément n’a pas encore consultés. Avec leurs avis il est donc possible de déterminer les notes que Clément aurait pu mettre à ses objets.

Dans notre cas, environ 95% des données que nous utiliserons (notes d’utilisateurs sur des films) sont manquantes, ce qui fait que nous allons partitionner les éléments à partir des 5% de données présentes.

Le clustering

Le principe du clustering est de diviser automatiquement un ensemble de données en différents groupes, appelés Clusters. Le regroupement des données se fait par mesure de distance. Chaque donnée est représentée par des caractéristiques qui sont utilisées pour calculer la distance entre deux éléments. Plus ces distances sont faibles plus les éléments seront proches.
Pour que le clustering soit optimal il faut que l’inertie intra-classe (c’est-à-dire les distances entre les points et le centre du cluster) soit minimisée, et que l’inertie inter-classe (c’est à dire les distances entre chaque cluster) soit maximisée.
Le domaine d’applications du clustering est vaste :

- Reconnaissances des formes (ex : Formation de 2 clusters pour différentier les oies et les canards en fonction de la taille et de la couleur). *(voir 3)*
- Analyse de données spatiales (ex : La distribution géographique des maladies. Un cluster peut être défini comme une concentration de cas "anormalement élevée", supérieure à celle attendue, dans un groupe de personnes, une zone géographique ou une période de temps. Et cela permet d'en étudier l'hétérogénéité spatiale) *(voir 4)*
- Traitement d'images (ex : segmentation d'images en couleurs) *(voir 5)*
- Recherche d'informations *(voir 6)*
- Web mining *(voir 7)*

Méthodes de clustering existantes

- K-Means

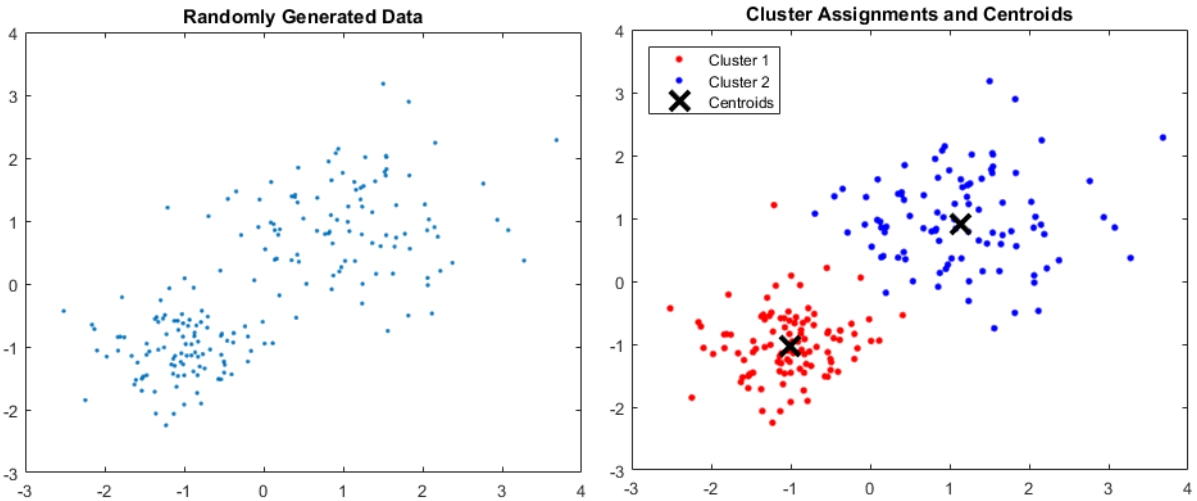
Le K-Means est une des méthodes de clustering les plus populaires car elle est facile à comprendre et à mettre en oeuvre. Elle peut être utilisée pour effectuer une segmentation d'image par exemple. C'est une méthode itérative qui, selon le point de départ, pourra donner une solution différente. C'est pour cette raison que l'on répète plusieurs fois le calcul pour ne retenir que la solution la plus optimale pour le critère choisi. Le partitionnement en K-Means se fait de la façon suivante :

[1] On choisit un point de départ qui consiste à associer le centre des k classes (centroïdes) à k objets aléatoirement (k étant le nombre clusters voulu)

[2] On met à jour les clusters en calculant les distances entre les objets et les k centroïdes et on affecte les objets aux centroïdes dont ils sont les plus proches.

[3] Réévaluation des centroïdes à partir des objets qui ont été affectés aux différentes classes.

[4] On reproduit les étapes [2] et [3] jusqu'à la stabilisation des centroïdes. *(voir 8)*



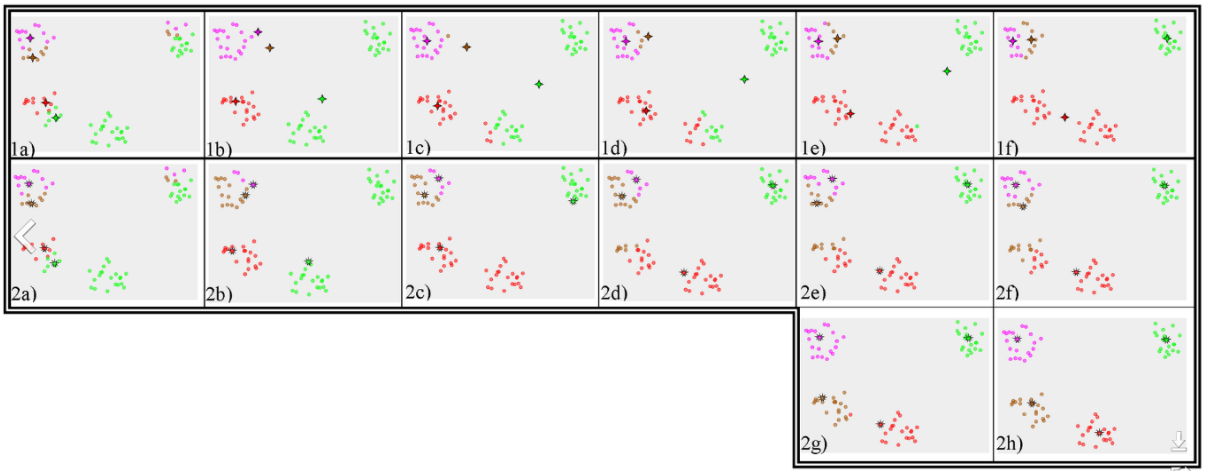
A gauche les éléments représentés sur un tableau 2D à clusteriser, et à droite le résultat final si l'on a choisi 2 clusters comme paramètre. Le cluster 1 en rouge, le cluster 2 en bleu et leur centroïd représentés par une croix (voir 12)

Un des avantages de cette méthode, mis à part sa simplicité conceptuelle et sa rapidité, est qu'un objet peut être affecté à une classe au cours d'une itération puis changer de classe à

l'itération suivante, ce qui peut être avantageux lorsque un élément a été mal placé dès le départ, et ce qui n'est pas possible avec la classification ascendante hiérarchique, que l'on présente par la suite, pour laquelle une affectation est irréversible.
L'inconvénient avec cette méthode c'est que le nombre de classe doit être fixé au départ et que le résultat dépend du tirage initial des centroïdes. De plus cette méthode de clustering a un gros problème connu : un problème de convergence. En effet cet algorithme n'est pas sûr de s'arrêter, il se peut qu'il y ait un élément qui oscille entre deux ou plusieurs clusters. *(voir 10)*

- K-Médoïdes

L'algorithme des K-Médoïdes est un algorithme de partitionnement qui se base essentiellement sur la notion de médoïdes pour créer des clusters. Les médoïdes sont des éléments d'un ensemble qui minimise la somme entre eux et chacun des autres éléments du cluster. Le médoïde d'un cluster est donc le point du cluster le plus proche de tous les autres.
Si les clusters sont relativement convexes, leur position dans l'espace de représentation est très proche du barycentre. Par contre, pour les clusters non convexes, ou en présence de données atypiques, cette méthode se révèle plus robuste que le K-Means en dépassant le caractère artificiel du barycentre.
Comme le K-Means, le K-Médoïdes va minimiser l'erreur quadratique moyenne qui est la distance entre les points du cluster et le centroïde.
Sa différence avec le K-Means est que les centroïdes du K-Medoïdes seront représentés par un de ses points. *(voir 12)*
Parmi les premières versions de la méthode K-Médoïdes nous pouvons citer l'algorithme PAM (Partitioning Around Medoids) et l'algorithme CLARA (Clustering LARge Applications).



