Projet 3
Développer un moteur de recommandation de films

Le 16/03/2018

Par Sofiane NAAR

Objectif

L'objectif est de créer un moteur de recommandation de films.

Le data set de travail est une base d'environ 5000 films issue d'IMDB

Le moteur devra être capable de retourner 5 recommandations de films susceptibles de plaire à l'utilisateur

Cela à partir d'une requête d'un nom ou un id de film

Approche suivie

- Pas d'historique des choix d'utilisateurs

Approche suivie

- Pas d'historique des choix d'utilisateurs
  - Une approche filtrage collaboratif non envisageable

Approche suivie

- Pas d'historique des choix d'utilisateurs
  - Une approche filtrage collaboratif non envisageable
- Seulement des informations concernant les films

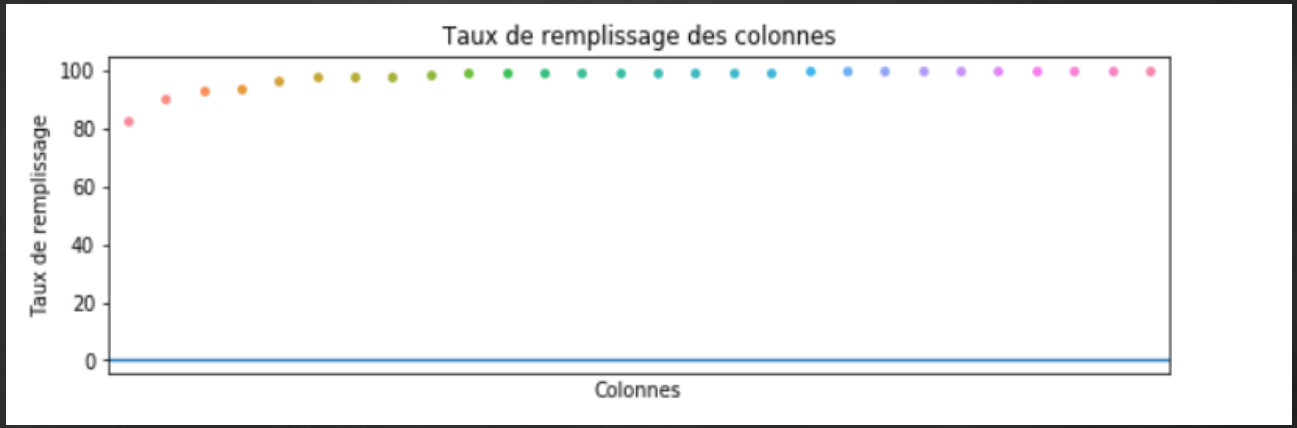
Approche suivie

- Pas d'historique des choix d'utilisateurs
  - Une approche filtrage collaboratif non envisageable
- Seulement des informations concernant les films
  - une approche basée sur le contenu s'y prête bien

- Objectif
- Approche suivie
- Etat du datset
- Codification de la donnée
- Construction de la distance
- Gérer le poids
- Cas Acteurs et mots clés
- Modélisation effective de films
- La definition de la distance
- Propriétés de la distance
- Matrice des distances
- Classification
- DBSCAN
- DBSCAN analyse
- Clustering hiérarchique
- Solution retenue
- Popularité
- La fonction *nearset* redéfinie
- Mise en ligne

Etat du dataset

Une « photo » de l'état du data set



L'état du data set est « **propre** »
Variables utilisée renseignées à au moins 96.95%.

Les valeurs manquantes gérées -au besoin - à la construction des notions de
« **distance** » et « **popularité** »

Codification de la donnée

```python
# fonction générant à chaque appel une valeur numérique différente
symbole = 1
def gen_symbole():
    global symbole
    symbole = symbole + 1
    return symbole
```

8491 acteurs et réalisateurs

26 genres différents

8087 keywords différents

65 pays différents

24 périodes de 5 ans (depuis 1900)

Toute valeur none est remplacée par une valeur numérique unique

Construction de la distance

Recommander 5 films similaires

➡ Que veut dire deux films similaires ?

Et si deux films sont « similaires de la même façon » au film de l'utilisateur

➡ Quel film doit prioriser le moteur ?

Construction de la distance

Recommander 5 films similaires

➡ Que veut dire deux films similaires ?

➡ Quantifier la similarité

Et si deux films sont « similaires de la même façon » au film de l'utilisateur

➡ Quel film doit prioriser le moteur ?

