

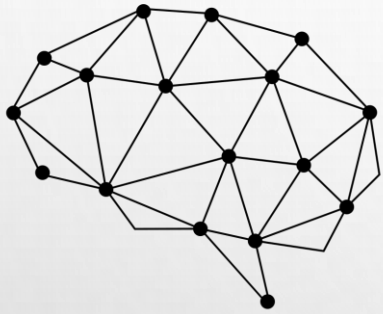
The background of the slide is a light gray gradient, decorated with numerous realistic water droplets of various sizes. Some droplets are large and prominent, while others are small and subtle, scattered across the top and bottom of the page.

# **SYSTÈMES DE RECOMMANDATION: LE FILTRAGE COLLABORATIF**

**Numéro Candidat: 40270**



Source: CNN



Cambridge  
Analytica

Source: Wikipedia

facebook

Source: Wikipedia

Systèmes de  
Recommandation

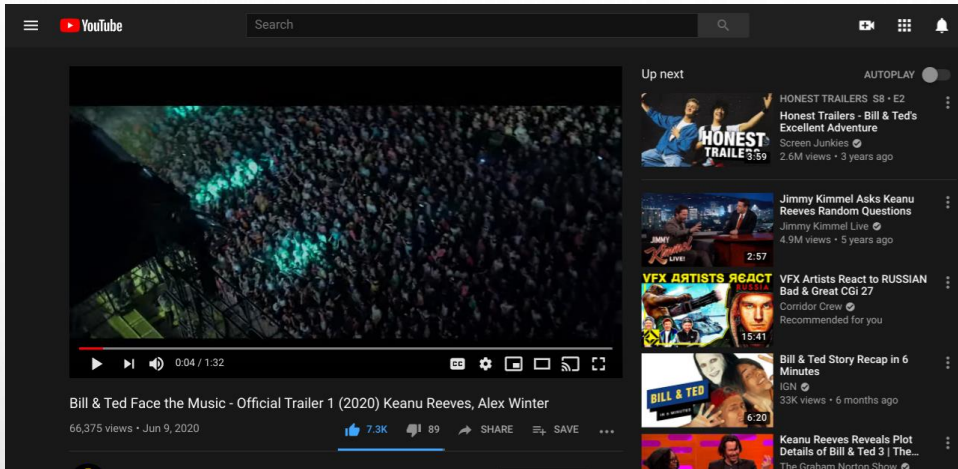


Source: New York Post



# VOUS LES CONNAISSEZ DÉJÀ...

YouTube

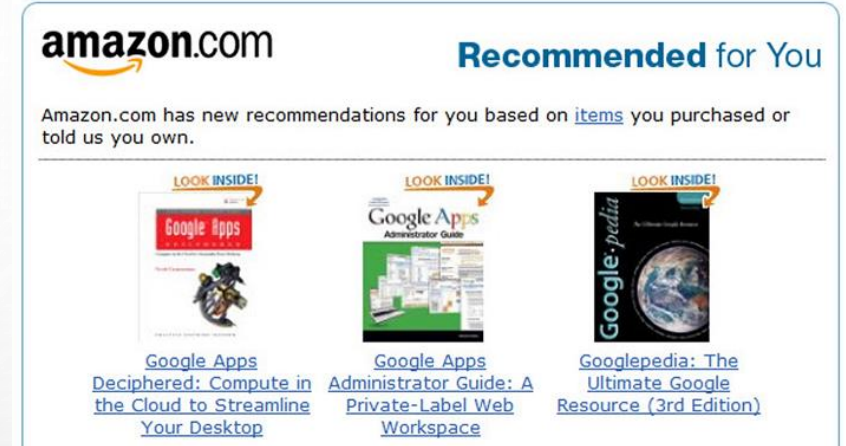


Source: YouTube

Netflix



Source: ResearchGate




Source: Mageplaza

Amazon



« Netflix Prize »

# DÉFINITION, BUT ET DÉBUT...

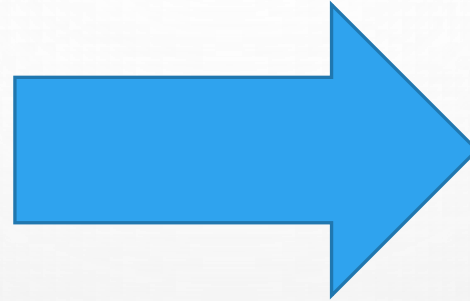


**TOP 10 SONGS**

1	GOD'S PLAN	DRAKE
2	PERFECT	ED SHEERAN
3	MEANT TO BE	BEBE REXHA & FLORIDA GEORGIA LINE
4	HAVANA	CAMILA CABELLO FEAT. YOUNG THUG
5	ROCKSTAR	POST MALONE FEAT. 21 SAVAGE
6	PSYCHO	POST MALONE FEAT. TY DOLLA SIGN
7	I LIKE IT	CARDI B, BAD BUNNY & J BALVIN
8	THE MIDDLE	ZEDD, MAREN MORRIS & GREY
9	IN MY FEELINGS	DRAKE
10	GIRLS LIKE YOU	MAROON 5 FEAT. CARDI B

billboard

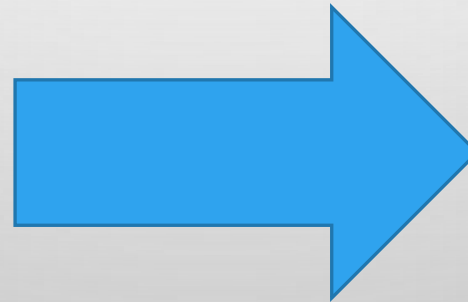
Source: Pinterest



Problème:  
Très peu de  
personnalisation...