De 1a) à 1f) : K-Means, de 2a) à 2h) : K-Médoïdes (voir 12)

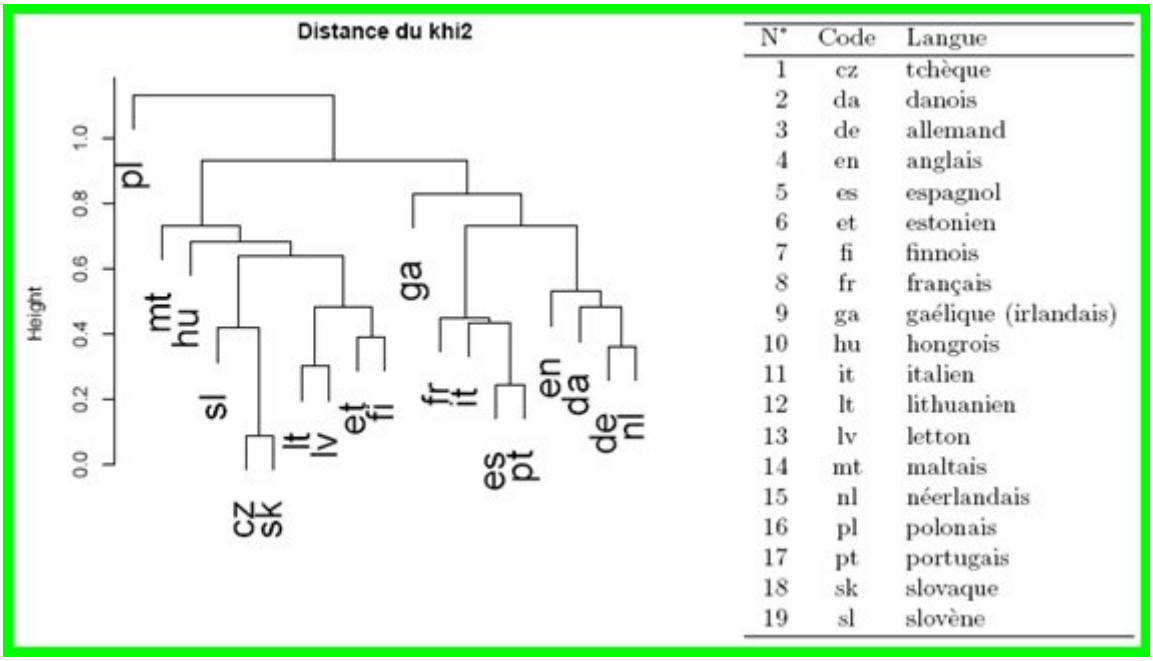
On peut voir dans l'exemple ci-dessus que la méthode des K-médoïdes est la plus efficace car elle converge vers la classification la plus évidente contrairement au K-Means qui converge vers un minimum local.

Mis à part la plus grande robustesse vis à vis des données aberrantes et la plus grande complexité de chaque itération de cette méthode, le K-Médoïdes présente les mêmes avantages et inconvénients que le K-Means. *(voir 15)*

- Classification ascendante hiérarchique

La classification ascendante hiérarchique est dite ascendante car elle part d'une situation où tous les individus sont seuls dans une classe, puis sont rassemblés en classes de plus en plus grandes. *(voir 17)*

Initialement, chaque individu forme une classe, soit *n* classes. On cherche à réduire le nombre de classes, ceci se fait itérativement. À chaque étape, on fusionne deux classes, réduisant ainsi le nombre de classes. Les deux classes choisies pour être fusionnées sont celles qui sont les plus « proches », en d'autres termes, celles dont la similarité entre elles est maximale, cette valeur de similarité est appelée *indice d'agrégation*. Comme on rassemble d'abord les individus les plus proches, la première itération a un indice d'agrégation faible, mais celui-ci va croître d'itération en itération. Le résultat sera souvent représenté par un dendrogramme comme représenté ci dessous.



N°	Code	Langue
1	cz	tchèque
2	da	danois
3	de	allemand
4	en	anglais
5	es	espagnol
6	et	estonien
7	fi	finnois
8	fr	français
9	ga	gaélique (irlandais)
10	hu	hongrois
11	it	italien
12	lt	lithuanien
13	lv	letton
14	mt	maltais
15	nl	néerlandais
16	pl	polonais
17	pt	portugais
18	sk	slovaque
19	sl	slovène

Dans cet exemple si le nombre de classes est de 3, alors, en choisissant les éléments les plus proches par rapport à la distance Khi2, cela donnerait les clusters suivants : (pl), (mt,hu,sl,cz,sk,lt,lv,et,fi) et (ga,fr,it,es,pt,en,da,de,nl) (voir 19)

L'avantage de cette méthode est que ça nous permet de déterminer le nombre optimal de clusters en comparant les distances intra-clusters et inter-clusters en fonction du nombre de classes choisi. Mais cette méthode est coûteuse en temps de calcul.

- Algorithme d'espérance-maximisation (EM)

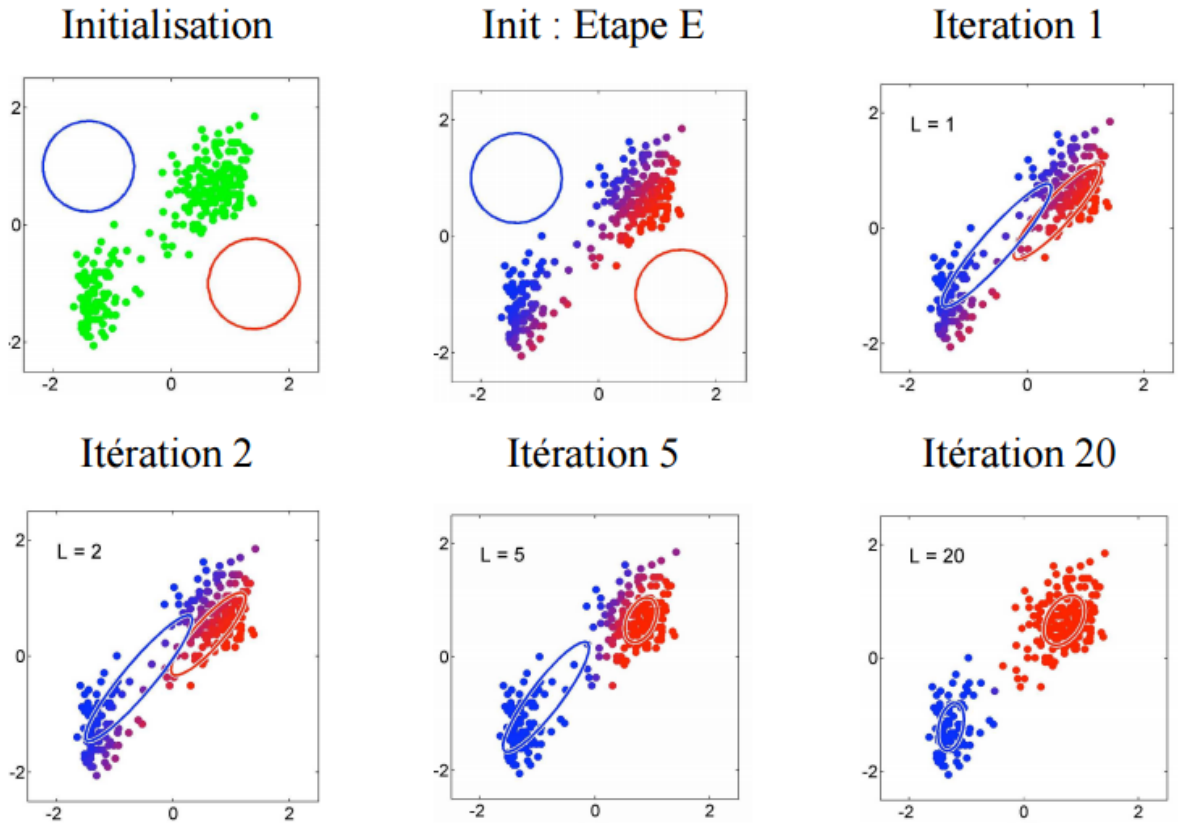
On utilise souvent l'algorithme EM pour la classification de données, l'apprentissage automatique, ou la vision artificielle.

L'algorithme d'espérance-maximisation est une méthode itérative qui comporte 2 phases :

- Phase E (estimation), où l'on calcule l'espérance de la vraisemblance des données en tenant compte des dernières variables observées.

- Phase M (maximisation), où l'on estime le maximum de vraisemblance des paramètres en maximisant la vraisemblance trouvée à l'étape E.

On itère jusqu'à la convergence en utilisant les paramètres trouvés en M comme point de départ d'une nouvelle phase d'évaluation de l'espérance. *(voir 20)*



Initialisation de l'algorithme EM en haut gauche, Etape E qui va, dans cet exemple, diviser les données en 2 groupes. Puis affichage de la représentation graphique après 1, 2, 5 et 20 itérations.