➡ Modéliser la « popularité »

Construction de la distance

Si **d** est une distance mathématique normée définie sur l'ensemble des films

➡ La similarité entre deux films, f1, f2 peut se mesurer par la quantité ***1 - d (f1, f2)***

La distance augmente quand la similarité diminue et inversement.

Modéliser la notion de popularité

➡ Le moteur peut ainsi – *à distance égale* par rapport à un film F - recommander le film le plus populaire .

Construction de la distance

Les gens préfèrent-t-ils des films selon

- Le genre ?
- Les acteurs ?
- La période ?
- Une ressemblance (présumée) dans l'histoire via les mots clés ?
- etc. etc.

➡ Un choix forcement **subjectif** !

Construction de la distance

➡ Un choix forcement **subjectif** !

Pour définir la distance, je ne considère que les champs relatifs aux :

- Acteurs : acteur 1, acteur 2 , acteur 3
- Réalisateur
- Genres : 26 genres possibles
- Mots clés : 5 mots clés possibles.
- Période de production (plages de 5 années)
- Pays

Toutes **qualitatives** ➡ ~~**D. Euclidienne**~~

Construction de la distance

Entre deux objets, représenté chacun par une séquence finie de N symboles

La distance de Hamming sert à quantifier la différence en se basant sur le nombre de symboles **non-concordants.**

La distance entre deux objets ayant ***n*** points en commun sera donc égale à ***1 – (n / N)***

(➡ Hamming_loss dans le module **sklearn.metrics**)

Hamming(
('x', 'y', 'z', 'e'),
('p', 'y', 'q','d')
)

➡ **1- 1/4**

Construction de la distance

Un autre exemple « allégé »

acteur 1	acteur 2	acteur 3	réalisateur	horreur	action	comedie	amour	mot clé 1	mot clé 2	mot clé 3	période	pays
Tom	Peter	Emma	Mark		1		1	amour	crime		5	France
Chris	Tom	Tony	Mark		1			crime	espion	prison	10	France

Convient bien aux variables réalisateur, genres, période et pays

Ne convient pas pour mesurer la distance apportée via mots clés

Ne « capte » pas « toute » la similarité apportée par les variables acteur 1, acteur 2 et acteur 3 dans la mesure où elle ne tient pas compte du fait que « Tom » joue le premier rôle dans le film 1 et le deuxième rôle dans le film 2

Toutes les variables sont considérées avec la même importance, or je souhaite qu'une coïncidence dans la variable acteur 1 apporte 3 fois la similarité par laquelle contribue la coïncidence en un genre.

Gérer le poids

Avec Hamming il suffit de dupliquer n fois le même champ pour lui attribuer **n** fois l'importance accordées aux autres.

Hamming (
('x', 'y', 'z', 'e' , 'y'),
('p', 'y', 'q', 'd', 'y')
) ➡ **1 - 2/5**

Hamming (
('x', 'y','z', 'e', 'y'),
('x', 't','q', 'd', 't')
) ➡ **1 - 1/5**

Cas acteurs et mots clés

Hamming(
('un', 'amour', 'ciel', 'le'),
('ciel', 'vélo', 'un' 'li')
) ➡ **1 - 0/4**

Pour capter la similarité , calculer la distance de hamming entre deux nouveaux vecteurs jetables générés à la volée

Hamming(
(1, 1, 1, 1),
(1, 1, 0, 0)
) ➡ **1 - 2/4**

Modélisation effective de films

Un film F vecteur de taille 40 :
concaténation **des sous vecteurs** suivants : F = (A , K , T)

- **A** : (*Acteurs*) de taille **3** contenant dans l'ordre l'acteur 1, acteur 2, acteur 3 codifiés numériquement
- **K** : (*Keyswords*) de taille **5** correspondants aux **5** mots clé codifiés
- **T** : (*auTres*) sous vecteur de taille 32 contenant :
  - **26** codes numériques correspondant au 26 genres **possibles** que peut avoir F dans un ordre bien précis et établie au préalable, si le genre est présent pour F, nous mettons la valeur **1** sinon, un code numérique unique
  - **1** code numérique dupliqué correspondant à l'acteur 1
  - **1** code numérique dupliqué correspondant au réalisateur
  - **1** code numérique correspondant au pays
  - **1** code numérique correspondant à la période

La définition de la distance

Soient F1, F2 deux films modélisés comme décrit précédemment, alors:

$d(F1, F2) = d((A1, K1, T1), (A2, K2, T2))$

$= Hamming((A1', K1', T1), (A2', K2', T2))$

Pour tout U, V , on associe U', V' , de même taille que U et V :

- U' un premier vecteur dont toutes les composantes sont = à **1**
- V' dont toutes les composantes sont égale à **0** , excepté **i** composantes qui valent **1**, **i** étant le cardinal de U intersection V.