Source: Newsdemon



Si deux personnes ont aimé  
des contenus identiques par  
le passé, elles ont une  
probabilité élevée d'aimer les  
mêmes choses dans le futur.

# NOS OBJECTIFS

- Étudier les méthodes de comparaisons entre utilisateurs:
  - Similitude Cosinus
  - Corrélation de Pearson
  - k-plus-proches voisins
  - Décomposition en Valeurs Singulières
- Concevoir et tester ces systèmes.
- Discuter les limites

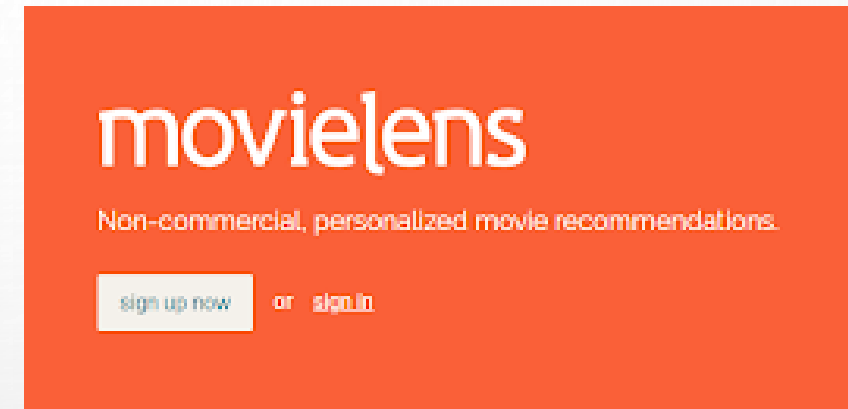
# MODÉLISATION ET BASE DE DONNÉE

$R =$

				
	4	3	0	4
	0	1	2	3
	0	2	0	4
	3	0	1	2
	0	2	0	4

Matrice des données

Source: The Conversation



Source: bookandfilmglobe

- 9724 FILMS
- 610 UTILISATEURS
- 100000 NOTES
- DEGRÉ DE PARCIMONIE:  
$$\frac{\text{Nombres de zéros}}{\text{Nombre de cases}} = 98,3\%$$

# SIMILITUDE COSINUS

Articles  $i, j$  notés en commun par  $u$  et  $v$

	1	2	3	$i$	$j$	$n-1$	$n$
1				0	$r$		
2				$r$	0		
$u$				$r$	$r$		
$v$				$r$	$r$		
$m-1$				$r$	0		
$m$							

Similarité entre les utilisateurs  $u$  et  $v$ :

$$\text{sim}(u, v) = \cos(\vec{x}_u, \vec{x}_v) = \frac{\sum_{i \in I_{uv}} r_{u,i} r_{v,i}}{\sqrt{\sum_{i \in I_{uv}} r_{u,i}^2} \sqrt{\sum_{i \in I_{uv}} r_{v,i}^2}}$$

Note prédite:

$$\text{pred}(u, i) = \frac{\sum_{w \in U_i} \text{sim}(w, u) r_{w,i}}{\sum_{w \in U_i} |\text{sim}(w, u)|}$$



# CORRÉLATION DE PEARSON

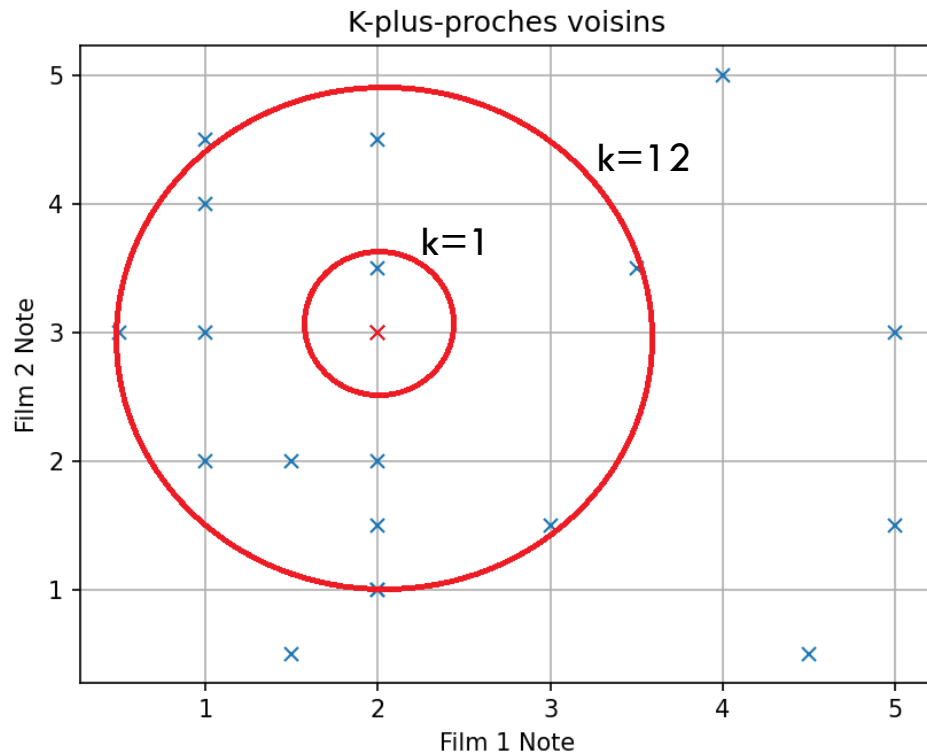
Similarité entre les utilisateurs  $u$  et  $v$ :