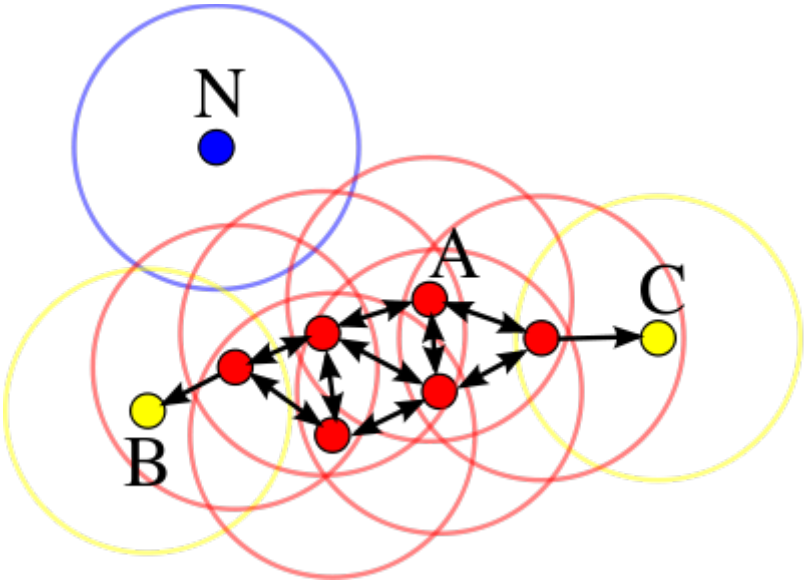
Les avantages pour cet algorithme est qu'il est stable numériquement et la vraisemblance croît a chaque itération (sauf à un point fixe de l'algorithme), il converge globalement sous certaines conditions, il est facile à programmer, puisque ni l'évaluation de la vraisemblance des données observées ni celle de ses dérivées ne sont nécessaires, il demande peu d'espace de stockage, et enfin le travail analytique nécessaire est plus simple que celui des autres méthodes puisque seulement l'espérance conditionnelle de la log-vraisemblance pour les données complètes a besoin d'être maximisée. *(voir 22)*

Et les inconvénients de cette méthode sont qu'elle peut converger lentement même pour les problèmes qui semblent inoffensifs. Elle peut converger lentement aussi lorsqu'il y a beaucoup d'informations manquantes. Et dans certains problèmes, l'étape E peut être analytiquement impossible à trouver. *(voir 22)*

- DBSCAN

L'algorithme DBSCAN cherche les groupes de points proches. Pour cela, l'algorithme utilise 2 paramètres : la distance E et le nombre minimum de points MinPoints. Il recherche, pour chaque point, les points étant dans un rayon E de celui ci, et il regroupe tous les points

atteignables de proche en proche. Une fois tous les points regroupés, les groupes de points sont considérés soit comme des clusters soit comme des valeurs aberrantes, si le nombre d'observations est inférieur à MinPoints.

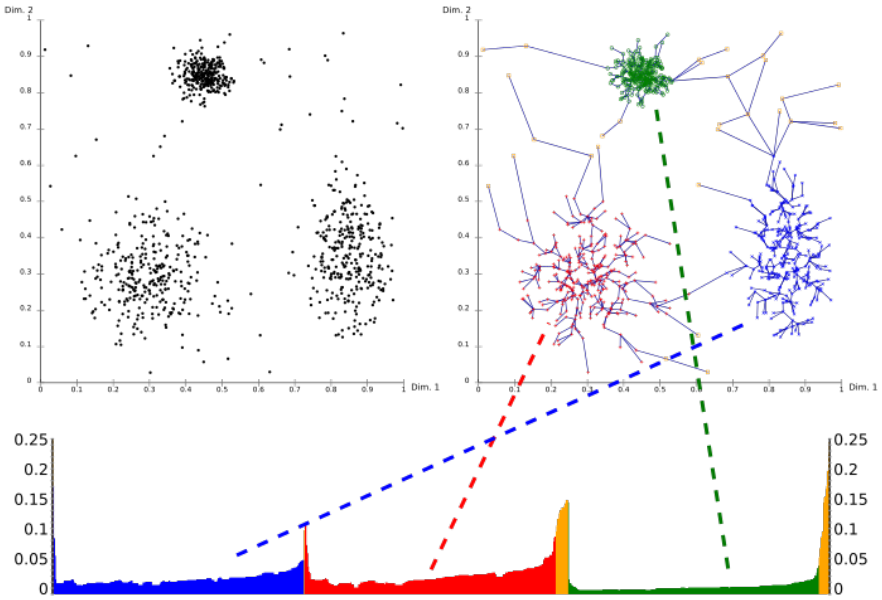


Les points A en rouge sont les points déjà dans le cluster. Les points B et C sont atteignables depuis A et appartiennent donc au même cluster. Le point N est une donnée aberrante puisque son epsilon voisinage ne contient pas MinPts points ou plus. (voir 23)

L'algorithme est très simple et il n'est pas nécessaire qu'on lui précise le nombre de clusters à l'avance. Il détecte et isole de lui même les données aberrantes. Les clusters n'ont pas pour obligation d'être linéairement séparables (en comparaison avec l'algorithme K-MEans par exemple). Mais cet algorithme n'est pas capable de gérer des clusters de densités différentes. Une amélioration de DBSCAN prenant ce problème en compte existe, elle s'appelle OPTICS et est expliquée ci dessous. *(voir 23)*

- OPTICS

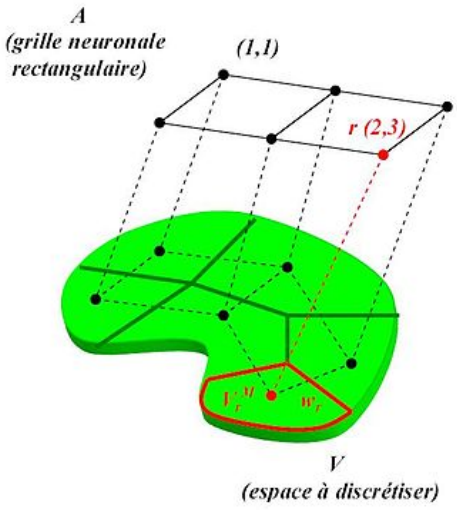
L'algorithme OPTICS est une extension de DBSCAN et, comme celui ci, OPTICS requiert deux paramètres : E, représentant un rayon maximum à considérer, et MinPts, représentant un nombre de points minimum. Ces 2 paramètres permettent donc de définir une densité minimale pour constituer un groupe de données. Un point p appartient à un groupe si au moins MinPts points existent dans son voisinage. En revanche, à l'inverse de DBSCAN, le paramètre ϵ est optionnel. Si il n'est pas spécifié, il sera alors considéré comme infini. L'algorithme définit pour chaque point une distance qui décrit la distance avec le MinPtsème point le plus proche. *(voir 26)*



La sortie de l'algorithme permet de construire un diagramme appelé reachability-plot. C'est un diagramme en 2D dont l'axe x correspond à la position d'un point dans la liste ordonnée et l'axe y la reachability-distance associée à ce point. Les points d'un même cluster ont une reachability-distance assez basse, les vallées du diagramme représentent donc les différents clusters du jeu de données. Plus les vallées sont profondes, plus les clusters sont denses. (voir 26)

- Cartes auto-adaptatives

La carte auto adaptative est un type de réseau de neurones artificiels dont l'apprentissage s'effectue de manière non supervisé. Elle se déploie de façon à représenter un ensemble des données, et chaque neurone se spécialise pour représenter un groupe bien particulier des données selon les points communs qui les rassemblent. Elle permet une visualisation en dimension multiple de données croisées. *(voir 28)*



Architecture des cartes auto-organisatrices. L'espace d'entrée V est divisé en plusieurs zones. ωr représente un vecteur référent associé à une petite zone de l'espace Vr^M et r(2,3) représente son neurone associé dans la grille A. Chaque zone peut être adressée facilement par les index des neurones dans la grille. (voir 28)

Après l'initialisation aléatoire des valeurs de chaque neurones, on soumet une à une les données à la carte auto adaptative. Selon les valeurs des neurones, celui dont la valeur sera la plus proche de la donnée présentée sera gratifié d'un changement de valeur pour qu'il réponde encore mieux à un autre *stimulus* de même nature que le précédent. Par là même, on gratifie aussi les neurones voisins du gagnant. Ainsi, c'est toute la région de la carte autour du neurone gagnant qui se spécialise. En fin d'algorithme, lorsque les neurones sont stables, la carte auto organisatrice recouvre toute la topologie des données. *(voir 28)*