Propriétés de la distance

- **d** est normée comprise entre 0 et 1
  d prend ces valeurs dans l'ensemble discret {0, 1/40, 2/40 , 3/40 ….39/40, 1}
- **d** répond à mes hypothèses initiales :
  - Elle capte la similarité dû à la présence d'un acteur commun peu import sa place
  - Elle capte la similarité dû au fait d'avoir l'acteur principale ou le réalisateur de commun
  - Elle capte la similarité donnée par les mots clés
  - Elle capte la similarité dû aux genres, pays et période
  - Elle peut attribuer facilement des poids différents

Matrice des distances

```python
from sklearn.metrics import hamming_loss
from sklearn.neighbors import DistanceMetric


def handle_intersection(u,v): # u et v doivent avoir la meme tailles
    intersection = set(u).intersection(set(v))
    n = len(u)
    return [
        [1 for i in range(n)],
        [1 if u[i] in intersection else 0 for i in range(n)]
    ]


def ham_distance(u,v):
    keywords = handle_intersection(u[0:5:], v[0:5:])
    actors = handle_intersection(u[5:8:], v[5:8:])
    u = list(u)
    v = list(v)
    return hamming_loss(
        keywords[0] + actors[0] + u[7::],
        keywords[1] + actors[1] + v[7::]
    )
```

```python
h_dist = DistanceMetric.get_metric('pyfunc', func=ham_distance)
X = h_dist.pairwise(codified_movies.as_matrix())
```

TEST : définition de la fonction nearest(film, n)

Classification

➡ Disposant de la matrice des distance X

➡ Regrouper les films les *plus semblables* à l'aide d'un algorithme de classification non supervisée

➡ La recommandation des 5 films similaires au film de l'utilisateur, se fera au sein du cluster dans lequel il se trouve.

K-means implique la notion de « centroïde »

incompatible avec ma distance !

➡ DBSCAN !

DBSCAN

➡ Agnostique à la distance utilisée

➡ N'exige pas de prédéfinir le nombre de clusters souhaité.

Il faut tout de même éstimer :

- epsilon : un nombre réel positif
- *MinPts* : le nombre minimum de points devant se trouver dans un rayon de taille epsilon pour que ces points soient considérés comme un cluster

DBSCAN

Une heuristique permettant de déterminer conjointement epsilon et MinPts :

epsilon :

- Calculer pour chaque point de l'espace la distance à son plus proche voisin.
- Prendre *epsilon* tel qu'une part "suffisamment grande" des points aient une distance à son plus proche voisin inférieure à epsilon.

MinPts :

- Calculer pour chaque point le nombre de ses voisins dans un rayon de taille *epsilon*
- Prendre MinPts tel qu'une part "suffisamment grande" des points aient plus de MinPts points dans leur *epsilon -voisinage*.

Par "suffisamment grand" on entend, par exemple, 95% ou 90% des points (WIKIPEDIA)

DBSCAN

Epsilon : 0.87 ET MinPts : 3

```
from sklearn.cluster import DBSCAN
db = DBSCAN(epsilon, minPts, metric="precomputed")
db.fit(X)
```

1686 bruits

34 clusters de différentes tailles

[(-1, 1686), (0, 3192), (1, 3), (2, 4), (3, 5), (4, 5), (5, 3), (6, 3), (7, 6), (8, 3), (9, 3), (10, 3), (11, 3), (12, 3), (13, 3), (14, 4), (15, 5), (16, 4), (17, 3), (18, 3), (19, 3), (20, 4), (21, 3), (22, 5), (23, 6), (24, 3), (25, 4), (26, 3), (27, 3), (28, 4), (29, 3), (30, 3), (31, 3), (32, 4), (33, 3)]

DBSCAN analyse

Les films des differents clusters autre que 0 sont bien proches les uns des autres (testé avec nearest)

Le cluster 0 contient 3192 éléments

certains totalement différents

pourquoi?

