



École d'Ingénierie Digitale
et d'Intelligence Artificielle

RAPPORT

Système de recommandation

Réalisé par :
Firdawse Guerbouzi - Khadija Bayoud -
Machrouh Mohammed - Ravel Wourougou

Encadré par:
Mme Chougrad Hiba



SOMMAIRE

-
- 01.** Introduction
 - 02.** Système de recommandation et ses différents types
 - 03.** Exploitation de la base de données
 - 04.** Recommandation par méthode traditionnelle
 - 05.** Recommandation par KNN
 - 06.** Recommandation par Kmeans
 - 07.** Recommandation par Neural network
 - 08.** Démonstration par Application Web
 - 09.** Bibliographie

Introduction

Dans cette vie bien remplie, les gens n'ont pas le temps de chercher l'article qu'ils souhaitent même avec un petit effort. Ainsi, le système de recommandation est devenu un élément important pour les aider à faire le bon choix pour ce qu'ils souhaitent. Étant donné que les données augmentent jour après jour et à cette époque avec une base de données aussi volumineuse, il est même devenu une tâche difficile de trouver un élément plus pertinent qui nous intéresse, car souvent nous ne pouvons pas rechercher un élément qui nous intéresse avec juste un titre et même parfois c'est plus difficile. Ainsi, le système de recommandation nous aide à fournir un article le plus pertinent à l'individu disponible dans notre base de données.

MovieLens¹ est une dataset créée par 138493 utilisateurs entre Janvier 09, 1995 et Mars 31, 2015. Elle a été générée en Octobre 17, 2016. Elle décrit les évaluations de 27278 films par 138493 utilisateurs. Elle contient 20 Millions observations.

Le but de ce projet est de construire un système de recommandation en utilisant différentes algorithmes d'apprentissage automatique (machine learning) et d'apprentissage profond (deep learning) en se basant sur les données MovieLens.

Dans notre projet, nous utiliserons les deux datasets '**rating.csv**' et '**movie.csv**'.

rating.csv : contient les évaluations des films par les utilisateurs.

- **userId**
- **moviId**
- **rating**
- **timestamp**

movie.csv : contient les informations sur les films.

- **moviId**
- **title**
- **genres**

Qu'est-ce qu'un système de recommandation ?

Sur la base des comportements antérieurs , il prédit la probabilité qu'un utilisateur préfère un article.

Par exemple, Netflix utilise un système de recommandation. Il suggère aux gens de nouveaux films en fonction de leurs activités passées(comme regarder , voter des films...)

Le but des systèmes de recommandation est de recommander de nouvelles choses qui ne sont pas vues auparavant par les gens.

I. Filtrage collaboratif (Collaborative Filtering)

C'est la recommandation des produits basés sur les intérêts d'un grand groupe d'utilisateurs. Cette méthode essaie de trouver un groupe d'utilisateurs qui possède les mêmes goûts et préférences de l'utilisateur cible. Ainsi, il utilise ce groupe pour recommander leurs produits. En partant de l'hypothèse que les utilisateurs avec des goûts similaires ont les mêmes préférences.

1. Recommandation basée sur l'utilisateur

Dans cette approche, on construit une matrice A : [Utilisateur x Produit]

A	Item 1	Item 2	Item 3	Item 4	Item 5
User 1	9	3	?	7	10
User 2		2	6	7	9
User 3	9	3	9	7	
User 4	6	6		2	1
User 5		7	9	3	4

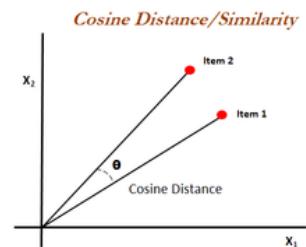
Dans cette matrice, on trouve qu'User 1 et User 2 sont corrélés (3 sur 5 produits ont le même score). C'est ainsi qu'on recommande à l'un des utilisateurs les produits préférés du deuxième. Mais comment calculer la corrélation ou la similarité entre deux utilisateurs ?

Les méthodes de similarité :

• Cosine similarity

$$\mu = \cos(\text{User 1}, \text{User 2}) = \cos(\alpha) = \frac{\text{User 1} \cdot \text{User 2}}{\|\text{User 1}\| \|\text{User 2}\|}$$

La corrélation de deux utilisateurs est importante pour des valeurs de μ aussi importantes



• Jaccard similarity

$$J(\text{User 1}, \text{User 2}) = \frac{|\text{User 1} \cap \text{User 2}|}{|\text{User 1} \cup \text{User 2}|}$$

Cette méthode consiste à ignorer les scores

L'avantage :

- Pas d'information sur les produits et leurs spécialités (features)

Les inconvénients :

- Un nouvel utilisateur commence sans avoir une recommandation puisque ses scores sont tous à zéro (de même pour les nouveaux produits).
- Dans le cas d'un large nombre de produits, la matrice devient creuse. Alors, il est difficile de "matcher" les utilisateurs qui corrélés (qui ont les mêmes préférences).

I. Filtrage collaboratif (Collaborative Filtering)

2. Recommandation basée sur le produit

Dans cette approche, on construit aussi la même matrice A. Mais la différence se présente par corrélation entre les produits. On cherche les produits qui sont potentiellement corrélés. Et on offre un produit qui est rattaché aux autres produits (ayant le plus de degré de corrélation)

Les avantages :

- Pas d'information sur les produits et leurs spécialités (features).
- Trouver une corrélation entre un nombre limité de produit est mieux de trouver une corrélation entre nombre très grand d'utilisateurs.
- Les produits sont plus simple à corrélérer que les utilisateurs et elle est plus significative.

L'inconvénient :

- Un nouvel utilisateur commence sans avoir une recommandation puisque ses scores sont tous à zéro (de même pour les nouveaux produits).
- Dans le cas d'un large nombre de produits, la matrice devient creuse. Alors, il est difficile de "matcher" les utilisateurs qui corrélés (qui ont les mêmes préférences).

II. Filtrage basé sur le contenu (content-based filtering)

Ce type de système de recommandation se base sur des profils. En fait, on construit des profils pour les utilisateurs et ainsi que pour les produits.

Par exemple, on construit un profil d'un utilisateur A qui préfère les séries dont les genres préférés sont actions et romances (cas de Netflix). Et, on essaie de recommander des produits qui de la même section que A préfère. (On ne se base plus sur les opinions des autres utilisateurs).

Les avantages :

- Pas besoin de données sur les autres utilisateurs.
- Possibilité de recommander des produits qui ne sont pas populaires ou nouveaux.
- Possibilité de recommander aux utilisateurs ayant un goût unique ou rare.

Les inconvénients :

- Trouver les bons "features" n'est pas toujours facile.
- Comment créer un profil pour les nouveaux utilisateurs ?

Exploration de la dataset

Avant de présenter les différents algorithmes utilisés et leurs performances, nous commençons d'abord par l'exploration de notre dataset. Cette étape nous permet d'avoir une vue générale sur les données et les caractéristiques ainsi de les visualiser pour bien comprendre leurs nature.

rating.csv

	userId	movieId	rating	timestamp
0	1	2	3.5	2005-04-02 23:53:47
1	1	29	3.5	2005-04-02 23:31:16
2	1	32	3.5	2005-04-02 23:33:39
3	1	47	3.5	2005-04-02 23:32:07
4	1	50	3.5	2005-04-02 23:29:40

```
ratings.isnull().any()
```

```
userId      False
movieId     False
rating      False
timestamp   False
dtype: bool
```

movie.csv

	movieId	title	genres
0	1	Toy Story (1995)	Adventure Animation Children Comedy Fantasy
1	2	Jumanji (1995)	Adventure Children Fantasy
2	3	Grumpier Old Men (1995)	Comedy Romance
3	4	Waiting to Exhale (1995)	Comedy Drama Romance
4	5	Father of the Bride Part II (1995)	Comedy

```
movies.isnull().any()
```

```
movieId      False
title       False
genres      False
dtype: bool
```

Nous constatons qu'il n'y a pas de valeurs manquantes.