Les ancêtres des cartes auto-organisatrices (les algorithmes comme K-Means), réalisent la discrétisation de l'espace d'entrée en ne modifiant à chaque cycle d'adaptation qu'un seul vecteur référent. Leur processus d'apprentissage est donc très long. *(voir 28)*

L'algorithme de Kohonen profite des relations de voisinage dans la grille pour réaliser une discrétisation dans un temps très court et il présente des opérations simples ; il est donc très léger en termes de coût de calculs. Mais l'inconvénient est que le voisinage dans les cartes auto adaptatives est fixe, et une liaison entre neurones ne peut être cassée même pour mieux représenter des données discontinues. *(voir 28)*

Documents fournis par nos tuteurs

Au début de ce projet, nos tuteurs nous ont donné plusieurs documents pour nous montrer ce qui avait été fait et avoir une première approche de ce problème de leur part et d'autres scientifiques :

- A) Toward the Next Generation of Recommender Systems: A Survey of the State-of-the-Art and Possible Extensions (Gediminas Adomavicius, Member, IEEE, and Alexander Tuzhilin, Member, IEEE)
- B) Identifying Grey Sheep Users in Collaborative Filtering: a Distribution-Based Technique (Benjamin Gras, Armelle Brun, Anne Boyer)
- C) Survey of Clustering Data Mining Techniques (Pavel Berkhin)
- D) Data Clustering Techniques (Periklis Andritsos)
- E) Identifying Users with Atypical Preferences to Anticipate Inaccurate Recommendations (Benjamin Gras, Armelle Brun, Anne Boyer)
- F) Identification des utilisateurs atypiques dans les systèmes de recommandation : vers une recommandation de qualité pour tous (Rapport de stage de Benjamin Gras)

Ces documents nous ont permis de bien comprendre la problématique et les enjeux de ce projet. Et ainsi avoir une première approche de cette problématique pour démarrer ce projet. Nous avons donc lu tous les documents sans trop nous attarder sur ceux ci pour essayer d'avoir une approche différente de nos tuteurs.

Les documents B, E et F nous ont permis de comprendre dans quel contexte se place notre projet, nous avons pu comprendre ce que sont les utilisateurs atypiques, de quelle façon ils sont identifiés. Quelles étaient les données qui pouvaient être à notre disposition.

Les documents A, C et D quand à eux présentent les systèmes de recommandation, le clustering dans le cadre de récupérations de données et les différentes techniques de clustering.

3- Travail réalisé

Rappels sur les objectifs

Notre objectif est de permettre les recommandations de films aux utilisateurs et plus particulièrement aux utilisateurs atypiques (moutons gris). Nous avons donc choisis de classifier nos données par rapport aux films. En effet, regrouper les films va nous permettre d'identifier des groupes de films similaires suivant différentes critères. Grâce à cela nous devrions être en mesure d'améliorer les recommandations quant aux moutons gris.
Nous disposons d'un fichier de données de 100 000 lignes.
Chaque ligne dispose de 3 éléments :

- l'identifiant de l'utilisateur
- l'identifiant du film
- la note que l'utilisateur a attribuée au film

Nous avons 1682 films différents ainsi que 943 utilisateurs.

Une fois que les films sont regroupés en cluster, nous avons émis 5 hypothèses :

- Les films avec une forte erreur de recommandation sont regroupés dans un ou deux clusters.
- Les films avec une faible erreur de recommandation sont regroupés dans un ou deux clusters.
- Les films très controversés sont réunis dans un cluster.
- Les films très controversés sont répartis de manière égale dans les clusters.
- Il existe un cluster regroupant des films qui plaisent aux utilisateurs atypiques

Nous reviendrons sur ces hypothèses dans une partie qui va suivre.

Premiers pas

Notre but étant de regrouper les films, il faut donc dans un premier temps "simplifier" la liste.
Au lieu d'avoir 100 000 éléments nous avons regroupé pour chaque film les notes des utilisateurs. Ce qui nous donnait 1682 éléments à traiter.
Nous nous sommes donc retrouvés avec un élément qui est représenté par des "coordonnées" (ici les notes données par les utilisateurs) sur 943 dimensions (le nombre d'utilisateur possible pouvant avoir noté le film).
Pour traiter ces données il a fallu choisir des méthodes de clustering.
Dans un premier temps, nous nous sommes penchés sur une liste d'algorithmes :

- l'algorithme de maximisation de l'espérance
- les cartes auto-organisatrices
- La Classification hiérarchique ascendante
- Le k-means et le k-médoïde
- DBSCAN et OPTICS

Après une réunion avec nos tuteurs nous avons décidé qu'il n'était pas pertinent d'implémenter tous les algorithmes, mais qu'il suffisait d'en choisir quelques uns.

Méthodes retenues

Les premiers algorithmes que nous avons "validés" sont l**e K-means et le K-médoïdes.**

Le K-means avait déjà été utilisé par l'un de nos encadrants et il nous a été recommandé. Pour le k-médoïde, nous avons choisis de valider cet algorithme pour plusieurs raisons, tout d'abord la similarité entre le k-means et le k-médoïde nous a poussé vers ce choix. Comme expliqué dans l'article (13), l'algorithme k-médoïde va s'adapter aux données considérées comme aberrantes, car au lieux de représenter les clusters par un point étant la moyenne des coordonnées des points du cluster, le k-médoïde va lui utiliser un point existant dans le cluster pour représenter le cluster.

Ensuite nous avons choisis de nous intéresser à **DBSCAN** ainsi que OPTICS, ces deux algorithmes ont une approche différente du k-means car ils utilisent une notion de voisinage.
Comme expliqué dans le document (20) l'algorithme DBSCAN n'a besoin que de deux paramètres, le nombre minimum de points dans un cluster et la distance qui permet de définir le voisinage d'un point.
OPTICS quand à lui n'a pas été retenus car sa seule différence avec le DBSCAN était la facultativité de spécifier la distance permettant de définir les voisins d'un point dans les paramètres d'après la référence (23).

La **classification hiérarchique ascendante** n'a pas été retenue car on part d'un nombre de cluster égal au nombre de données , et nous allons essayer de regrouper les points qui sont proches en un nouveau cluster et ainsi de suite jusqu'à arriver au nombre de clusters voulu. Ce qui dans notre cas va donner de très longues exécutions. (voir référence 15).
Ayant suffisamment de méthodes pour effectuer le clustering de nos données nous avons, avec l'approbation de nos tuteurs, délaissés ces méthodes :

- **L'algorithme de maximisation de l'espérance**.
- **Les cartes auto-organisatrices**.

Implémentation et difficultés

Nous avons décidé d'utiliser le langage Java sur Eclipse car nous étions plus familier avec celui-ci. Afin de tester nos algorithmes nos tuteurs nous ont recommandé d'utiliser des fichiers de données sur le site GroupLens. La nature de ces données sont des notes de films de 1 à 5 que les utilisateurs ont donné.
Nous avons utilisé des fichiers de données de différentes tailles mais la finalité de notre projet était de pouvoir utiliser le fichier qui comportait 100 000 de ces données avec 1682 films notés par 943 utilisateurs.
Nous avions plusieurs possibilités pour mettre en place les différents algorithmes de clustering mais nous avons retenu deux choses :

1. Le site dataonfocus *(voir référence 31)* qui propose un algorithme de k-means
2. La librairie JavaML qui propose les trois algorithmes.

Algorithme du site *dataonfocus (voir : 31)*

Ce site propose un algorithme de k-means très abordable, ce dernier comporte des classes bien découpées, claires et précises. Nous avons donc dans un premier temps choisi cet algorithme pour plusieurs raisons :

- Notre tuteur nous avait conseillé cet algorithme
- Il est très simple à comprendre et donc facile à adapter à notre problème

Nous avons, pour commencer, utilisé des algorithmes de K-Means sur un fichier de 100 données seulement que nous avions créé à partir du fichier de 100 000 données. Cela nous

a permis de comprendre le fonctionnement et d'analyser les résultats d'un premier algorithme de clustering.
Comme expliqué précédemment, nous avons 100 000 données, chaque donnée comporte un id utilisateur, un id de film et une note.
Nous souhaitons traiter les films, il faut donc répartir les données sous forme d'une HashMap.
Chaque clé de la HashMap sera l'id du film et la valeur associée sera une autre HashMap qui elle va lier l'id utilisateur à la note.

Une fois ces données récupérées elles sont placées dans la classe points (qui vont être répartis plus tard dans les cluster).