$$\text{sim}(u, v) = \text{Pearson}(u, v) = \frac{\sum_{i \in I_{uv}} (r_{u,i} - \bar{r}_u)(r_{v,i} - \bar{r}_v)}{\sqrt{\sum_{i \in I_{uv}} (r_{u,i} - \bar{r}_u)^2} \sqrt{\sum_{i \in I_{uv}} (r_{v,i} - \bar{r}_v)^2}}$$

Note prédite:

$$\text{pred}(u, i) = \bar{r}_u + \frac{\sum_{w \in U_i} \text{sim}(w, u)(r_{w,i} - \bar{r}_w)}{\sum_{w \in U_i} |\text{sim}(w, u)|}$$

# K PLUS PROCHES VOISINS



UNE FOIS LES K VOISINS DÉTERMINÉS, NOUS PRENONS LA MOYENNE DES NOTES SUR CE GROUPE:

$$\text{pred}(u,i) = \frac{\sum_{w \in \text{voisin}(u)} r_{w,i}}{|\text{voisin}(u)|}$$

Nous travaillons sur une partie de la matrice peu creuse, dans notre cas: les 500 premières colonnes.

# DÉCOMPOSITION EN VALEURS SINGULIÈRES TRONQUÉE

SVD complète:

$$R = D_{|U|,|U|} \Sigma_{|U|,|I|} {}^t T_{|I|,|I|}$$

Approximation via SVD tronquée:

$$R \approx R' = D_{|U|,k} \Sigma_{k,k} {}^t T_{k,|I|}$$

$$1 \leq k \leq 610$$



$$\text{pred}(u, i) = [R']_{u,i}$$

# MÉTRIQUE D'ÉVALUATION

Pour évaluer nos systèmes de recommandation, nous allons utiliser la MAE, la moyenne arithmétique des valeurs absolues des écarts.

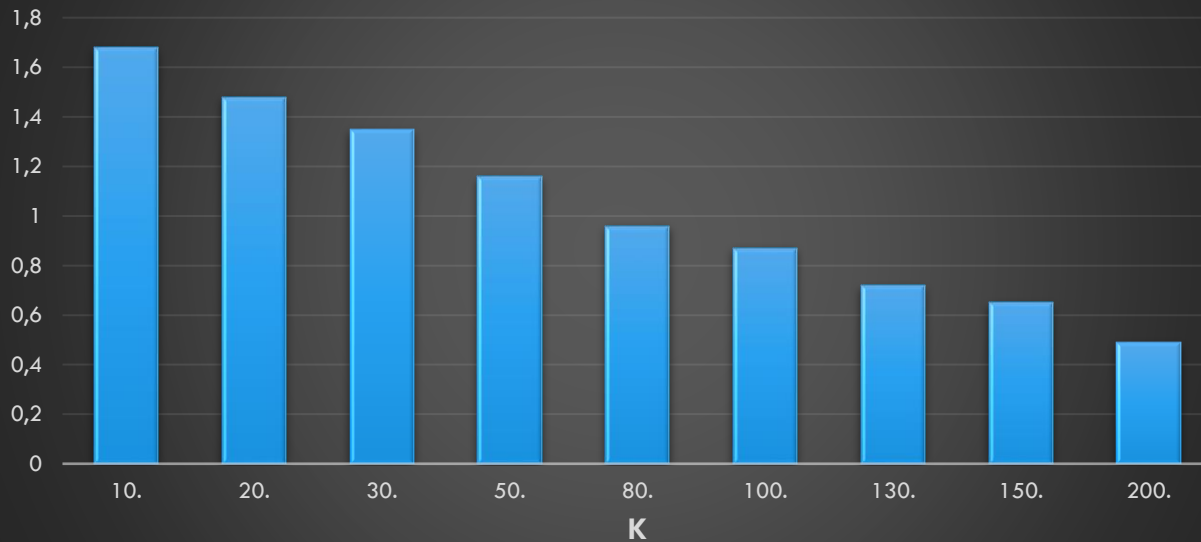
$$\text{MAE} = \frac{1}{N} \sum_{i=1}^N |r_i - \hat{r}_i|$$