La popularité des genres

Nous visualisons la popularité des genres par deux méthodes:

1. Le nuage de mots fourni par WordCloud
2. Histogramme à bâtons (bar chart)



figure 1 : Genres word cloud

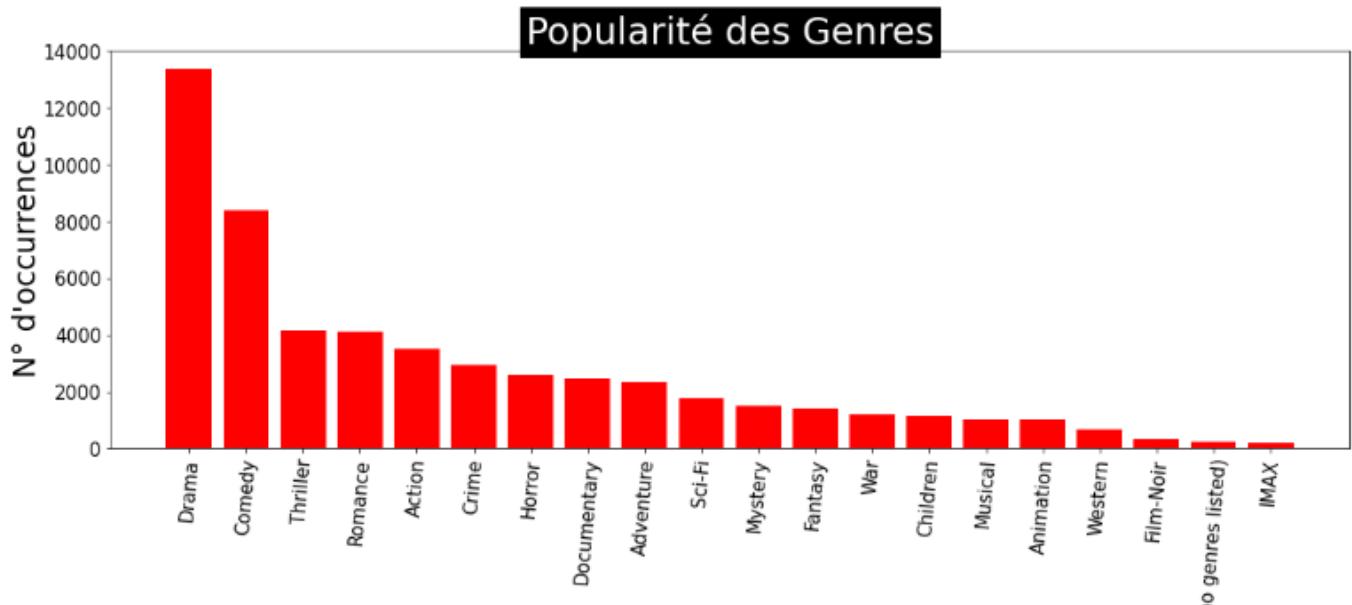


figure 2 : Genres bar chart

Nous constatons que le genre **Drama** constitue le **mode** de la dataset.

Top 10 films notés

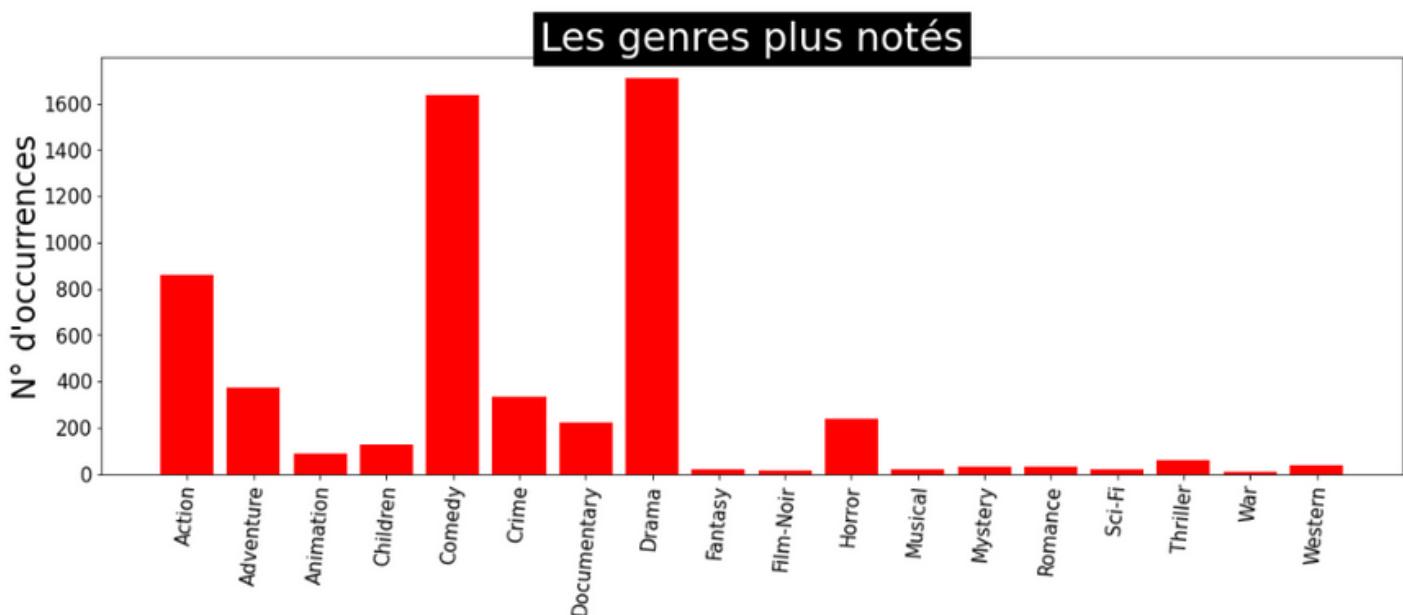
Les films les plus notés nous donnent une idée sur les préférences des utilisateurs.

```
top10Movies = users_fav_movies[users_fav_movies['rating'] == 5.0][:10]
top10Movies = movies[movies['movieId'].isin(top10Movies['movieId'])]['title']
top10Movies
```

```
61                  Mr. Holland's Opus (1995)
69                  From Dusk Till Dawn (1996)
257             Star Wars: Episode IV - A New Hope (1977)
263             Legends of the Fall (1994)
476             Jurassic Park (1993)
537             Blade Runner (1982)
4897    Lord of the Rings: The Fellowship of the Ring, ...
5853    Lord of the Rings: The Two Towers, The (2002)
7041    Lord of the Rings: The Return of the King, The...
7860                 Freaks (1932)
```

Nous constatons que le film **Mr. Holland's Opus (1995)** est le plus noté par les la plupart des utilisateurs.

Les genres les plus notés



Recommandation des films basée sur le produit

	movieId	title	userId	rating
0	1	Toy Story (1995)	3	4.0
1	1	Toy Story (1995)	6	5.0
2	1	Toy Story (1995)	8	4.0
3	1	Toy Story (1995)	10	4.0
4	1	Toy Story (1995)	11	4.5
5	1	Toy Story (1995)	12	4.0
6	1	Toy Story (1995)	13	4.0
7	1	Toy Story (1995)	14	4.5

Nous avons créé une jointure de données afin d'analyser les 4 paramètres **movie id**, **title**, **user id** et **rating**

Le nombre d'échantillons dans la base de données est de 20 millions, c'est trop!! Par conséquent, Nous travaillerons sur 1 million de données.

- pivot table

la 2ème étape est de créer un tableau croisé dynamique afin que les lignes soient des utilisateurs et que les colonnes soient des films. Et les valeurs sont notées(rating).

Faisons maintenant un scénario, nous avons un site Web de films et le film "Bad Boys (1995)" est regardé et évalué par les gens. **La question est de savoir quel film recommandons-nous à ces personnes qui ont regardé le film "Bad Boys (1995)".**

Afin de répondre à cette question, nous trouverons des similitudes entre le film "Bad Boys (1995)" et d'autres films.

```
movie_watched = pivot_table["Bad Boys (1995)"]
similarity_with_other_movies = pivot_table.corrwith(movie_watched) # find correlation between "Bad Boys (1995)" and other movies
similarity_with_other_movies = similarity_with_other_movies.sort_values(ascending=False)
similarity_with_other_movies.head()
```

```
title
Bad Boys (1995)           1.000000
Headless Body in Topless Bar (1995)    0.723747
Last Summer in the Hamptons (1995)    0.607554
Two Bits (1995)            0.507008
Shadows (Cienie) (1988)      0.494186
dtype: float64
```

On peut en conclure que nous devons recommander le film "Headless Body in Topless Bar (1995)" aux personnes qui ont regardé "Bad Boys (1995)".

KNN

Introduction

L'algorithme KNN (K-plus proches voisins) est un algorithme de base: définissez une métrique de distance entre les éléments de votre ensemble de données et recherchez les K éléments les plus proches. Vous pouvez ensuite utiliser ces éléments pour prédire certaines propriétés d'un élément de test, en les faisant "voter" d'une manière ou d'une autre.

Nous allons construire un système pour classer les utilisateurs en groupes de même intérêt en utilisant l'algorithme KNN pour la recommandation de film.

Principe du KNN

La méthode des K plus proches voisins (KNN) a pour but de classifier des points cibles (classe méconnue) en fonction de leurs distances par rapport à des points constituant un échantillon d'apprentissage (c'est-à-dire dont la classe est connue a priori).

KNN est une approche de classification supervisée intuitive, souvent utilisée dans le cadre du machine learning. Il s'agit d'une généralisation de la méthode du voisin le plus proche (NN). NN est un cas particulier de KNN, où $k = 1$.