DBSCAN analyse

X de taille (4998 x 4998), symétrique, donc représente (4998 x 4998) /2 = 12 490 002 distances entre couples de films.

Or, toutes ces valeurs sont dans {0, 1/40, 2/40…,39/40, 1} de taille 41. c'est très dense.

Cela empêche DBSCAN de *propager les clusters* et il arrive de passer de proche en proche en restant « *assez longtemps* » à l'intérieur du même cluster.

Confirmation par coefficient de silhouette -0.03 plus proche de 0 que de 1 insatisfaisant

Clustering hiérarchique

Vu la faible taille du dataset (4998)

expérimenter la classification avec un algorithme hiérarchique est tout à fait envisageable.

```
from sklearn.cluster import AgglomerativeClustering
ch = AgglomerativeClustering(n_clusters=50, linkage="complete", affinity="precomputed")
ch.fit(X)
```

Pas plus satisfaisant avec 50 clusters :

Coefficient de silhouette -0.01

Solution retenue

A défaut de trouver des clusters « loin les uns des autres » et « bien resserrés sur eux-mêmes » comme on aimerait avoir ! (*existent-ils dans mon data set ?*)

Et vu la taille du data set relativement faible.

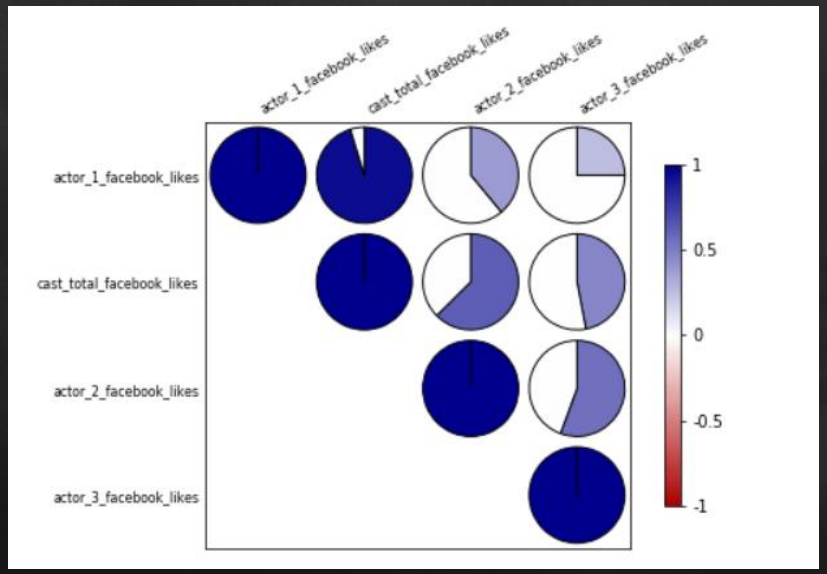
Et disposant déjà de la matrice des distances 2 à 2 entre les films

- J'ai décidé, étant donné un film F, de faire la recommandation des 5 films les plus proches directement lus dans la matrice des distances X