Mean-Average-Error



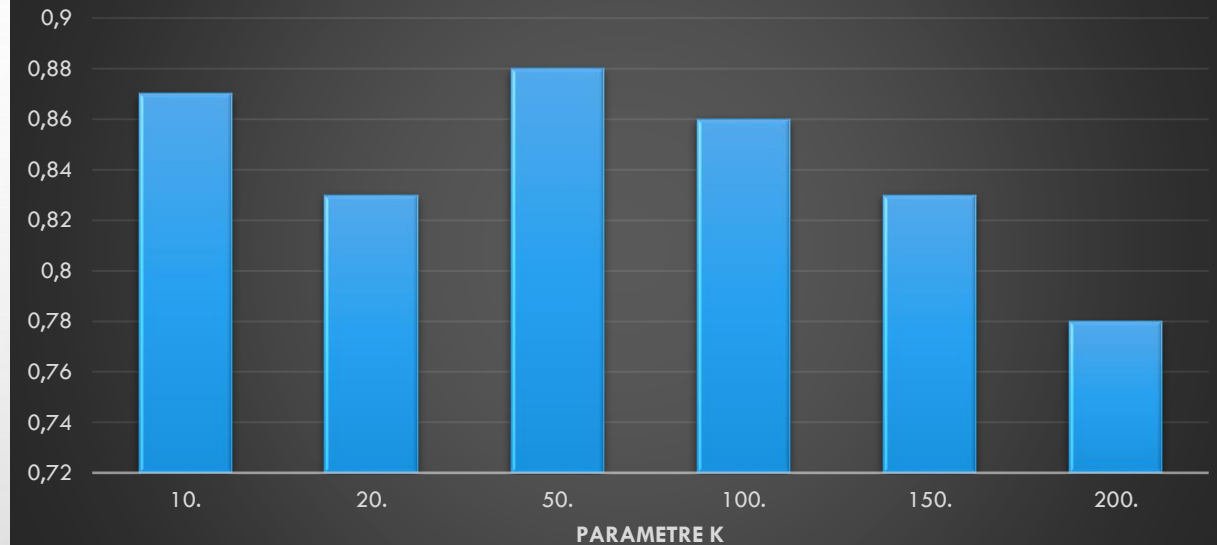
# DÉTERMINATION DU FACTEUR K

MAE EN FONCTION DE K-SVD



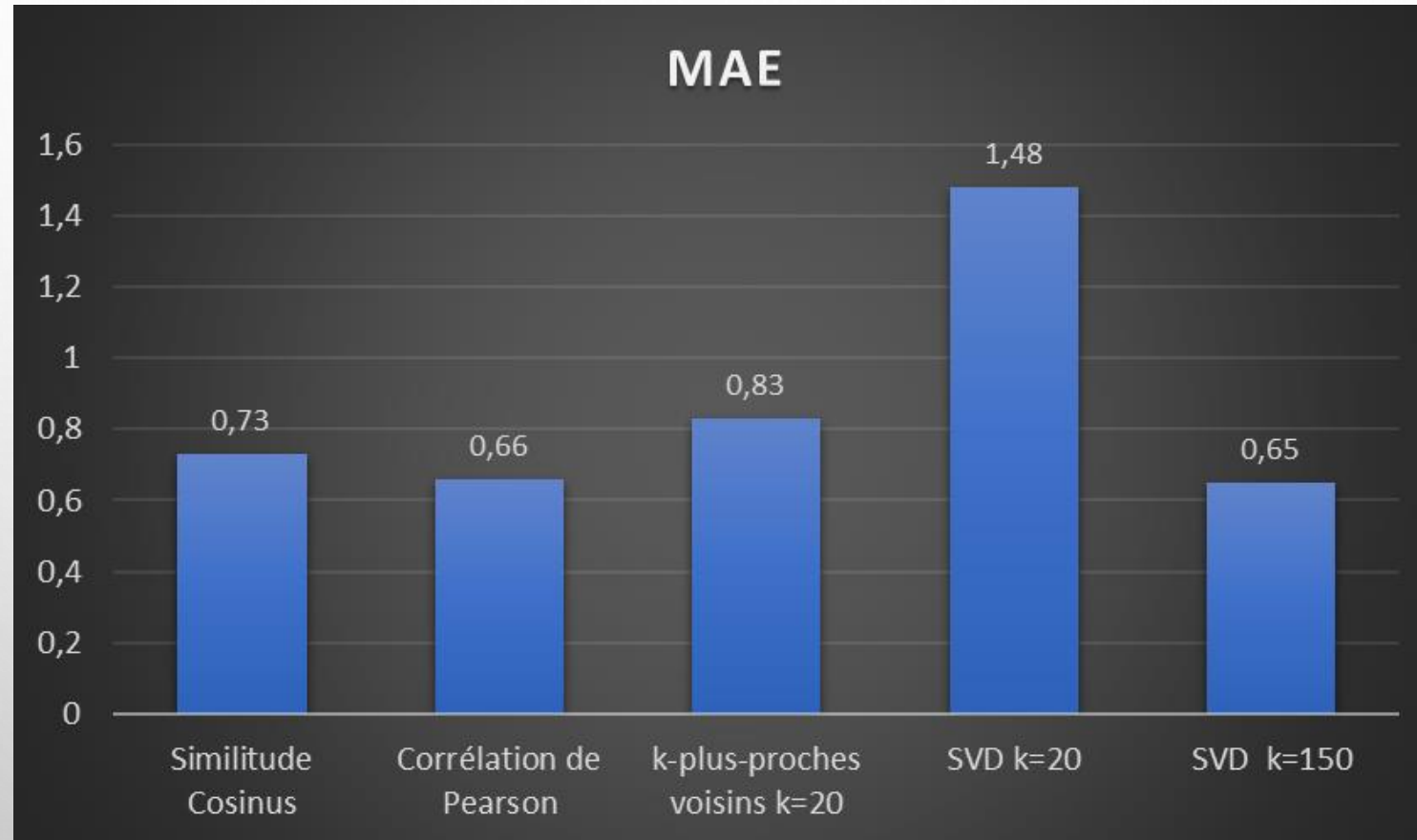
Détermination du facteur k pour le SVD

MAE EN FONCTION DE K