Etapes de l'algorithme

1. Charger les données
2. Initialiser k au nombre de plus proches voisins choisi
3. Pour chaque exemple dans les données:
 - 3.1 Calculer la distance entre notre requête et l'observation itérative actuelle de la boucle depuis les données.
 - 3.2 Ajouter la distance et l'indice de l'observation concernée à une collection ordonnée de données
4. Trier cette collection ordonnée contenant distances et indices de la plus petite distance à la plus grande (dans ordre croissant).
5. Sélectionner les k premières entrées de la collection de données triées (équivalent aux k plus proches voisins)
6. Obtenir les étiquettes des k entrées sélectionnées
7. Si régression, retourner la moyenne des k étiquettes
8. Si classification, retourner le mode (valeur la plus fréquente/commune) des k étiquettes
9. Calculer l'efficacité

Calcul de la distance entre voisins

Le calcul de la distance entre les voisins dans l'algorithme du KNN peut se faire de différentes manière. On peut utiliser la fameuse mesure euclidienne, la distance cosine, ou plein d'autres distances calculées auparavant. Leur formules sont les suivantes:

• Distance de Minkowski

C'est une métrique destinée aux espaces vectoriels à valeurs réelles. Nous ne pouvons calculer la distance de Minkowski que dans un espace vectoriel normé, c'est-à-dire dans un espace où les distances peuvent être représentées comme un vecteur qui a une longueur et les longueurs ne peuvent pas être négatives.

La métrique de distance doit satisfaire à quelques conditions :

Non-négativité : $d(x, y) \geq 0$

Identité : $d(x, y) = 0$ si et seulement si $x = y$

Symétrie : $d(x, y) = d(y, x)$

Inégalité triangulaire : $d(x, y) + d(y, z) \geq d(x, z)$

$$\left(\sum_{i=1}^n |x_i - y_i|^p \right)^{1/p}$$

Cette formule ci-dessus pour la distance de Minkowski est sous forme généralisée et nous pouvons la manipuler pour obtenir différentes métriques de distance.

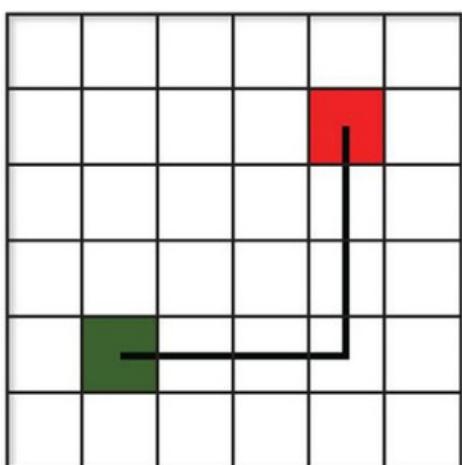
La valeur p dans la formule peut être manipulée pour nous donner différentes distances comme :

$p = 1$, lorsque p est défini sur 1, nous obtenons la distance de Manhattan

$p = 2$, lorsque p est fixé à 2, nous obtenons la distance euclidienne

• Distance de Manhattan

Cette distance est également connue sous le nom de distance en taxi ou distance de pâté de maisons, c'est parce que la façon dont cette distance est calculée. La distance entre deux points est la somme des différences absolues de leurs coordonnées cartésiennes.



Manhattan Distance

Comme nous le savons, nous obtenons la formule de la distance de Manhattan en remplaçant p=1 dans la formule de distance de Minkowski.

$$d = \sum_{i=1}^n |\mathbf{x}_i - \mathbf{y}_i|$$

Supposons que nous ayons deux points comme indiqué sur l'image, le rouge (4,4) et le vert (1,1).

Et maintenant, nous devons calculer la distance en utilisant la métrique de distance de Manhattan.

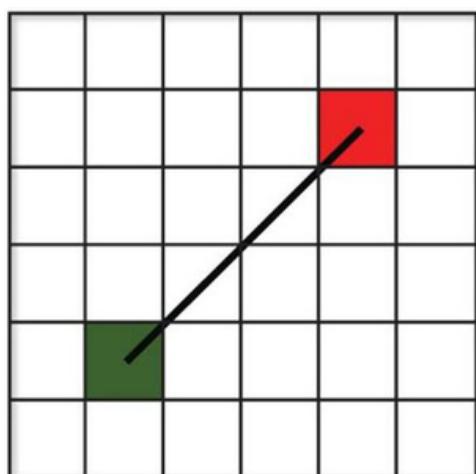
Nous obtiendrons,

$$r_e = |4-1| + |4-1| = 6$$

Cette distance est préférée à la distance euclidienne lorsque nous avons un cas de haute dimensionnalité.

- **Distance euclidienne**

Cette distance est la plus largement utilisée car il s'agit de la métrique par défaut que la bibliothèque SKlearn de Python utilise pour K-Nearest Neighbour. C'est une mesure de la vraie distance en ligne droite entre deux points dans l'espace euclidien.



Euclidean Distance

Il peut être obtenu en définissant la valeur de p égale à 2 dans la métrique de distance de Minkowski.

Supposons maintenant que nous ayons deux points le rouge (4,4) et le vert (1,1). Et maintenant, nous devons calculer la distance en utilisant la métrique de distance euclidienne.

Nous obtiendrons,

• Distance cosinus

Cette mesure de distance est principalement utilisée pour calculer la similarité entre deux vecteurs. Il est mesuré par le cosinus de l'angle entre deux vecteurs et détermine si deux vecteurs pointent dans la même direction. Il est souvent utilisé pour mesurer la similarité des documents dans l'analyse de texte. Lorsqu'elle est utilisée avec KNN, cette distance nous donne une nouvelle perspective sur un problème commercial et nous permet de trouver des informations cachées dans les données que nous n'avons pas vues en utilisant les deux matrices de distance ci-dessus.

Il est également utilisé dans l'analyse de texte pour trouver des similitudes entre deux documents en fonction du nombre de fois qu'un ensemble particulier de mots y apparaît.

La formule pour la distance cosinus est :

$$\cos \theta = \frac{\vec{a} \cdot \vec{b}}{\|\vec{a}\| \cdot \|\vec{b}\|}$$

En utilisant cette formule, nous obtiendrons une valeur qui nous renseigne sur la similitude entre les deux vecteurs et $1 - \cos \theta$ nous donnera leur distance cosinus.

En utilisant cette distance, nous obtenons des valeurs comprises entre 0 et 1, où 0 signifie que les vecteurs sont similaires à 100 % et 1 signifie qu'ils ne sont pas du tout similaires.

• Distance de Jaccard

Le coefficient Jaccard est une méthode de comparaison similaire à la similarité cosinus en raison de la façon dont les deux méthodes comparent un type d'attribut réparti entre toutes les données. L'approche Jaccard examine les deux ensembles de données et trouve l'incident où les deux valeurs sont égales à 1. Ainsi, la valeur résultante reflète le nombre de correspondances 1 à 1 par rapport au nombre total de points de données. Ceci est également connu sous le nom de fréquence de correspondance 1 à 1, ce que recherche la similarité cosinus, la fréquence à laquelle un certain attribut se produit.

Il est extrêmement sensible aux petites tailles d'échantillons et peut donner des résultats erronés, en particulier avec de très petits ensembles de données avec des observations manquantes.

La formule de l'indice de Jaccard est :

$$J(A, B) = \frac{|A \cap B|}{|A \cup B|} = \frac{|A \cap B|}{|A| + |B| - |A \cap B|}$$

La distance de Jaccard est le complément de l'indice de Jaccard et peut être trouvée en soustrayant l'indice de Jaccard de 100 %, ainsi la formule de la distance de Jaccard est :

$$D(A,B) = 1 - J(A,B)$$

- **Distance de Hamming**

La distance de Hamming est une mesure permettant de comparer deux chaînes de données binaires. Lors de la comparaison de deux chaînes binaires de longueur égale, la distance de Hamming est le nombre de positions de bits dans lesquelles les deux bits sont différents. La méthode de distance de Hamming examine l'ensemble des données et trouve quand les points de données sont similaires et dissemblables un à un. La distance de Hamming donne le résultat du nombre d'attributs différents.

Ceci est principalement utilisé lorsque vous encodez vos données à chaud et que vous devez trouver les distances entre les deux vecteurs binaires.

Supposons que nous ayons deux chaînes "ABCDE" et "AGDDF" de même longueur et que nous voulions trouver la distance de Hamming entre celles-ci. Nous allons aller lettre par lettre dans chaque chaîne et voir si elles sont similaires ou non, comme les premières lettres des deux chaînes sont similaires, puis la seconde n'est pas similaire et ainsi de suite.