Méthodes de la librairie Java-ML

Cette libraire nous a permis d'avoir un k-means, un k-médoïde et un DBScan.
Tout comme précédemment, il a fallu regrouper les données. Mais cette librairie n'utilise pas de HashMap. Elle utilise un DataSet qui est une collection d'Instances, ces Instances sont des listes à clés qui peuvent contenir un double ou une Instance. Chaque Instance représente donc un point.
L'inconvénient à utiliser une librairie toute faite c'est qu'elle utilise un certain nombre de classes et qu'il est complexe de comprendre et d'affiner tous les paramètres dont nous avons besoin.
De plus le DBScan ne proposant pas de résultats exploitable, nous allons concentrer la suite de ce rapport sur le k-means et le k-médoïdes.

Difficultés rencontrées

L'algorithme récupéré sur le site dataonfocus*(voir 31)* n'était pas adapté à nos données il a fallu changer plusieurs méthodes afin qu'elles correspondent à notre objectif. C'est pour cela que nous avons redéfini :

- La méthode de calcul de distance entre deux points
  Pour cela, nous avons simplement définis une distance euclidienne qui colle à nos donnée en effet si deux films sont représentés comme cela :

utilisateur	Film 1	Film2
1	5	
2	4	1
3	2	5
4		4

  Nous calculons donc la distance uniquement sur les valeurs où les deux films possèdent une note. Ce qui donne : distance = $\sqrt{(4-1)^2+(2-5)^2}$
  Ce choix a été fait suite à un problème lié aux deux algorithmes que nous avons utilisés. En effet puisque nos données ne sont pas complètes nous ne pouvons pas comparer une coordonnée à une autre qui n'est pas renseignée.

De plus la comparaison de deux Instances ne prend pas en compte le fait qu'une instance à une taille de 300 et l'autre une taille de 10 pour les comparer et renvoie donc une erreur.

- La façon de définir les centroïdes :
  La génération de centroïdes se fait de manière aléatoire pour un k-means et à la différence de la librairie Java-ML qui va créer des centroïdes de tailles différentes, nous avons choisi de créer des centroïdes qui avaient la taille maximale (dans notre cas 943) de coordonnées. Par contre ces coordonnées sont générées aléatoirement entre 0 et 5.
  Nous avons choisis cela afin de permettre que les centroïdes soient comparables avec tous les points possibles. En effet puisqu'un centroïde a une "note" renseignée pour tous les utilisateurs, il est comparable à n'importe quel film donné (dans la mesure où nous n'ajoutons pas d'utilisateur).
- La façon de calculer le nouveau centroïde d'un Cluster
  Pour calculer un nouveau centroïde la méthode du k-means consiste à faire la moyenne des coordonnées des points de ce cluster pour définir le nouveau centroïde.
  Dans notre cas il est possible qu'aucun point du cluster ne possède de valeurs pour la dimension 1 par exemple (aucun film n'a été noté par l'utilisateur 1).
  On laissait donc la valeur aléatoire définie au départ.
  Ce choix permet de ne pas laisser de coordonnées vide pour un centroïde, dans le but de toujours pouvoir comparer un centroïde à n'importe quel point.

De plus on constate qu'il y a un phénomène d'absorption des points par un cluster lorsque l'on se retrouve avec un grand nombre de clusters (au delà de 6 clusters). En effet, très rapidement un des clusters va prendre tous les points, et les autres clusters vont se retrouver vides.
Après une réunion avec nos tuteurs nous avons essayé de générer les premiers centroïdes différemment, nous choisissons aléatoirement nos centroïdes parmi les points que nous souhaitons clusteriser (c'est encore en développement actuellement).

Pour le K-Means de Java-ML nous avons rencontré deux problèmes qui se posaient à nous, le premier étant que lors du parcours des notes utilisateurs pour un film, les algorithmes de Java-ML prenaient comme indices les valeurs de 1 jusqu'au nombre d'utilisateurs ayant données une note à ce film, or cela ne peut se faire étant donné que les indices seront différents en fonction du film.
Exemple :

```
196 242 3   881250949
186 302 3   891717742
22  377 1   878887116
244 51  2   880606923
166 346 1   886397596
298 474 4   884182806
115 265 2   881171488
253 465 5   891628467
305 451 3   886324817
6   86  3   883603013
62  257 2   879372434
286 1014    5   879781125
```

Avec idUsers IdFilm Note

Le 2ème problème était lié au fait qu'il manque 95% des données et donc que les instances films sont de tailles différentes car certains films sont notés plus que les autres et pas forcément par les mêmes utilisateurs. La librairie Java-ML ne prend pas cela en compte, et ce problème survient lors du calcul de la distance euclidienne entre deux instances car la fonction va tout d'abord comparer le nombre d'éléments qu'ont les deux instances (utilisateurs ayant votés ce film). Si ils n'ont pas le même nombre alors cela renvoi une erreur. Et si par hasard les deux instances ont le même nombre d'éléments, les utilisateurs de ces instances pourraient être différents et cela pose un problème pour calculer la somme des carrés de ces distances car nous avons besoin des mêmes utilisateurs pour les 2 instances.
Exemple : Calcul de distance entre les films n°154 : (id 1742 : 5, id 45 : 2, id 98 : 1) et n°84 : (id 14 : 4, id 65 : 5, id 74 : 2). Il aurait fallu que le film n°84 ait les utilisateurs 1742, 45 et 98 par exemple.

Nous avions résolu le 1er problème assez rapidement en utilisant une Hashmap mais le 2ème problème s'est avéré plus complexe. Nous avions tout d'abord supprimé la condition qui renvoyait une erreur si la taille des deux instances était différente. Et ensuite nous avons opté pour une méthode qui faisait la moyenne des notes de chacune des deux instances pour effectuer, comme précédemment, la somme des carrés de ces deux moyennes.
Mais après entretien avec nos tuteurs nous en avons conclu que l'on perdrait de l'information avec cette méthode.

Nous avons donc opté pour une méthode parcourant les deux instances et ne prenant en compte que les éléments en communs. Pour réduire le temps de calcul nous avons décidé de ne comparer que les instances ayant au moins 5 éléments en communs. *(voir paragraphe sur l'algorithme de dataonfocus)*

Hypothèses

Afin de valider nos hypothèses, nous avons besoin en plus des données sur lesquelles nous avons effectué la classification d'un certain nombre de données :

- L'erreur moyenne lorsque l'on recommande un film.
- Un indice de controverse qui permet de savoir si un film est controversé ou non.(noté uniquement 1 ou 5)
- L'erreur sur les films proposés aux utilisateurs atypiques.

Ces informations vont nous permettre de répondre aux hypothèses prévues précédemment :

- Les films avec une forte erreur de recommandation sont regroupés dans un ou deux clusters.

Cette hypothèse nous permettrait d'identifier les films sur lesquels le système a besoin d'autres informations afin de recommander ce film car on sait que majoritairement nous nous trompons lorsque ce film est recommandé.

- Les films avec une faible erreur de recommandation sont regroupés dans un ou deux clusters.

En lien avec l'hypothèse précédente, en fonction des résultats de celle ci nous pouvons savoir quelles recommandations marchent toujours ce qui permet de valider, pour les films concernés, la façon de les recommander

- Les films très controversés sont réunis dans un cluster.

L'indice de controverse indique si les notes des utilisateurs sont très différentes (exemple toutes à 5 ou 1 mais pas entre).
Les films très controversés sont généralement soit très appréciés, soit pas du tout. Cela implique que d'une personne à une autre l'erreur possible est très grande.
Identifier les films très controversés permettrait d'anticiper l'erreur de recommandation et peut être de penser pour ce groupe de film à une autre fonction de recommandation qui prendrait d'autres paramètres à prendre en compte.

- Les films très controversés sont répartis de manière égale dans les clusters.

S'ils ne le sont pas alors cela nous permettrait de trouver un clusters qui regroupe la majorité des films avec un grand indice de controverse ainsi que des films avec un petit indice de controverse.

- Il existe un cluster regroupant des films qui plaisent aux utilisateurs atypiques.

Cette hypothèse permettrait à partir des erreurs que l'on fait sur les utilisateurs atypiques d'identifier si l'on fait plus d'erreurs sur un cluster qu'un autre et d'identifier si dans un cluster on ne fait jamais d'erreur lorsque l'on recommande un film aux utilisateurs atypiques en général. Ou encore identifier pour chaque utilisateur atypique le cluster de film sur lequel on ne se trompe jamais. Ce qui permettrait de lui recommander des films en priorité dans ce cluster.

Nous avons choisis de nous intéresser plus particulièrement à la première hypothèse. Car dans le cas des utilisateurs atypiques, nous avons très peu de données par rapport à l'erreur de recommandation.

4- Résultats

Dans cette partie nous allons présenter tout d'abord la validation des clusters, puis les tests de l'hypothèse 1 et pour terminer, les tests de l'hypothèse 2.