Détermination du facteur k pour le k-plus-proches voisins

# RÉSULTATS ET COMPARAISONS



Comparaison des systèmes de recommandation

# HYBRIDATION

$R$



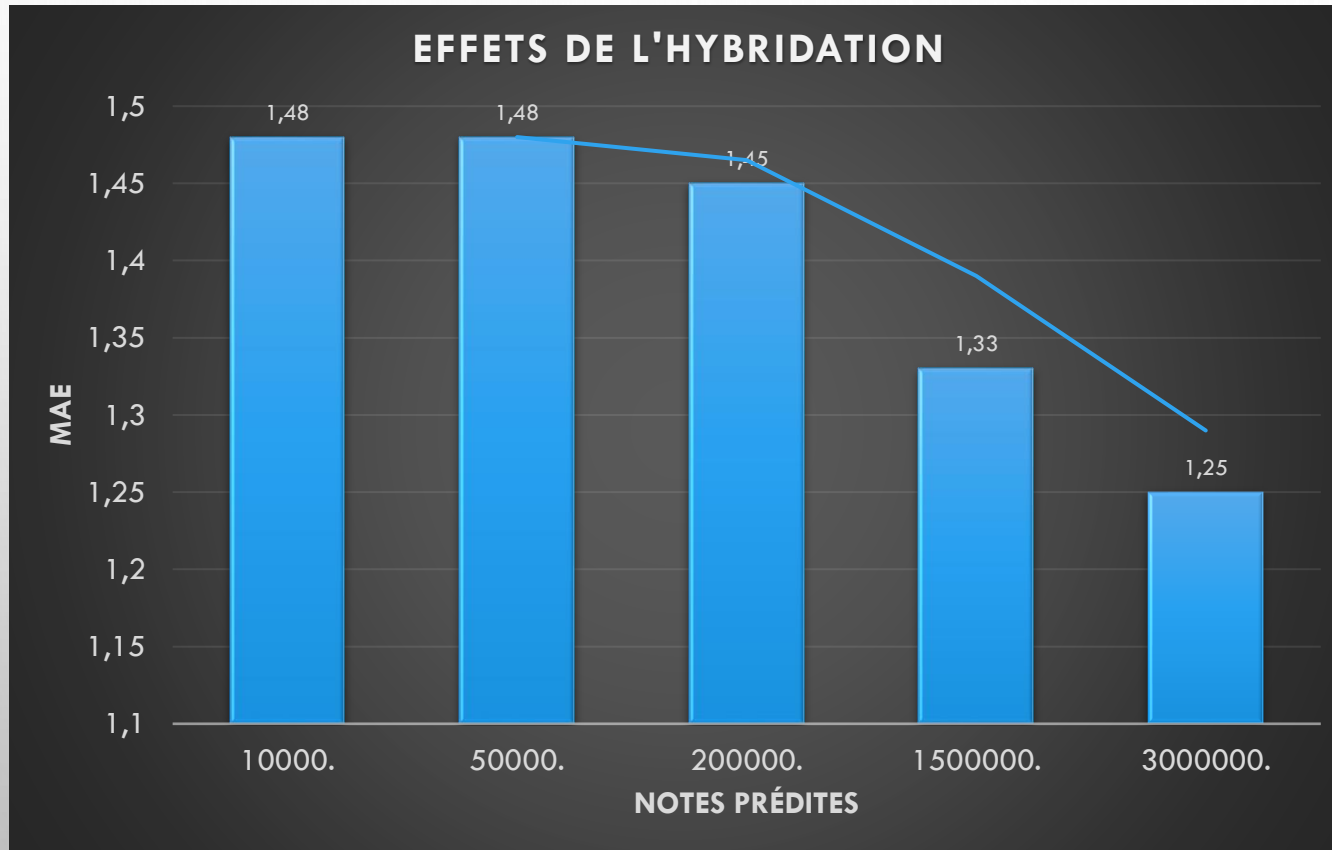
Remplissage partiel de la matrice à l'aide de la Corrélacion de Pearson