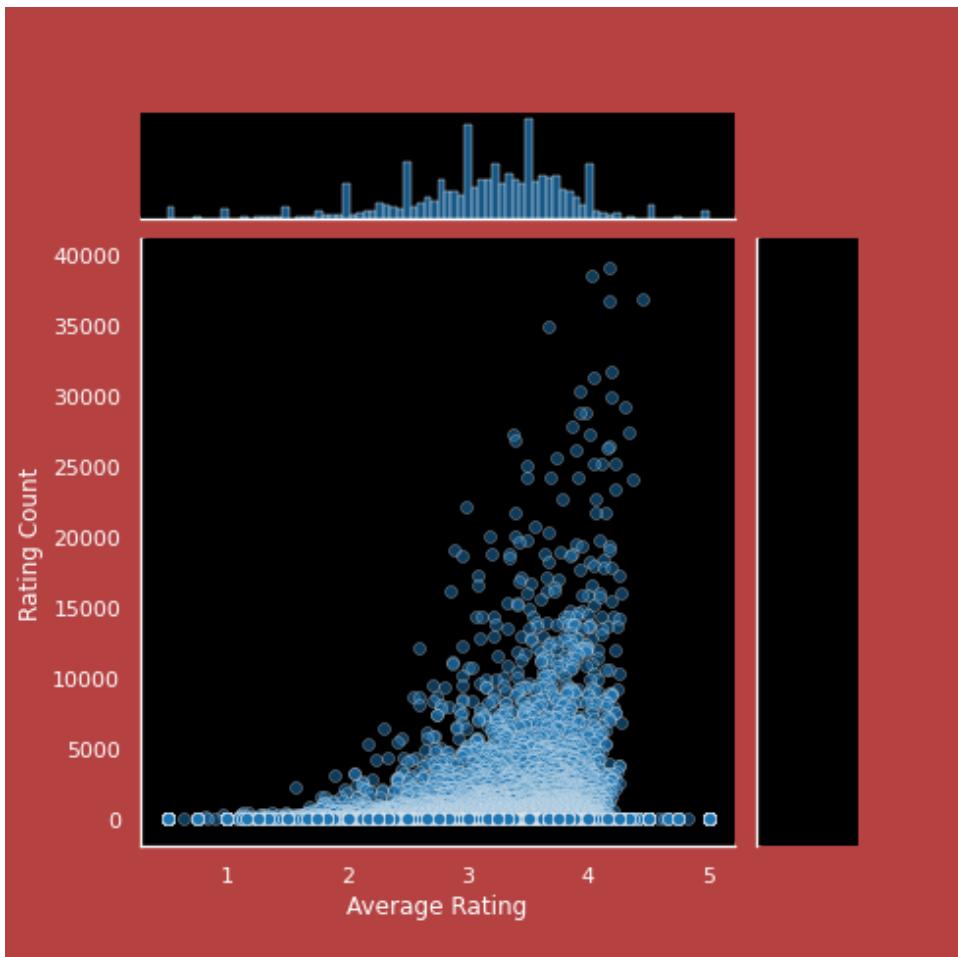
ABCDE et AGDDF

Lorsque nous aurons terminé, nous verrons que seules deux lettres marquées en rouge étaient similaires et trois étaient différentes dans les chaînes. Par conséquent, la distance de Hamming ici sera de 3. Notez que plus la distance de Hamming entre deux cordes est grande, plus ces cordes seront dissemblables (et vice versa).

Notre code utilisera principalement la distance Cosine car c'est la plus performante dans le modèle KNN

Exploration et nettoyage de la dataset

Comme nous l'avons stipulé précédemment, après exploration poussée, nous avons remarqué qu'il n'y a aucune donnée manquante. Nous devons juste faire concorder les deux bases de données pour avoir une base de donnée exploitable par l'algorithme



Après plusieurs nettoyages et manipulations, on obtient la base de données ci-dessous:

userId	1	2	3	4	5	6	7	8	9	10	...	1084	1085	1086	1087	1088
title																
10 Things I Hate About You (1999)	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	...	0.000	0.000	0.000	0.000	0.000
101 Dalmatians (1996)	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	...	0.000	0.000	0.000	0.000	0.000
101 Dalmatians (One Hundred and One Dalmatians) (1961)	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	...	0.000	0.000	0.000	0.000	0.000
12 Angry Men (1957)	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	...	0.000	0.000	0.000	0.000	0.000
13th Warrior, The (1999)	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	...	0.000	0.000	0.000	0.000	0.000

Entrainement du modèle KNN

Le code ci-dessous présente comment s'est déroulé l'entraînement de notre modèle:

Collaborative filtering

```
[ ] # Importations
from scipy.sparse import csr_matrix
movie_features_matrix = csr_matrix(movie_features.values)

❶ # Entrainement du modèle
from sklearn.neighbors import NearestNeighbors
model_knn = NearestNeighbors(metric = 'cosine', algorithm = 'brute')
model_knn.fit(movie_features_matrix)

➋ NearestNeighbors(algorithm='brute', metric='cosine')

[ ] movie_features.shape
(5921, 24081)
```

Content based filtering

```
❶ df_genres=df_movies['genres'].str.get_dummies(sep='|')
display(df_genres.head())

print(df_genres.shape)
print(df_genres.columns)

from sklearn.neighbors import NearestNeighbors
nns2 = NearestNeighbors(metric='cosine',algorithm = 'brute')
nns2.fit(df_genres)
```

Réultat des tests du modèle KNN

Le code ci-dessous présente les résultats fournis par les tests après l'entraînement de notre modèle:

Collaborative filtering

```
[ ] # Test avec les identifiants en indice
np.random.seed(5)
query_index = np.random.choice(movie_features.shape[0])
print("recomendation for movie id:",query_index)
distances, indices = model_knn.kneighbors(movie_features.iloc[query_index,:].values.reshape(1, -1), n_neighbors = 6)
print("distances:",distances)
print("indices",indices)

recomendation for movie id: 2915
distances: [[6.66133815e-16 8.30594451e-01 8.47000716e-01 8.51642502e-01
 8.55658649e-01 8.55740497e-01]]
indices [[2915 4912 236 5345 559 1700]]
```

```
▶ # Test avec les noms en indice
for i in range(0, len(distances.flatten())):
    if i == 0:
        print('Recommendations for {}:\n{}'.format(movie_features.index[query_index]))
    else:
        print('{}: {} with distance of {}'.format(i, movie_features.index[indices.flatten()[i]], distances.flatten()[i]))

□ Recommendations for Kaspar Hauser (1993):
1: Song of the Little Road (Pather Panchali) (1955), with distance of 0.8305944505598201:
2: Alphaville (Alphaville, une étrange aventure de Lemmy Caution) (1985), with distance of 0.8470007164437787:
3: Tin Drum, The (Blechtrommel, Die) (1979), with distance of 0.8516425016596493:
4: Beauty of the Day (Belle de jour) (1967), with distance of 0.855658648778105:
5: Enigma of Kaspar Hauser, The (a.k.a. Mystery of Kaspar Hauser, The) (Jeder für sich und Gott Gegen Alle) (1974), with distance of 0.8557404971699502:
```

Content based filtering

```
▶ np.random.seed(33)
query_index = np.random.choice(contents.shape[0])
print("recomendation for movie id:",query_index)
distances, indices = nns.kneighbors(contents.iloc[query_index,:].values.reshape(1, -1), n_neighbors = 6)
print("distances:",distances)
print("indices",indices)

□ recomendation for movie id: 2439
distances: [[0. 0. 0. 0. 0. 0.]]
indices [[25087 2439 18284 4048 785 20969]]
```

```
[ ] for i in range(0, len(distances.flatten())):
    if i == 0:
        print('Recommendations for {}:\n{}'.format(df_movies.at[df_movies.index[query_index],'title']))
    else:
        print('{}: {} with distance of {}'.format(i, df_movies.at[df_movies.index[indices.flatten()[i]],'title'], distances.flatten()[i]))

Recommendations for Towering Inferno, The (1974):
1: Towering Inferno, The (1974), with distance of 0.0:
2: Miss Bala (2011), with distance of 0.0:
3: Left Behind: The Movie (2000), with distance of 0.0:
4: Daylight (1996), with distance of 0.0:
5: Airspeed (1999), with distance of 0.0:
```

Comme nous pouvons le remarquer, le content based filtering présente des limites (proposer le même genre que le film intéressé pourrait ne pas susciter de l'intérêt chez l'utilisateur).

De ce fait, naît le collaborative filtering, qui, en plus de se baser sur les différents genres, se base aussi sur l'avis des autres utilisateurs ayant suivi le même film et leur préférence, pour fournir une recommandation adéquate.

KMeans

Introduction

Nous allons construire un système basé sur le clustering pour classer les utilisateurs en groupes de même intérêt en utilisant l'algorithme k-means. Nous utiliserons des données, où les utilisateurs ont noté des films avec une note de 4+ sur la supposition de cela, si un utilisateur note un film 4+, il / elle peut l'aimer.