Validation des clusters

Afin de valider nos clusters, nous avons besoin de vérifier si la distance intra-cluster et la distance inter-cluster est bonne.
Nous avons donc pour chaque exécution des résultats de ce type :
(avec 4 clusters et 20 itérations)

```
[Cluster: 0]
[Nombre de Points: 1103]
[Cluster: 1]
[Nombre de Points: 198]
[Cluster: 2]
[Nombre de Points: 295]
[Cluster: 3]
[Nombre de Points: 86]
Tableau de distance entre les centroides des clusters :
-------------------------------------------------
|  | 0.0 |  | 4.238121475344254 |  | 0.9108671296182437 |  | 0.7072588657375195 |
-------------------------------------------------
|  | 4.238121475344254 |  | 0.0 |  | 4.1468789141281395 |  | 3.5706782779288986 |
-------------------------------------------------
|  | 0.9108671296182437 |  | 4.1468789141281395 |  | 0.0 |  | 1.037107662817126 |
-------------------------------------------------
|  | 0.7072588657375195 |  | 3.5706782779288986 |  | 1.037107662817126 |  | 0.0 |
Tableau de distance intra cluster
 | 14271.097650167272 |  | 7496.08514911177 |  | 11931.312095737938 |  | 3096.0195770502214 |
```

On peut voir dans cet exemple que le Cluster 0 est très proche des Clusters 2 et 3. De plus toutes les distances intra-Cluster sont très grandes.

Par contre :

```
[Cluster: 0]
[Nombre de Points: 1165]
[Cluster: 1]
[Nombre de Points: 110]
[Cluster: 2]
[Nombre de Points: 224]
[Cluster: 3]
[Nombre de Points: 183]
Tableau de distance entre les centroides des clusters :
-------------------------------------------------
|  | 0.0 |  | 1.5614778983326338 |  | 2.6568651966365584 |  | 5.13676304836164 |
-------------------------------------------------
|  | 1.5614778983326338 |  | 0.0 |  | 1.1837481427922898 |  | 4.129762748185081 |
-------------------------------------------------
|  | 2.6568651966365584 |  | 1.1837481427922898 |  | 0.0 |  | 3.59666688001248 |
-------------------------------------------------
|  | 5.13676304836164 |  | 4.129762748185081 |  | 3.59666688001248 |  | 0.0 |
Tableau de distance intra cluster
 | 17642.623136910286 |  | 3496.968604612493 |  | 7328.300438560959 |  | 8321.134483846561 |
```

Dans ce cas en moyenne les distances sont assez grandes, on pourrait donc valider ce modèle.

Si l'on passe à 5 Clusters, on obtient des résultats plus satisfaisant :

```
[Cluster: 0]
[Nombre de Points: 966]
[Cluster: 1]
[Nombre de Points: 197]
[Cluster: 2]
[Nombre de Points: 190]
[Cluster: 3]
[Nombre de Points: 287]
[Cluster: 4]
[Nombre de Points: 42]
Tableau de distance entre les centroides des clusters :
-----------------------------------------------
|  | 0.0 |  | 0.14823062366450074 |  | 9.241692251866823 |  | 0.785440640947586 |  | 2.5887854601612816 |
-----------------------------------------------
|  | 0.14823062366450074 |  | 0.0 |  | 9.314336560025096 |  | 0.8687126165508134 |  | 2.5172416340356385 |
-----------------------------------------------
|  | 9.241692251866823 |  | 9.314336560025096 |  | 0.0 |  | 8.517122070324756 |  | 11.713598943823952 |
-----------------------------------------------
|  | 0.785440640947586 |  | 0.8687126165508134 |  | 8.517122070324756 |  | 0.0 |  | 3.3551024855032914 |
-----------------------------------------------
|  | 2.5887854601612816 |  | 2.5172416340356385 |  | 11.713598943823952 |  | 3.3551024855032914 |  | 0.0 |
Tableau de distance intra cluster
 | 13015.815701698788 |  | 7324.830599088582 |  | 3685.3476345702798 |  | 11793.912137992835 |  | 909.2679066522397 |
```

En effet on peut voir qu'en moyenne les distances inter-cluster sont plus grandes, on observe une meilleure répartition des données dans les clusters.
Mais avant de valider le modèle il faut faire plusieurs exécutions et calculer la distance intra-cluster moyenne des 5 clusters et faire une moyenne pour toutes les exécutions.

```
[Cluster: 0]
[Nombre de Points: 0]
[Cluster: 1]
[Nombre de Points: 1682]
[Cluster: 2]
[Nombre de Points: 0]
[Cluster: 3]
[Nombre de Points: 0]
[Cluster: 4]
[Nombre de Points: 0]
[Cluster: 5]
[Nombre de Points: 0]
Tableau de distance entre les centroides des clusters :
-----------------------------------------------
|  | 0.0 |  | 0.0 |  | 0.0 |  | 0.0 |  | 0.0 |  | 0.0 |
-----------------------------------------------
|  | 0.0 |  | 0.0 |  | 0.0 |  | 0.0 |  | 0.0 |  | 0.0 |
-----------------------------------------------
|  | 0.0 |  | 0.0 |  | 0.0 |  | 0.0 |  | 0.0 |  | 0.0 |
-----------------------------------------------
|  | 0.0 |  | 0.0 |  | 0.0 |  | 0.0 |  | 0.0 |  | 0.0 |
-----------------------------------------------
|  | 0.0 |  | 0.0 |  | 0.0 |  | 0.0 |  | 0.0 |  | 0.0 |
-----------------------------------------------
|  | 0.0 |  | 0.0 |  | 0.0 |  | 0.0 |  | 0.0 |  | 0.0 |
Tableau de distance intra cluster
 | 0.0 |  | 0.0 |  | 0.0 |  | 0.0 |  | 0.0 |  | 0.0 |
```

A partir de 6 clusters, en fonction des initialisations de centroïdes, on peut voir un phénomène d'absorption, en effet un cluster va au fil des itérations récupérer tous les films et vider tous les autres clusters.
Pour palier à ce problème il faudrait dans un premier temps initialiser les centroïdes d'une façon différente. Au lieu de prendre un centroïde avec des coordonnées au hasard. Prendre

au hasard un point parmi les données à regrouper et définir un centroïde avec ses coordonnées.

Nous avons cette fois ci testés un K-Means différent de celui testé ci dessus et un K-medoids.
Pour obtenir les résultats qui vont suivre, la condition qui empêchait l'arrêt lorsqu'un des clusters était vide du K-Means a été retirée car cela engendrait des programmes qui ne s'arrêtaient pas.

Nous avons effectué tout d'abord deux tests sur le K-Means avec 4 clusters et 10 itérations :

```
Temps clustering : 12850 millisecondes
Cluster 1 : 333
Cluster 2 : 4
Cluster 3 : 1292
Cluster 4 : 53
Number of clusters: 4
Score according to SumOfDistanceIntraCluster: 4.83139581738195E9
Score according to SumOfDistanceInterCluster: 5.1104685083820 35E9
Temps clustering : 13284 millisecondes
Cluster 1 : 333
Cluster 2 : 1125
Cluster 3 : 13
Cluster 4 : 211
Score according to SumOfDistanceIntraCluster: 3.6293524655233 08E9
Score according to SumOfDistanceInterCluster: 6.3125118602384 31E9
```

On peut voir qu'il y a un cluster de 333 pour les 2 tests ainsi qu'un plus gros cluster de 1200 environ. En ce qui concerne les distances intra et inter clusters, elles sont assez inconstantes du premier au deuxième test, le but étant de minimiser la distance intra clusters et maximiser la distance inter clusters.

Nous changeons nos paramètres pour tester avec 5 clusters et 30 itérations cette fois :

```
Nombre iterations : 8
Temps clustering : 12658 millisecondes
Cluster 1 : 333
Cluster 2 : 19
Cluster 3 : 57
Cluster 4 : 23
Cluster 5 : 1250
Score according to SumOfDistanceIntraCluster: 4.336430976263794E9
Score according to SumOfDistanceInterCluster: 5.605433349500248E9
```

On retrouve le cluster de 333 et le cluster plus important de 1200, cependant on peut voir que l'algorithme s'est arrêté avant les 30 itérations, il n'en a fait que 8 car les centroïdes étaient stables. Les distances intra et inter clusters restent assez similaires aux premiers tests.