Approximation de la matrice à l'aide de la décomposition en valeurs singulières

$R'$

# RÉSULTATS DE L'HYBRIDATION

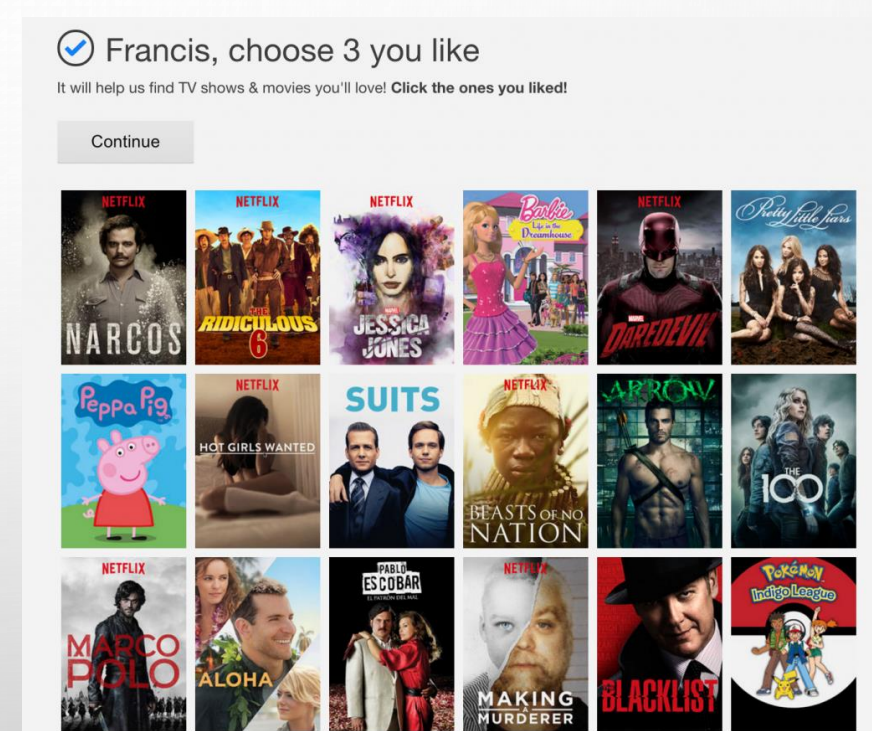


Source: Thrillist



# LIMITES ET SOLUTIONS

- DÉMARRAGE A FROID
- PARCIMONIE
- LE PROBLÈME DU MOUTON GRIS



Source: Netflix Tech Blog

# CONCLUSION ET PERSPECTIVES

- ÉTUDE ET COMPARAISON DE DIFFÉRENTS SYSTÈMES
- LIMITES
- HYBRIDATION

# DÉMONSTRATION SVD

## 1 Enoncé

Soit  $M \in \mathcal{M}_{m,n}(\mathbb{R})$

Alors il existe une factorisation de la forme:

$$M = U \Sigma^t V$$

où  $U \in \mathcal{O}_m(\mathbb{R})$  et  $V \in \mathcal{O}_n(\mathbb{R})$  et  $\Sigma \in \mathcal{M}_{m,n}(\mathbb{R})$  dont les coefficients diagonaux sont des réels positifs ou nuls (et tous les autres sont nuls).

## 2 Démonstration

Soit  $M \in \mathcal{M}_{m,n}(\mathbb{R})$

Alors  ${}^t M M$  est positive symétrique réelle, donc d'après le théorème spectral,  ${}^t M M$  est diagonalisable.

donc il existe  $V \in \mathcal{O}_n(\mathbb{R})$  tel que

$${}^t V {}^t M M V = \begin{pmatrix} D & 0 \\ 0 & 0 \end{pmatrix}$$

où  $D$  est diagonale, définie positive et de même rang  $r$  que  $M$ .

En écrivant  $V$  de façon appropriée:

$$\begin{pmatrix} {}^t V_1 \\ {}^t V_2 \end{pmatrix} {}^t M M \begin{pmatrix} V_1 & V_2 \end{pmatrix} = \begin{pmatrix} {}^t V_1 {}^t M M V_1 & {}^t V_1 {}^t M M V_2 \\ {}^t V_2 {}^t M M V_1 & {}^t V_2 {}^t M M V_2 \end{pmatrix} = \begin{pmatrix} D & 0 \\ 0 & 0 \end{pmatrix}$$

avec  $V_1$  matrice  $n \times r$  de rang  $r$  et  $V_2$  matrice  $n \times (n - r)$

Ainsi,  ${}^t V_1 {}^t M M V_1 = D$  et  $M V_2 = 0$

On pose  $U_1 = D^{-\frac{1}{2}} {}^t V_1 {}^t M$

Alors on a  $U_1 M V_1 = D^{\frac{1}{2}}$

On constate que c'est presque le résultat attendu, on a  $U_1$  une matrice  $r \times m$  telle que  ${}^t U_1 U_1 = I$

On complète  $U_1$  pour la rendre orthogonale. On choisit  $U_2$  tel que  $\begin{pmatrix} U_1 \\ U_2 \end{pmatrix}$  soit orthogonale.

Ainsi, nous avons:

$$\begin{pmatrix} U_1 \\ U_2 \end{pmatrix} M \begin{pmatrix} V_1 & V_2 \end{pmatrix} = \begin{pmatrix} D^{\frac{1}{2}} & 0 \\ 0 & 0 \end{pmatrix}$$

On utilise  $M V_2 = 0$  et on constate que  $U_2 M V_1 = U_2 {}^t U_1 U_1 M V_1 = 0$  car  ${}^t U_1 U_1 = I$  et  $U_2 {}^t U_1 = 0$  comme  $\begin{pmatrix} U_1 \\ U_2 \end{pmatrix}$  est orthogonale.

En prenant pour  $U$  la matrice transposée de  $\begin{pmatrix} U_1 \\ U_2 \end{pmatrix}$ , nous avons le résultat attendu.

# CODES

Création de la matrice à partir de la base de donnée:

```
import sqlite3
# (Id Utilisateur, Id Film, Note)
db = sqlite3.connect('ratings.sqlite3')
curseur = db.cursor()
userId = list(curseur.execute('SELECT DISTINCT userId FROM valeurs ORDER BY userId'))
movieId = list(curseur.execute('SELECT DISTINCT movieId FROM valeurs ORDER BY movieId'))
userId = [userId[i][0] for i in range(len(userId))]
movieId = [movieId[i][0] for i in range(len(movieId))]
# userId de la forme : [(1,), (2,), (3,)] en gros c'est une liste de tuple
print(len(userId), len(movieId))
#req = ' AND movieId = ' + ' AND movieId = '.join([str(movieId[i][0]) for i in range(len(movieId))])
#print(req)
matrice = []
for i in userId:
    liste = list(curseur.execute('SELECT movieId, rating FROM valeurs WHERE userId = {} ORDER BY movieId'.format(i)))
    ligne = []
    c = 0
    for j in movieId:
        if c < len(liste) and j == liste[c][0]:
            ligne.append(liste[c][1])
            c += 1
        else:
            ligne.append(0)
    matrice.append(ligne)
db.close()
```