Etapes d'algorithme

- 1.Tout d'abord, nous devons sélectionner le nombre de clusters que nous voulons pour notre ensemble de données. Plus tard, une **méthode de coude(Elbow method)** sera expliquée pour la sélection du nombre optimal de clusters.
- 2.Ensuite, nous devons sélectionner k points aléatoires appelés centroïdes qui ne sont pas nécessaires dans notre jeu de données. Parce que pour éviter le piège d'initialisation aléatoire qui peut coller aux mauvais clusters, nous utiliserons k-means ++ pour initialiser k centroïdes et il est fourni par Scikit-Learn dans l'algorithme K-Means.
- 3.L'algorithme K-Means attribuera chaque point de données à son centroïde le plus proche, ce qui nous donnera finalement k clusters.
- 4.Le centroïde sera recentré sur une position qui est maintenant en fait le centroïde de son propre cluster et sera le nouveau centroïde.
- 5.Il réinitialisera tous les clusters et attribuera à nouveau chaque point de l'ensemble de données à son nouveau centroïde le plus proche.
- 6.Si les nouveaux clusters sont identiques au cluster précédent OU si le nombre total d'itérations est terminé, il s'arrêtera et nous donnera les clusters finaux de notre ensemble de données. Sinon, il reviendra à l'étape 4.

Méthode de coude : Elbow Method

La méthode du coude est le meilleur moyen de trouver le nombre optimal de clusters. Pour cela, nous devons trouver dans les clusters la somme des carrés, **within-clusters sum of squares(WCSS)**. WCSS est la somme des carrés de chaque point distant de son centroïde et sa formule mathématique est la suivante :

$$WCSS = \sum_{i=1}^K \sum_{j=1}^{N_i} \text{distance}(P_{i,j}, C_i)^2$$

K : le nombre total de clusters .

N_i : la taille du i-ème cluster ou nous pouvons également dire que les points de données dans le i-ème cluster.

C_i : le centroïde du i-ème cluster.

P_{i,j} : la j-ème donnée point du i-ème cluster.

WCSS nous indiquera à quelle distance se trouve le centroïde de ses points de données. Au fur et à mesure que nous augmentons le nombre de clusters, le **WCSS** deviendra petit et après une certaine valeur de K, le **WCSS** diminuera lentement et nous nous arrêterons là et choisirons le nombre optimal de clusters.

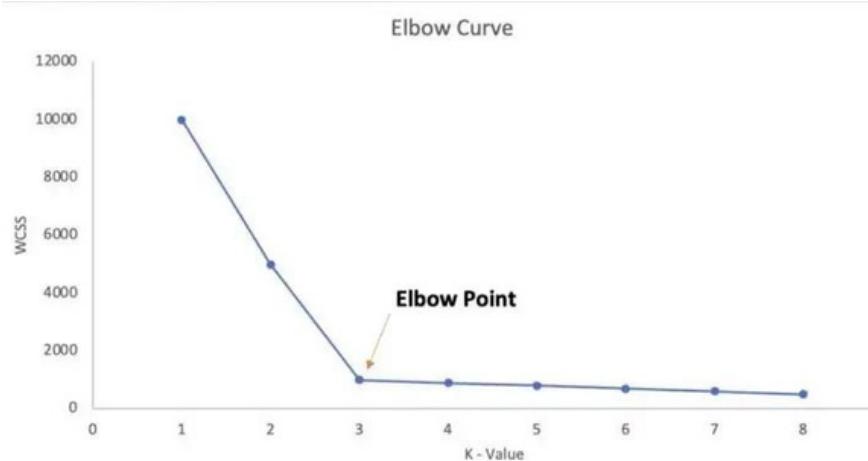


Figure 1 — Elbow Method Plot

Ingénierie des données

Cette section est divisée en deux sous-sections. Tout d'abord, nous allons importer des données et les réduire dans un sous-DataFrame, afin que nous puissions nous concentrer davantage sur notre modèle et voir quel type d'utilisateurs a évalué les films et quel type de recommandation pour lui en fonction de cela. Deuxièmement, nous effectuerons l'ingénierie des caractéristiques (feature engineering) afin d'avoir des données sous une forme valide pour l'algorithme d'apprentissage automatique.

- **Préparation des données pour le modèle**

Information sur la dataset :

```
Shape of ratings dataset is : (20000263, 4)
```

```
Max values in dataset are :  
userId           138493  
movieId          131262  
rating           5.0  
timestamp    2015-03-31 06:40:02  
dtype: object
```

```
Min values in dataset are :  
userId           1  
movieId          1  
rating           0.5  
timestamp    1995-01-09 11:46:44  
dtype: object
```

Nous avons téléchargé MovieLens Dataset depuis Kaggle.com. Ici, nous allons d'abord importer un ensemble de données d'évaluation (**'rating.csv'**), car nous voulons que les utilisateurs évaluent les films et nous filtrerons ensuite les données où les utilisateurs ont donné 4+ évaluations.

Nous filtrerons ensuite les données où les utilisateurs ont donné 4+ évaluations.

```
Shape of ratings dataset is: (9995410, 4)
```

```
Max values in dataset are :
```

```
    userId           138493  
    movieId         131262  
    rating          5.0  
    timestamp      2015-03-31 06:11:28  
    dtype: object
```

```
Min values in dataset are :
```

```
    userId           1  
    movieId         1  
    rating          4.0  
    timestamp      1995-01-09 11:46:44  
    dtype: object
```

Dataset après filtre :

Ainsi, la note minimale donnée par les utilisateurs est désormais de 4,0 et l'ensemble de données est également diminué par 10M, mais l'ensemble de données est encore volumineux et nous voulons le réduire davantage.

Nous utilisons une sous base de données de 100000 observations où 7230 films ont été notés par 1375 utilisateurs.

```
Shape of ratings dataset is : (100000, 4)
```

```
Max values in dataset are :
```

```
    userId           1378  
    movieId         130219  
    rating          5.0  
    timestamp      2015-03-30 21:55:54  
    dtype: object
```

Dataset réduite :

```
Min values in dataset are :
```

```
    userId           1  
    movieId         1  
    rating          4.0  
    timestamp      1996-03-22 13:17:21  
    dtype: object
```

```
Total Users: 1375
```

```
Total Movies which are rated by 1357 users: 7230
```

- Data featuring

L'ingénierie de caractéristiques ou features engineering correspond aux étapes de prétraitement qui transforment les données brutes en caractéristiques pouvant être utilisées dans les algorithmes d'apprentissage automatique.

Dans notre cas, nous transformons les données brutes en une matrice creuse qui représente la relation entre les utilisateurs et les films.

Une matrice creuse est un cas particulier de matrice dans laquelle le nombre d'éléments nuls est beaucoup plus élevé que le nombre d'éléments non nuls. Cette matrice creuse constitue les données d'entraînement de notre algorithme KMeans.

Pour construire cette matrice, nous utiliserons la fonction **countVectorizer** fournie par la bibliothèque **sklearn** et qui permet, dans ce cas, de créer une matrice dans laquelle chaque film unique est représenté par une colonne de la matrice, et chaque utilisateur est une ligne dans la matrice. La valeur de chaque cellule vaut 1 si l'utilisateur a regardé le film, 0 sinon.

Cela peut être visualisé comme suit :

Construction du modèle

- **K optimal**

Pour regrouper les données, nous devons tout d'abord trouver le nombre optimal de clusters. À cette fin, nous définirons un objet pour la méthode du coude qui contiendra deux fonctions, d'abord pour exécuter l'algorithme K-Means pour un nombre différent de clusters et l'autre pour afficher le tracé.(code joigné)

Parfois, nous ne pouvons pas déterminer la valeur optimale de K d'après le graphe de WCSS. Nous pouvons donc définir une limite pour des observations plus claires de l'évolution de la valeur WCSS. Nous traçons la différence entre chacune des deux valeurs WCSS en deux itérations consécutives. Autrement dit, lorsque les modifications de la valeur WCSS resteront à l'intérieur de la limite requise, nous dirons que nous avons trouvé le coude, après quoi les modifications sont faibles.

WCSS est calculer en utilisant la méthode **inertia_**.