On effectue 2 derniers tests sur le K-Means avec 8 clusters et 40 itérations :

```
Nombre iterations : 12
Temps clustering : 29198 millisecondes
Cluster 1 : 333
Cluster 2 : 4
Cluster 3 : 201
Cluster 4 : 12
Cluster 5 : 10
Cluster 6 : 1104
Cluster 7 : 6
Cluster 8 : 12
Score according to SumOfDistanceIntraCluster: 3.4400905972435 21E9
Score according to SumOfDistanceInterCluster: 6.5017737285181 09E9
Temps clustering : 134876 millisecondes
Cluster 1 : 333
Cluster 2 : 4
Cluster 3 : 3
Cluster 4 : 6
Cluster 5 : 61
Cluster 6 : 2
Cluster 7 : 3
Cluster 8 : 1270
Score according to SumOfDistanceIntraCluster: 4.6053469222924 79E9
Score according to SumOfDistanceInterCluster: 5.3365174034714 9E9
```

Pour les 2 tests les clusters de 333 et 1200 sont toujours là, les autres clusters varient entre 1 et 200. Le premier test se finit avant les 40 itérations, à 12 itérations. Les distances intra et inter clusters sont similaires aux tests précédents.

Aucun des clusters n'est vide parmi ces tests, on peut tout de même admettre que ce K-Means n'est pas vraiment efficace car il ne créer que 2 gros clusters et les distances intra et inter clusters ne sont pas très optimales.

Passons maintenant au K-Médoïdes, avec tout d'abord 2 tests avec comme paramètres : 4 clusters et 10 itérations :

```
Temps clustering : 5103 millisecondes
Cluster 1 : 616
Cluster 2 : 567
Cluster 3 : 140
Cluster 4 : 359
Score according to SumOfDistanceIntraCluster: 2.974654820684874E9
Score according to SumOfDistanceInterCluster: 6.967209505072284E9
Temps clustering : 5122 millisecondes
Cluster 1 : 1679
Cluster 2 : 0
Cluster 3 : 0
Cluster 4 : 3
Score according to SumOfDistanceIntraCluster: 9.91377782890771E9
Score according to SumOfDistanceInterCluster: 2.808649685299431E7
```

Pour le premier test on peut voir que les clusters sont plus ou moins remplis équitablement, et les distances intra et inter clusters sont très bonnes. Cependant pour le deuxième test, un seul cluster possède tous les éléments, on peut en déduire qu'il y a eu un phénomène d'absorption, et les distances intra et inter sont donc mauvaises.

Nous testons maintenant la méthode sur 8 clusters et 40 itérations :

```
Temps clustering : 36618 millisecondes
Cluster 1 : 1054
Cluster 2 : 185
Cluster 3 : 3
Cluster 4 : 160
Cluster 5 : 5
Cluster 6 : 91
Cluster 7 : 61
Cluster 8 : 123
Score according to SumOfDistanceIntraCluster: 5.3080274720370O4E9
Score according to SumOfDistanceInterCluster: 4.633836853723819E9
Temps clustering : 28293 millisecondes
Cluster 1 : 566
Cluster 2 : 0
Cluster 3 : 102
Cluster 4 : 249
Cluster 5 : 293
Cluster 6 : 0
Cluster 7 : 199
Cluster 8 : 273
Score according to SumOfDistanceIntraCluster: 2.094046226159725E9
Score according to SumOfDistanceInterCluster: 7.847818099595355E9
```

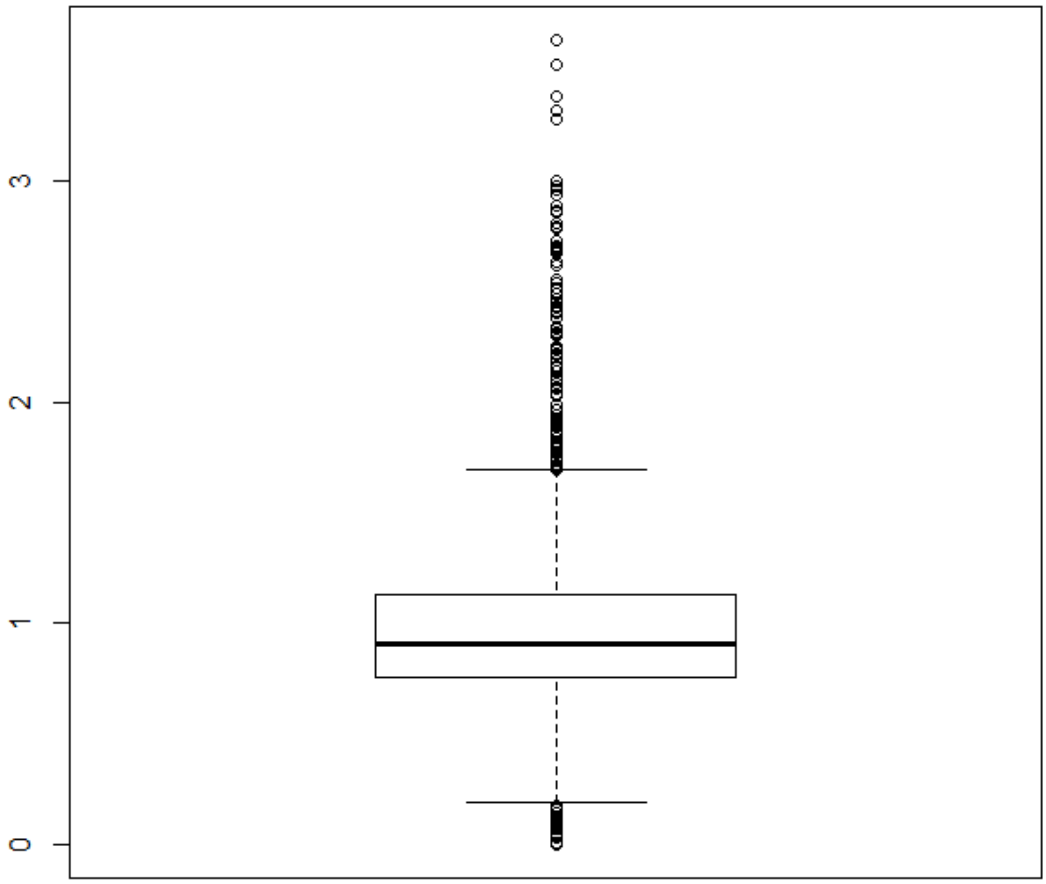
Pour le premier test il y a encore un phénomène d'absorption par le cluster 1 mais aucun des clusters ne sont vides. Pour le deuxième test il y a par contre deux clusters vides ce qui nous permet d'identifier 6 clusters sur les 8 voulus. De plus les distances intra et inter clusters sont très mauvaises.

Test de l'hypothèse 1

L'hypothèse que nous allons étudier est donc :

- Les films avec une forte erreur de recommandation sont regroupés dans un ou deux cluster.

Tout d'abord, il faut définir ce que nous appelons une forte erreur de recommandation. Nous allons donc regarder les données que nous avons par rapport à cette erreur.
Pour cela nous allons regarder une boite à moustache afin de voir comment sont réparties nos données :



De plus si on affiche les valeurs des différents quartiles on obtient :

```
       0%       25%       50%       75%      100%
0.0000000 0.7558153 0.9115768 1.1325830 3.6394046
```

On peut donc voir que la moitié de nos données se trouvent au dessus d'une erreur de 1.
On peut aussi voir que 25% de nos données se trouvent entre 1.13 et 3.64 avec 75% en dessous de 1.13.

Nous avons donc décidé qu'un film avec un fort taux d'erreur se trouverait donc au dessus de **2.39**
qui est la **moyenne** entre **3.64 et 1.13**.
Nous allons donc récupérer la liste des films donc le taux d'erreur est supérieur ou égal à **2.39**.
Ensuite nous allons regarder leurs répartitions dans les clusters ce qui donne :

```
Répartition des Films a fort taux d'erreur par cluster
 | 34 |  | 1 |  | 1 |  | 0 |  | 0 |
```

On peut voir que la population correspondant à ce seuil représente **2%** de notre population.
Nous avons donc cherché un autre seuil pouvant avoir un nombre de film correspondant plus élevé.

En choisissant un **seuil de 2**, nous avions 71 film qui correspondaient à ce critère ce qui équivaut à 4% de la population totale. C'est toujours très faible. Nous avons donc choisi de placer **le seuil à 1.5**.
Cela nous donne une répartition de ce type :

```
Répartition des Films a fort taux d'erreur par cluster
  | 164 |   | 3 |   | 3 |   | 4 |   | 5 |
```

En moyenne on retrouve toujours ce résultat avec la configuration de 5 clusters et 20 itérations.
On peut donc dire que les films avec un fort taux d'erreur moyen sont répartis dans le même cluster.

Notre hypothèse est donc valide.

Test de l'hypothèse 2

L'hypothèse que nous allons étudier est donc :

- Les films avec une faible erreur de recommandation sont regroupés dans un ou deux clusters.