Produit scalaire canonique et norme:

```
import math

def p_s_can(u,v):

    #produit scalaire canonique de  $R^n$ 
    #u et v sont deux vecteurs de  $R^n$ 
    n=len(u)
    resultat=0

    for i in range(n):

        resultat = resultat + u[i]*v[i]

    return resultat

def norme_euc(u): #Norme Euclidienne

    return math.sqrt(p_s_can(u,u))
```

Renvoie les vecteurs composés des composantes communes non nulles:

```
def composantes_communes(u,v):
    # Calcule les vecteurs u1 et v1 qui sont les vecteurs avec les composantes qui sont communes au vecteurs u et v

    u1=[]
    v1=[]
    n=len(u)

    for i in range(n):

        if u[i]!=0 and v[i]!=0:

            u1.append(u[i])
            v1.append(v[i])

    return u1,v1
```

Moyenne des composantes d'un vecteur:

```
def moyenne_vect(u): #Calcule la moyenne des composantes d'un vecteur

    moyenne=0
    compteur=0

    for i in u:

        if i>0: #On fait la moyenne sur les composantes non nulles

            compteur = compteur + 1

            moyenne= moyenne + i

    moyenne = moyenne / compteur

    return moyenne
```

Retranche une valeur à un vecteur:

```
def retrancher_vect(u,valeur): #retranche une valeur a toutes les composante d'un vecteur

    n=len(u)

    for i in range(n):

        u[i]= u[i] - valeur

    return u
```

## Calcule la similarité cosinus:

```
def calcul_liste_sim_cos(k,matrice):  
    #nous allons calculer la liste des similitudes de tous les utilisateurs avec l'utilisateur k ie [cos(theta1),cos(theta2),...]  
    #k est l'indice de l'utilisateur ie la ligne de la matrice  
    #matrice est la matrice utilisateur(lignes) x items(colonnes) remplie des notes  
  
    utilisateur=matrice[k] #vecteur de l'utilisateur  
    n=len(matrice) #nbre utilisateurs  
    liste_sim=[] #la ou on stocke les similitudes  
  
    for i in range(n):  
        if i!=k: #On traite tous les utilisateurs différents de celui passé en argument  
  
            u1,v1=composantes_communes(utilisateur,matrice[i])  
            norme_v1=norme_euc(v1)  
            norme_utilisateur=norme_euc(u1)  
  
            if norme_v1!=0 and norme_utilisateur!=0:  
  
                sim=(p_s_can(u1,v1))/(norme_utilisateur*norme_v1) |  
  
                liste_sim.append(sim)  
  
            else:  
  
                liste_sim.append('NaN') #Les utilisateurs n'ont pas noté de films en commun  
  
        else:  
  
            liste_sim.append('NaN') #on rajoute ca pour garder les bons indices dans la liste  
    return liste_sim
```

## Prédiction de note avec la similarité cosinus:

```
def prediction_note(item_indice,uti_indice,matrice):  
  
    #matrice : utilisateur (lignes) x items (colonnes)  
    liste_sim = calcul_liste_sim_cos(uti_indice,matrice)  
    n=len(liste_sim)  
    note=0  
    somme_sim = 0 #on ajoute la valeur absolue des similitudes pour faire une moyenne pondérée  
  
    for i in range(n):  
        if liste_sim[i]=='NaN':  
            note = note #utilisateurs non coréllés  
  
        elif matrice[i][item_indice]>0: #Nous prenons que les notes strictement positives  
            somme_sim = somme_sim + abs(liste_sim[i])  
            note = note + liste_sim[i] * matrice[i][item_indice]  
  
    if somme_sim==0: #si tous les utilisateurs en commun ( admettant une somilitude cos ) n'ont pas noté le film (donc somme sim = 0)  
        return -10  
  
    note = note/somme_sim #On normalise  
    return note
```



## Calcule la similarité de Pearson:

```
def correlation_pearson(k,matrice):
    #k est l'indice de l'utilisateur
    #matrice utilisateurs (lignes) x items (colonnes)
    #cette fonction renvoie la liste des corrélation de pearson de l'utilisateur k avec tous les autres utilisateurs
    utilisateur=matrice[k]
    moyenne_uti=moyenne_vect(utilisateur)
    liste_corr=[]
    n=len(matrice) #nombre d'utilisateurs
    for i in range(n): #parcourt les utilisateurs
        if i!=k: #on saute l'utilisateur k

            moyenne_aux=moyenne_vect(matrice[i]) #moyenne des composantes de l'utilisateur i
            u1,v1=composantes_communes(utilisateur,matrice[i]) #vecteurs avec seulement composantes en commun
            u1,v1=retrancher_vect(u1,moyenne_uti),retrancher_vect(v1,moyenne_aux) #vecteurs avec moyenne retranchée
            norme_u1=norme_euc(u1)
            norme_v1=norme_euc(v1)

            if norme_u1!=0 and norme_v1!=0:

                produit_scal=p_s_can(u1,v1)
                pearson = produit_scal/(norme_u1*norme_v1)
                liste_corr.append(pearson)

            else:

                liste_corr.append('NaN') #les utilisateurs n'ont pas notés de films en commun

        elif i==k:

            liste_corr.append('NaN') #pour garder les bons indices
    return liste_corr
```