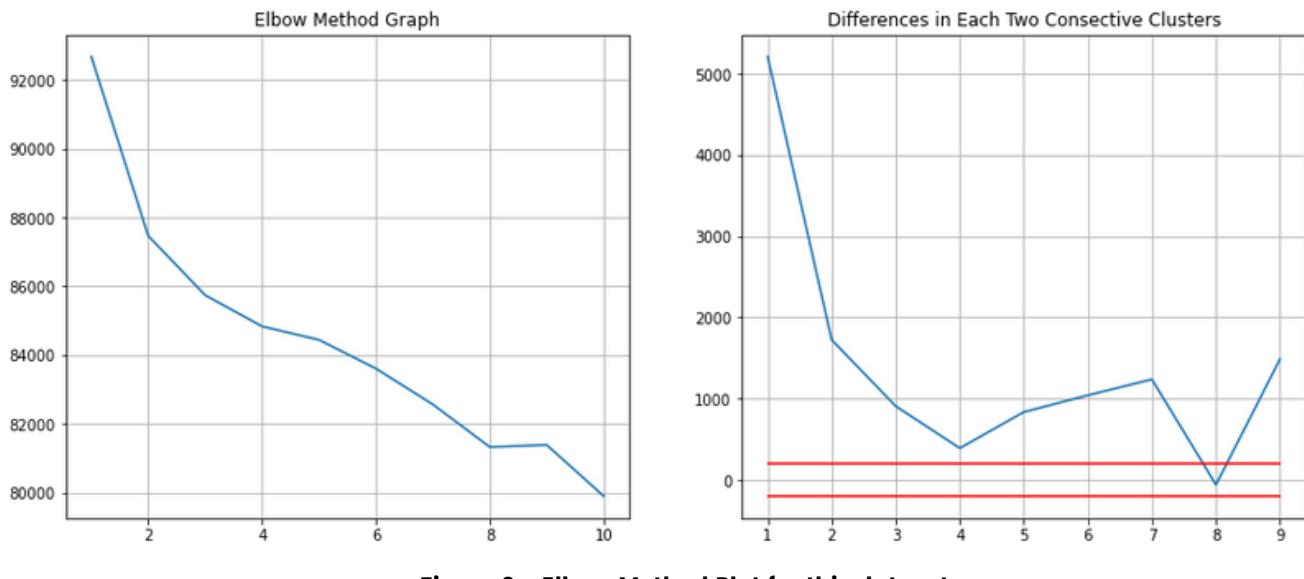


Figure 2— Elbow Method Plot for this dataset

Nous constatons qu'il n'y a pas de coude clair et nous n'avons pas non plus de différences à l'intérieur de la frontière.
Nous essayerons avec la méthode de Silhouette.

Construction du modèle

• La méthode de Silhouette

Le coefficient de silhouette ou le score de silhouette K-Means est une mesure de la similarité d'un point de données au sein d'un cluster (cohésion) par rapport à d'autres clusters (séparation).

L'équation pour calculer le coefficient de Silhouette pour un point de données particulier :

$$S(i) = \frac{b(i) - a(i)}{\max \{a(i), b(i)\}}$$

- **S(i)** est le coefficient de silhouette du point de données i.
- **a(i)** est la distance moyenne entre i et tous les autres points de données du cluster auquel i appartient.
- **b(i)** est la distance moyenne de i à tous les clusters auxquels i n'appartient pas.



Nous calculerons ensuite la silhouette_moyenne pour chaque k.

$$\text{AverageSilhouette} = \text{mean}\{S(i)\}$$

- La valeur du coefficient de silhouette est comprise entre [-1, 1].
- Un score de 1 dénote le meilleur sens que le point de données *i* est très compact au sein du cluster auquel il appartient et éloigné des autres clusters.
- La pire valeur est -1.
- Les valeurs proches de 0 indiquent des clusters qui se chevauchent.

- Résultat de la méthode de Silhouette

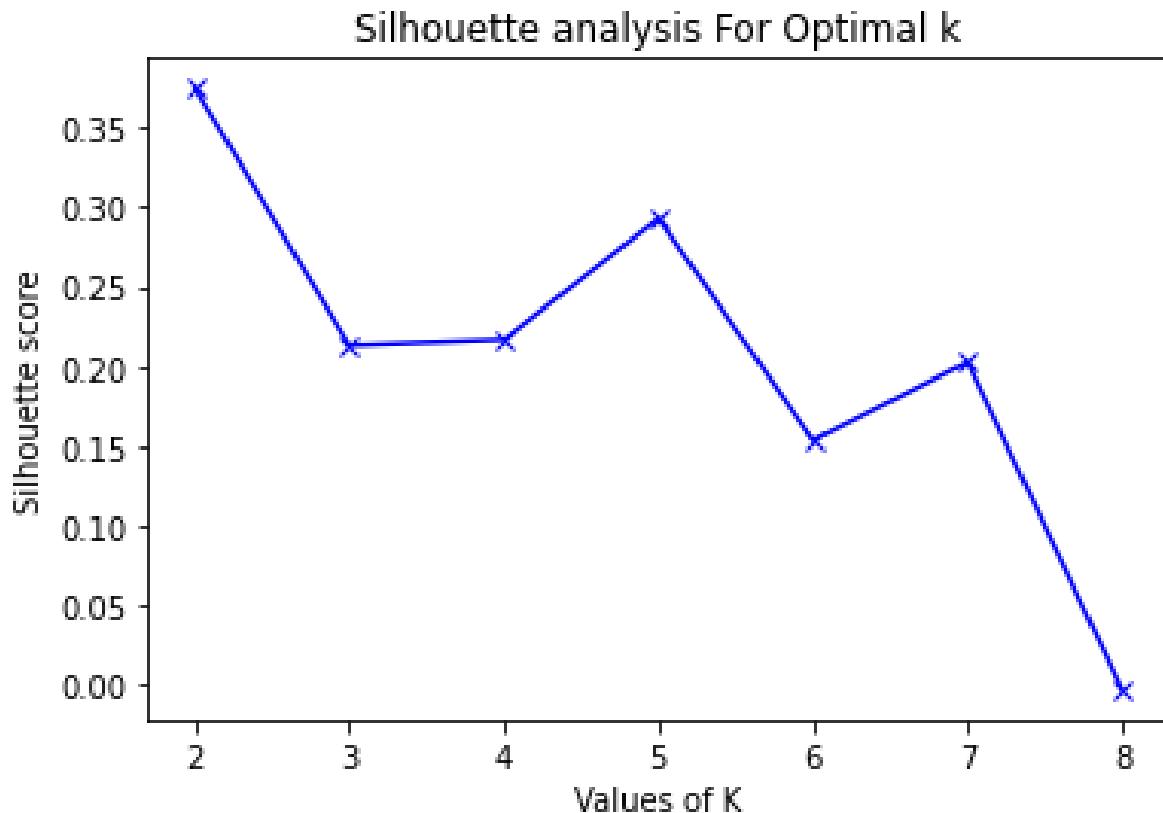


figure 3 : Silhouette method result

Avec la méthode de Silhouette, nous pourrons dire que la valeur optimal de K est k=2 pour cette dataset.

Entrainement du modèle

En utilisant 2 cluster, nous obtenons les résultats suivants :

- **Les utilisateurs et leur cluster correspondant**

	0	1	2	3	4	5	6	7	8	9	...	1365	1366	1367	1368	1369	1370	1371	1372	1373	1374
userId	1	2	3	4	5	6	7	8	9	10	...	1369	1370	1371	1372	1373	1374	1375	1376	1377	1378
Cluster	1	1	0	1	1	1	1	1	1	1	...	1	1	1	0	1	1	1	0	1	0

2 rows × 1375 columns

- **Les films populaires dans le cluster N°1**

	0	1	2	3	4	5	6	7	8	9	...	6290	6291	6292	6293	6294	6295	6296	6297	6298	6299
movielid	1198	593	296	1196	260	318	2571	50	356	1210	...	6279	6288	6289	6298	6300	6308	6322	6327	6330	130219
Count	150	146	145	144	143	141	141	136	132	132	...	1	1	1	1	1	1	1	1	1	1

2 rows × 6300 columns

- **Les films populaires dans le cluster N°2**

	0	1	2	3	4	5	6	7	8	9	...	4936	4937	4938	4939	4940	4941	4942	4943	4944	4945
movielid	318	296	356	593	527	260	50	110	2571	457	...	3181	3188	3219	8336	8334	8332	8327	8255	3222	128832
Count	417	379	370	347	300	283	275	266	262	244	...	1	1	1	1	1	1	1	1	1	1

2 rows × 4946 columns

Performance du modèle

- La méthode de Silhouette

```
# compute an average silhouette score for each point
from sklearn.metrics import silhouette_score, silhouette_samples

silhouette_score_average = silhouette_score(sparseMatrix, kmeans.predict(sparseMatrix))
silhouette_score_average

0.3751412335511355
```

```
# Let's see the points
silhouette_score_individual = silhouette_samples(sparseMatrix, kmeans.predict(sparseMatrix))

# iterate through to find any negative values
s=0
for each_value in silhouette_score_individual:
    if each_value < 0:
        #print(f'We have found a negative silhouette score: {each_value}')
        s = s+1
s
```

198

Avec un $k=2$, nous constatons que parmi 1375 utilisateurs, 198 utilisateurs ne sont pas bien classés. Avoir classé 1177 points constitue 85,6% des points qui appartiennent bien à leur classe et 14,4% des points qui n'appartiennent pas à la classe qui leur correspond.

Faire des prédictions

Maintenant, nous allons recommander la plupart des films préférés des utilisateurs du cluster que l'utilisateur n'a pas ajouté au favori plus tôt. Et aussi lorsqu'un utilisateur a ajouté un autre film dans sa liste de favoris, nous devons également mettre à jour les ensembles de données de films de clusters.