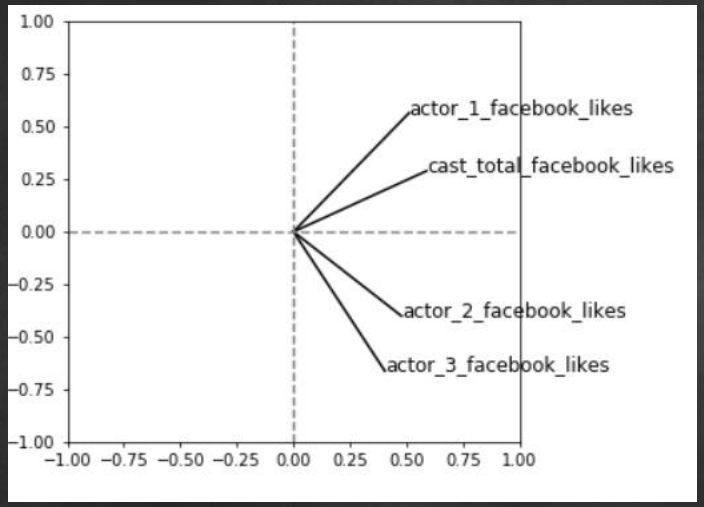
Popularité

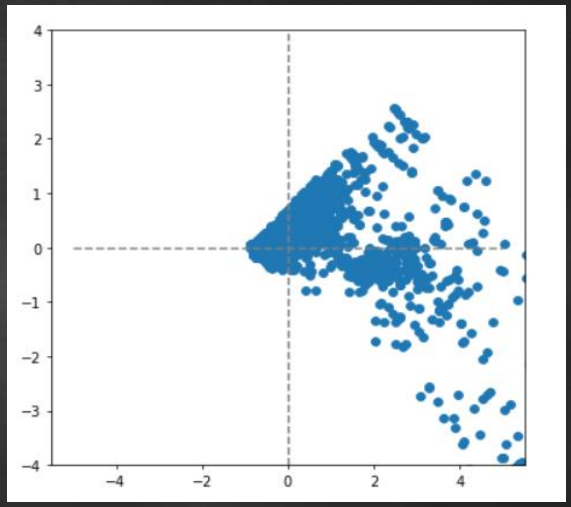
Les variables représentant les différentes «likes » Facebook

- actor_1_facebook_likes
- actor_2_facebook_likes
- actor_3_facebook_likes
- cast_total_facebook_likes



Popularité





ACP normée

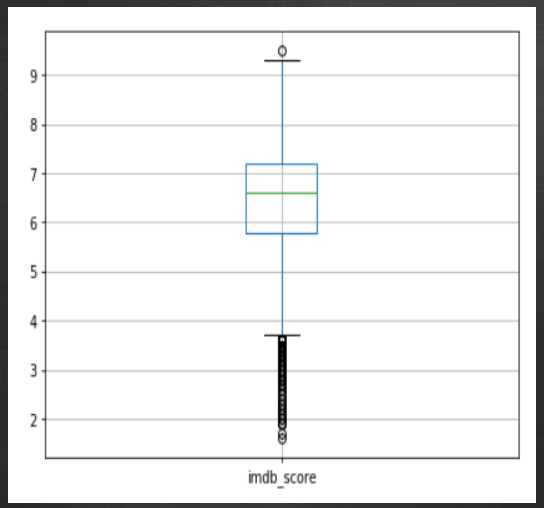
Effet « taille » , toutes ces variables sont de même signe sur le premier axe factoriel et elles sont toutes corrélées positivement entre elles .

le premier axe seul, « *explique* » presque 70% de l'information (la variance)

Ces variables peuvent être résumées par une seule > la projection sur le premier axe factoriel, je l'appelle fb_likes_scaled

Popularité

imdb_score est bien compris entre 0 et 10 et n'a pas de valeurs aberrantes.



popularity = fb_likes_scaled + imdb_scaled

(imdb_scaled est le imdb _score centré et réduit)

La fonction nearset redéfinie

La fonction *nearest* renvoie désormais les n films les plus similaires en considérant la popularité

```
In [236]: nearest(10, 5)

Out[236]: [(15, 0.75, 1.5013952648586799),
           (8, 0.82499999999999996, 11.097618668111455),
           (27, 0.82499999999999996, 8.458621532785461),
           (542, 0.82499999999999996, 0.98152911973990176),
           (65, 0.84999999999999998, 4.4314248439414339)]
```

Mise en ligne

A l'aide de la fonction, « nearest » une table movie est alimentée :

- L'id du film
- Le titre du film
- L'url IMDB du film
- Et les id des 5 films les plus proches

Mise en ligne

Une api (python/Falsk) est disponible en ligne pour pouvoir interroger le service

Appel par id :

curl -G "https://op-proj3.herokuapp.com/movie/5/nearest/" --data-urlencode "id=200"

Appel par titre :

curl -G "https://op-proj3.herokuapp.com/movie/5/nearest/" --data-urlencode "title=Harry Potter and the Sorcerer's Stone«

```
$ curl -G "https://op-proj3.herokuapp.com/movie/5/nearest/" --data-urlencode "title=Harry Potter and the Sorcerer's Stone"
[
    {
        "id": 282,
        "title": "Harry Potter and the Chamber of Secrets"
    },
    {
        "id": 193,
        "title": "Harry Potter and the Prisoner of Azkaban"
    },
    {
        "id": 114,
        "title": "Harry Potter and the Order of the Phoenix"
    },
    {
        "id": 115,
        "title": "Harry Potter and the Goblet of Fire"
    },
    {
        "id": 347,
        "title": "Percy Jackson & the Olympians: The Lightning Thief"
    }
]
```

Fin