Tout d'abord, il faut définir ce que nous appelons une faible erreur de recommandation. Nous allons donc regarder les données que nous avons par rapport à cette erreur.
La répartition de nos quartiles n'ayant toujours pas changé :

```
       0%        25%        50%        75%       100%
0.0000000 0.7558153 0.9115768 1.1325830 3.6394046
```

On peut donc voir que **la moitié de nos données** se trouvent **en dessous d'une erreur de 1**.
Et que **25% des données** ont une erreur **entre 0 et 0.75**. On peut donc dire qu'une faible erreur se trouve en dessous de **0.38.**
Nous allons donc récupérer la liste des films donc le taux d'erreur est inférieur ou égal à **0.38**.
Ensuite nous allons regarder leurs répartitions dans les clusters ce qui donne :

```
Répartition des Films a fort faible d'erreur par cluster
  | 110 |   | 2 |   | 0 |   | 1 |   | 1 |
```

En moyenne on retrouve toujours ce résultat avec la configuration de 5 clusters et 20 itérations.
On peut donc dire que les films avec un faible taux d'erreur moyen sont répartis dans le même cluster. Notre hypothèse est donc valide.

5- Discussions

On peut voir qu'avec les configurations citées précédemment, il est possible de valider nos hypothèses.

Nous pourrions donc affirmer qu'il est possible grâce à une classification des films d'identifier les films sur lesquels nous avons un faible taux d'erreur de recommandation et ceux pour lesquels il y a un fort taux. Nous pourrions aussi affirmer qu'il est possible grâce à une classification des films d'identifier les films sur lesquels nous sommes sur d'avoir assez d'informations afin de les recommander avec précision et les films sur lesquels nous avons besoin de plus d'informations car actuellement les données ne permettent pas d'avoir une bonne recommandation.

Il faut prendre en compte plusieurs choses. D'une part nous avons considéré ces deux hypothèses séparément. Or que se passe t'il si les films que nous avons identifiés précédemment se retrouvent dans le même cluster ?
Nos hypothèses indiqueraient que ce cluster est à la fois un cluster précis quant à ces recommandation et imprécis. Il faut donc considérer les deux hypothèses ensemble et non séparément et ajouter donc un test sur la cohabitation au sein d'un cluster de ces films.
D'une autre part nous avons effectué les tests par rapport aux hypothèses avec une configuration donnée de cluster mais qui n'est pas complètement satisfaisante.
En effet si l'on étudie la répartition des films dans les clusters on se rend compte que la répartition des films est déséquilibrée.
Par exemple pour cette exécution du programme :

```
[Cluster: 0]
[Nombre de Points: 1021]
[Cluster: 1]
[Nombre de Points: 75]
[Cluster: 2]
[Nombre de Points: 230]
[Cluster: 3]
[Nombre de Points: 165]
[Cluster: 4]
[Nombre de Points: 191]
```

On a donc une répartition des films très déséquilibrée.
On observe cette répartition pour les films à fort et faible taux d'erreur :

```
Répartition des Films a fort taux d'erreur par cluster
 | 34 |  | 1 |  | 1 |  | 0 |  | 0 |
Répartition des Films a fort faible d'erreur par cluster
 | 110 |  | 2 |  | 0 |  | 1 |  | 1 |
```

Premièrement comme évoqué précédemment, ils font partie du même cluster indiqué Cluster 0.
Mais ce Cluster possède une très grande population, il est donc normal qu'il aie plus de chance d'avoir les films qui possèdent les attributs qui nous intéressent.

Il faudrait donc valider un autre type de Cluster afin d'améliorer les résultats de nos hypothèses.

6- Conclusion

La mise en oeuvre de ce projet tutoré nous a permis de découvrir le domaine du clustering qui nous était encore peu familier. Nous l'avions vaguement abordé en cours Intelligence Artificielle Fouille de Données, et en Mémoire et Apprentissage Numérique. Le fait de travailler et se spécialiser dans ce domaine a été très enrichissant pour nous. Ce travail nous a montré l'importance de choisir les bonnes méthodes de clustering lorsque l'on travaille sur des fichiers de données conséquents pour optimiser au mieux le temps de calcul et de mémoire.

Nous avons donc abordé la problématique de départ qui est la recommandation de qualité pour les utilisateurs atypiques. L'idée de ce projet était de définir et de valider de manière expérimentale une ou plusieurs hypothèses quant aux systèmes de recommandations. Nous avons donc voulu montrer qu'en regroupant les films en clusters à l'aide de méthodes différentes existantes, il était possible d'identifier des améliorations possibles au sein de systèmes de recommandations pour que les utilisateurs atypiques puissent recevoir des recommandations de qualité.

Au terme de ce projet, par manque de temps nous n'avons malheureusement pas pu tester toutes les hypothèses émises en partie de discussion car nous n'avons pas complètement validé nos méthodes de clustering mais nous avons pu en tester et valider deux.

7- Bibliographie

Fiche de Projet tutoré

1) http://mathinfo.univ-lorraine.fr/sites/mathinfo.univ-lorraine.fr/files/users/documents/SCA/projtut/2016-2017/sujets/pt_brun_moutonsnoirs.pdf

Les systèmes de recommandations

2) https://fr.wikipedia.org/wiki/Syst%C3%A8me_de_recommandation

Le clustering

- Reconnaissances des formes
3) http://www.math-info.univ-paris5.fr/~lomn/Cours/RF/Material/RF2005.pdf
- Analyse de données spatiales
4) http://invs.santepubliquefrance.fr/publications/2011/methodes_statistiques_systeme_information/rapport_methodes_statistiques_si_geographique.pdf
- Traitement d'images
5) https://tcts.fpms.ac.be/cours/1005-07-08/speech/projects/2005/dhondt_elkhayati.pdf
- Recherche d'informations
6) https://www.cairn.info/revue-document-numerique-2010-1-page-9.htm
- Web mining
7) http://www-sop.inria.fr/axis/personnel/Doru.Tanasa/papers/sfc03.pdf

Description des méthodes

8) https://www-lmgm.biotoul.fr/enseignements/M2Pro_Bioinfo/Clustering.pdf
- K-means
9) https://fr.wikipedia.org/wiki/K-moyennes
10) https://www.xlstat.com/fr/solutions/fonctionnalites/classification-par-la-methode-des-nuees-dynamiques-k-means
11) http://www.tsi.enst.fr/pages/enseignement/ressources/beti/hyste-dyn/node2.html
12) https://fr.mathworks.com/help/stats/kmeans.html?requestedDomain=www.mathworks.com
- K-medoids
13) https://fr.wikipedia.org/wiki/Algorithme_des_k-m%C3%A9do%C3%AFdes
14) https://stats.stackexchange.com/questions/156210/difference-between-k-means-and-k-medoid
15) https://pdfs.semanticscholar.org/5d87/ef6efa31e9aa4f0b63aa1dabe31f9693a93b.pdf
16) http://dspace.univ-tlemcen.dz/bitstream/112/6391/1/Application-de-k-means.pdf
- CAH
17) https://fr.wikipedia.org/wiki/Regroupement_hi%C3%A9rarchique
18) http://larmarange.github.io/analyse-R/classification-ascendante-hierarchique.html
19) https://www.xlstat.com/fr/solutions/fonctionnalites/classification-ascendante-hierarchique-cah
20) http://www.coryent.com/compaclassif.html
- Algo EM
21) https://fr.wikipedia.org/wiki/Algorithme_esp%C3%A9rance-maximisation
22) http://www.pacea.u-bordeaux1.fr/IMG/pdf/algo-em.pdf
23) http://archimede.mat.ulaval.ca/theses/I-Michaud_05.pdf
- DBSCAN
24) https://fr.wikipedia.org/wiki/DBSCAN

25) http://www.sthda.com/english/wiki/dbscan-density-based-clustering-for-discovering-clusters-in-large-datasets-with-noise-unsupervised-machine-learning
26) http://devezeb.free.fr/downloads/ecrits/datamining.pdf
- OPTICS
27) https://fr.wikipedia.org/wiki/OPTICS
28) http://dictionnaire.sensagent.leparisien.fr/OPTICS%20algorithm/en-en/
- Cartes auto-organisatrices
29) https://fr.wikipedia.org/wiki/Carte_auto_adaptative
30) https://tel.archives-ouvertes.fr/tel-01142849/document

Données utilisées

31) https://grouplens.org/datasets/movielens/

Utilisation de méthodes existantes pour le K-means

32) http://www.dataonfocus.com/k-means-clustering-java-code/
33) https://tel.archives-ouvertes.fr/tel-00195779/file/these_boubou.pdf
34) http://webia.lip6.fr/~rifqi/COURS2001-2002/IA/Clustering.pdf