- **Etapes du processus de recommandation:**

1. Récupérer les films que l'utilisateur a déjà regardé.
2. Trouver le cluster auquel l'utilisateur appartient.
3. Récupérer les films les plus notés par les utilisateurs du cluster.
4. Recommander à l'utilisateur les films les plus notés par les utilisateurs du cluster et qu'il ne les a pas encore regardé.

Les 10 premiers films recommandés pour l'utilisateur N°2 qui appartient au cluster N°1.

```
user2RecommendedMovies = movies_metadata[movies_metadata['movieId'].isin(cluster_movies_list)]['title'][:10]
user2RecommendedMovies
```

0 Toy Story (1995)
1 Jumanji (1995)
3 Waiting to Exhale (1995)
4 Father of the Bride Part II (1995)
5 Heat (1995)
6 Sabrina (1995)
7 Tom and Huck (1995)
8 Sudden Death (1995)
9 GoldenEye (1995)
10 American President, The (1995)
Name: title, dtype: object

Les 10 premiers films recommandés pour l'utilisateur N°12 qui appartient au cluster N°2.

```
user12RecommendedMovies = movies_metadata[movies_metadata['movieId'].isin(cluster_movies_list)]['title'][:10]
user12RecommendedMovies
```

1 Jumanji (1995)
2 Grumpier Old Men (1995)
3 Waiting to Exhale (1995)
4 Father of the Bride Part II (1995)
5 Heat (1995)
6 Sabrina (1995)
7 Tom and Huck (1995)
8 Sudden Death (1995)
9 GoldenEye (1995)
10 American President, The (1995)
Name: title, dtype: object

Collaborative Filtering basé sur Neural Networks

Introduction

Nous allons construire un système basé sur un MLP (Multi Layer Perceptron). Pour estimer une évaluation sur un film en se basant sur les films déjà regardés. Nous allons utiliser ces évaluations afin de suggérer des nouveaux films pour l'utilisateur. Pour ce faire, on va entraîner notre réseau à partir de la dataset **MovieLens**.

Preprocessing

Puisque la dataset est très grande, et suite à des contraintes de ressources. On va se contenter d'une seule partie de **200K observations**.

On effectue une normalization sur les 'ratings'

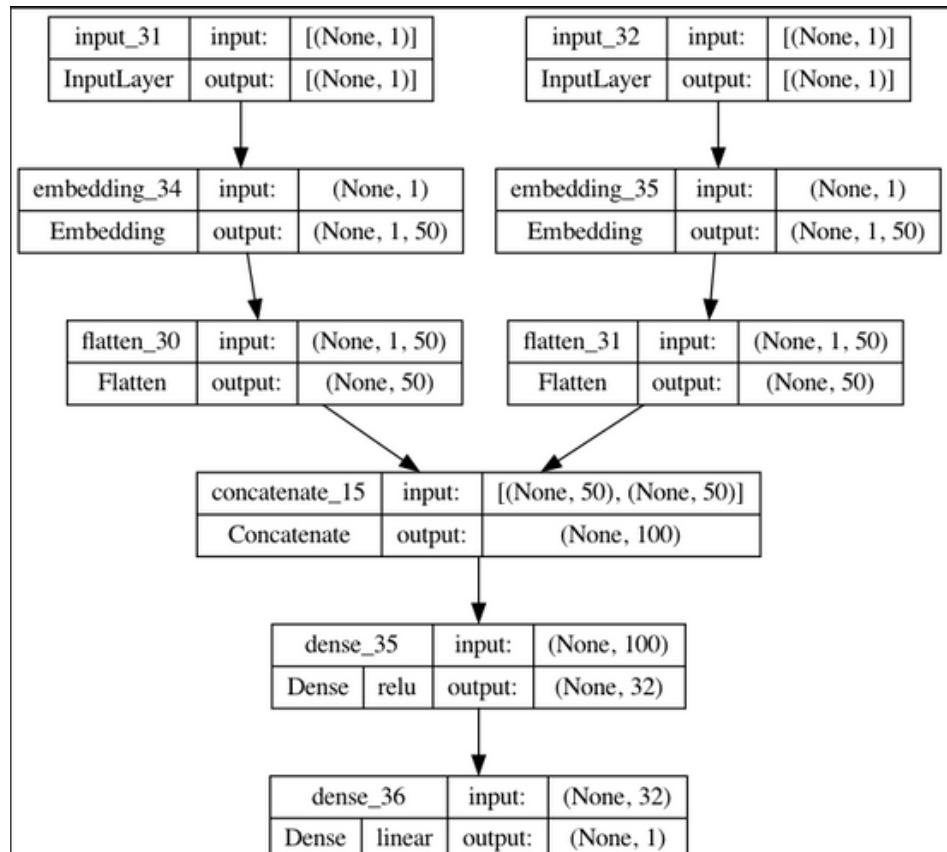
```
df = df.iloc[:200000, :]  
X = df[['userId', 'movieId']]  
y = df['rating']  
min_rate = y.min()  
max_rate = y.max()  
y = y.apply(lambda x: ((x-min_rate)/(max_rate - min_rate)))
```

Le modèle

Notre modèle se compose de deux parties:

1. La première partie avec 2 couches d'entrée, chaque une est suivie par une couche de vectorisation (**Embedding layer**). Ces derniers seront ensuite concaténées via une couche de concatenation.
2. La deuxième partie est un réseau MLP de deux couches avec **ReLU** comme activation, son rôle est de **prédir** le 'ratings' des films non visionnées.

Sa structure est résumée dans le graph suivant.



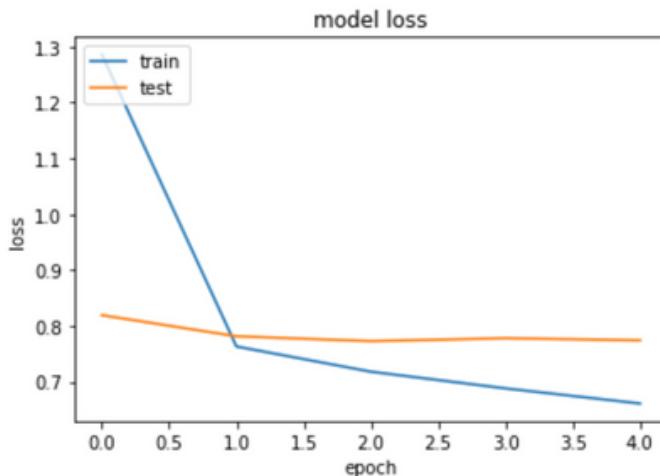
On a opté pour **Adam** comme notre optimisateur grâce à son performance et sa rapidité.
Pour la fonction loss, on a choisi **Mean Square Error (MSE)**

```
model.compile(loss='mean_squared_error', optimizer='adam')

history = model.fit([X_train.iloc[:,0], X_train.iloc[:,1]], y_train,
epochs=5, verbose=1, validation_data=([X_test.iloc[:,0],
X_test.iloc[:,1]], y_test), batch_size=64)
```

Après la phase d'entraînement, on a obtenu les résultats suivants.

```
Epoch 1/5
2500/2500 [=====] - 13s 5ms/step - loss: 1.2859 - val_loss: 0.8193
Epoch 2/5
2500/2500 [=====] - 13s 5ms/step - loss: 0.7633 - val_loss: 0.7815
Epoch 3/5
2500/2500 [=====] - 13s 5ms/step - loss: 0.7182 - val_loss: 0.7728
Epoch 4/5
2500/2500 [=====] - 13s 5ms/step - loss: 0.6883 - val_loss: 0.7781
Epoch 5/5
2500/2500 [=====] - 15s 6ms/step - loss: 0.6613 - val_loss: 0.7743
```



La prédiction

Dans cette phase, on va travailler avec une autre dataset **movies.csv** qui contient les détails sur chaque film.

Premièrement, on doit séparer les films regardés des films non encore vus. Pour ça on a créé deux listes qui contiennent le movield de chaque film.

```
movies_df = pd.read_csv('movies.csv')
movies_df.head(2)
```

	movield	title	genres
0	1	Toy Story (1995)	Adventure Animation Children Comedy Fantasy
1	2	Jumanji (1995)	Adventure Children Fantasy

```
user_id = df.userId.sample(1).iloc[0]
watched_mobs = df[df.userId == user_id].iloc[:,1]
not_watched_mobs =
movies_df[~movies_df.movieId.isin(watched_mobs.values)].movieId
```

Ensuite, on associe à chaque utilisateur les films qu'il n'a pas encore regardé et on les stocke dans un tableau à deux dimensions. C'est possible grâce à la fonction **hstack()** qui combine deux listes en un tableau 2D, comme illustré ci-dessous.

```
user_movie_array = np.hstack(  
    ([[user_encoder]] * len(not_watched_mobs), not_watched_mobs))  
  
[41]: array([[ 248,  5804],  
             [ 248,      1],  
             [ 248,   482],  
             ...,  
             [ 248, 3870],  
             [ 248, 2989],  
             [ 248, 7869]])
```

On passe ce dernier à notre modèle qui va évaluer chaque un des films. Après, on prend les top 10 premiers en termes de note d'évaluation.

```
def pred_ratings(user_movie_array):  
    ratings = model.predict([user_movie_array[:,0],user_movie_array[:,1]]).flatten()  
    return ratings  
  
print(pred_ratings(user_movie_array))  
  
def get_top_ratings(ratings, k=10):  
    top_ratings = ratings.argsort()[-k:]  
    top_ratings = [movie_encoded2movie.get([x][0]) for x in top_ratings]  
    return top_ratings  
  
print(get_top_ratings(ratings))
```

272/272 [=====] - 1s 2ms/step
[2.948369 3.9566817 3.285802 ... 3.7655272 3.14194 3.1224349]
[3737, 72694, 25757, 3675, 141844, 67295, 1961, 27826, 8117, 76]

	movielid	title	genres
1169	1193	One Flew Over the Cuckoo's Nest (1975)	Drama
2911	2997	Being John Malkovich (1999)	Comedy Drama Fantasy
4752	4848	Mulholland Drive (2001)	Crime Drama Film-Noir Mystery Thriller
4778	4874	K-PAX (2001)	Drama Fantasy Mystery Sci-Fi
4782	4878	Donnie Darko (2001)	Drama Mystery Sci-Fi Thriller

On affiche les top 10 films recommandés, ainsi que les films que l'utilisateur à aimé auparavant:

```
print("Showing recommendations for user: {}".format(user_id))
print("====" * 9)
print("Movies with high ratings from user")
print("----" * 8)
top_watched_movies = df[df.userId == user_id].sort_values(by='rating', ascending=False).movieId.head(5)
top_watched_movies = movies_df[movies_df.movieId.isin(top_watched_movies.values)]

for element in top_watched_movies.itertuples():
    print(element.title, element.genres)

print("====" * 9)
print("Top 10 Recommendations:")
print("----" * 8)

top_movies_ids = get_top_ratings(ratings, 10)
top_movies = movies_df[movies_df.movieId.isin(top_movies_ids)]

for element in top_movies.itertuples():
    print(element.title, element.genres)
```

Showing recommendations for user: 249
=====

Movies with high ratings from user

Star Wars: Episode V – The Empire Strikes Back (1980) Action|Adventure|Sci-Fi
Raiders of the Lost Ark (Indiana Jones and the Raiders of the Lost Ark) (1981) Action|Adventure
Good, the Bad and the Ugly, The (Buono, il brutto, il cattivo, Il) (1966) Action|Adventure|Western
Lord of the Rings: The Fellowship of the Ring, The (2001) Adventure|Fantasy
Deadpool 2 (2018) Action|Comedy|Sci-Fi

=====

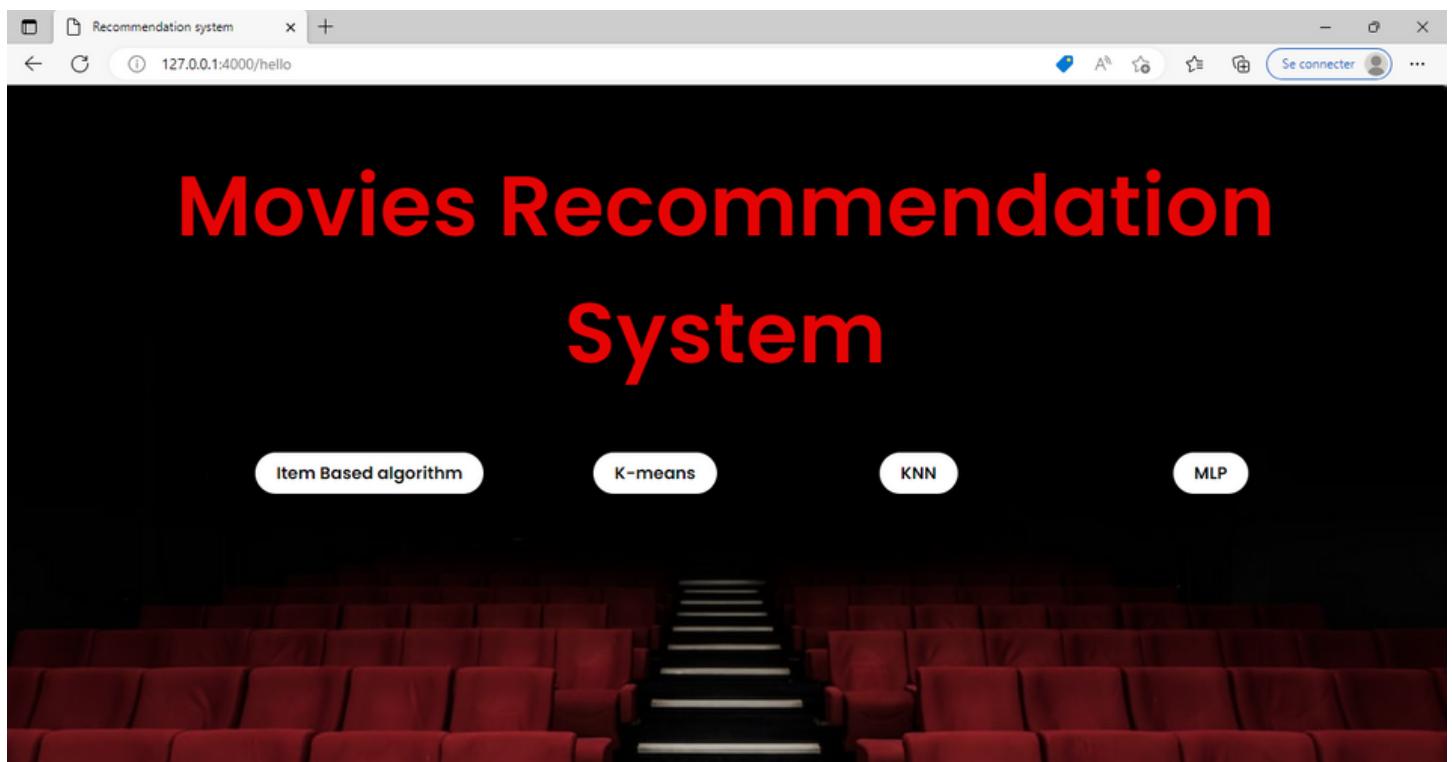
Top 10 Recommendations:

Screamers (1995) Action|Sci-Fi|Thriller
Rain Man (1988) Drama
White Christmas (1954) Comedy|Musical|Romance
Lonely Are the Brave (1962) Drama|Western
In China They Eat Dogs (I Kina spiser de hunde) (1999) Action|Comedy
Jazz Singer, The (1927) Drama|Musical|Romance
Touch of Pink (2004) Comedy|Drama|Romance
Kung Fu Panda: Secrets of the Furious Five (2008) Action|Animation|Children|Comedy
Shrink (2009) Drama
12 Chairs (1971) Adventure|Comedy

Démonstration :

Application Web pour

la recommandation de films



A screenshot of a web browser window showing the results of an item-based recommendation. The title 'Item based Recommendation System' is at the top. A search bar contains the text 'Bad Boys (1995)'. A 'Recommend!' button is below it. A link 'Top 5 Movies to watch' is shown. Five movie posters are displayed in a row: '71 (2014)', 'Hellboy: The Seeds of Creation (2004)', 'Round Midnight (1986)', 'Salem's Lot (2004)', and "'Til There Was You (1997)''. Each poster includes its title and year.

Kmeans movie Recommendation System

Enter your user ID :

Recommend!

Top 5 Movies to watch

Eragon

CASINO ROYALE

007 FROM RUSSIA WITH LOVE

JURASSIC PARK

Billy Madison

Knn movie Recommendation System

Enter the id of the user :

Recommend!

Top 5 Movies to watch

Addams Family Values

BATMAN FOREVER

ACE VENTURA PET DETECTIVE

Mrs. Doubtfire

BATMAN RETURNS

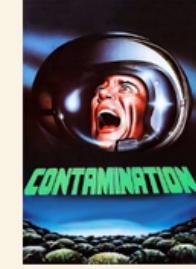
MLP Prediction — ⌂ ×
127.0.0.1:4000/result4 Se connecter

MLP Movie Recommendation System

Enter the id of the user:

Recommend!

★ Top Watched movies



Bibliographie

- 1- : F. Maxwell Harper and Joseph A. Konstan. 2015. The MovieLens Datasets: History and Context. ACM Transactions on Interactive Intelligent Systems (TiiS) 5, 4, Article 19 (December 2015), 19 pages
- 2- <https://www.kaggle.com/code/kanncaa1/recommendation-systems-tutorial>
- 3- <https://github.com/rposhala/Recommender-System-on-MovieLens-dataset>