## Prédiction de note avec la similarité de Pearson:

```
def pearson_note(item,k,matrice): # Prediction de la note a l'aide de pearson
    #k indice utilisateur
    #item indice de l'item
    #matrice utilisateurs (lignes) x items (colonne)
    liste_corr=correlation_pearson(k,matrice)
    moyenne_k=moyenne_vect(matrice[k])
    n=len(liste_corr)
    note = 0
    somme_corr = 0 #somme pour la moyenne pondérée

    for i in range(n):
        if liste_corr[i]=='NaN': #on prend que si les utilisateurs sont corrélés
            note=note

        elif matrice[i][item]>0: #on prend que les notes positives
            moyenne_aux=moyenne_vect(matrice[i])

            somme_corr = somme_corr + abs(liste_corr[i])

            note = note + liste_corr[i] * (matrice[i][item] - moyenne_aux)

    if somme_corr==0: #si tous les utilisateurs en commun ( admettant une similitude pearson ) n'ont pas noté le film (donc somme sim = 0)
        return -10

    note = note/somme_corr + moyenne_k

    return note
```

Calcule le vecteur UV:

```
def vect_uv(u,v): #Calculer le vecteur UV avec les points/vecteurs u et v

    vecteur=[]
    n=len(u)

    for i in range(n):
        vecteur.append(u[i]-v[i])

    return vecteur
```

Calcule la liste des distances à un utilisateur:

```
def liste_distances(indice_uti,matrice):

    u=matrice[indice_uti]
    n=len(matrice)
    list_dist=[]

    for i in range(n):

        if i!=indice_uti:

            vecteur=vect_uv(u,matrice[i])
            distance=norme_euc(vecteur)
            list_dist.append([distance,i])

        else:
            list_dist.append([0,indice_uti])

    return list_dist
```

Détermine les k-plus-proches voisins:

```
def scinde(liste):
    milieu=len(liste)//2
    Tg=liste[:milieu]
    Td=liste[milieu:]
    return Tg,Td

def fusion(L1,L2):
    if L1==[]:
        return L2
    if L2==[]:
        return L1
    if L1[0][0]<L2[0][0]:
        return [L1[0]]+fusion(L1[1:],L2)
    else:
        return [L2[0]]+fusion(L1,L2[1:])

def tri_fusion(liste):
    if len(liste)<=1:
        return liste
    else:
        Tg,Td=scinde(liste)
        Tg,Td=tri_fusion(Tg),tri_fusion(Td)
        return fusion(Tg,Td)

def k_plus_petits(liste,k):
    liste1=tri_fusion(liste)
    return liste1[0:k]
```

Prédiction de note à l'aide des k-plus-proches voisins:

```
def KNN(k,indice_uti,matrice):
    candidats=matrice
    liste=liste_distances(indice_uti,candidats)
    resultat=k_plus_petits(liste,k)
    return resultat

def knn_prediction(indice_item,indice_uti,matrice,k=20):
    liste_knn=KNN(k,indice_uti,matrice)
    n=len(liste_knn)
    note=0
    compteur=0
    for i in range(n):
        indice=liste_knn[i][1]
        if matrice[indice][indice_item]>0:
            note = note + matrice[indice][indice_item]
            compteur=compteur + 1

    if compteur ==0:
        return -10
    note=note/compteur
    return note
```

## SVD tronquée:

```
import numpy as np
import scipy.sparse as sp
from scipy.sparse.linalg import svds

matrice=np.array(matrice)

u, s, vt = svds(matrice, k = 300)
s_diag_matrix=np.diag(s)
n_users=len(matrice)
n_items=len(matrice[0])

X_pred = np.dot(np.dot(u, s_diag_matrix), vt)

print(X_pred[0][:20])

import math
x = np.zeros((n_users, n_items))
for i in range(0,n_users):
    a=max(X_pred[i])
    b=min(X_pred[i])
    c=0
    d=5
    for j in range(0,n_items):
        x[i][j]=((X_pred[i][j]-b)/(a-b))*d
```

## Exemple de calcul de MAE pour la SVD:

```
def maetest(m_reel,m_pred):
    compteur=0
    somme=0
    for i in range(len(m_reel)):
        for j in range(n_items):
            if m_reel[i][j]!=0:
                somme = somme + abs(m_reel[i][j]-m_pred[i][j])
                compteur=compteur+1
    print(somme)
    print(compteur)
    somme=somme/compteur
    return somme
```

## Exemple de calcul de MAE avec similarité cosinus:

```
def verifie(j): #verifie qu'il n'y ai pas juste une seule note dans la colonne sinon nous ne pouvons pas predire de note
    verification=0
    for i in range(len(matrice)):
        if matrice[i][j]>0:
            verification=verification+1
        if verification>1:
            return 5
    return 1
c,liste_prediction = 0,[]
import random
liste_id,vrai_notes = [],[]
NB_TEST = 20000
#on choisit aléatoirement NB_TEST notes
for i in range(NB_TEST):
    x,y = (random.randint(0,len(matrice)-1),random.randint(0,len(matrice[0])-1))
    while matrice[x][y] == 0 or verifie==1:
        x,y = (random.randint(0,len(matrice)-1),random.randint(0,len(matrice[0])-1))
    liste_id.append((x,y))
    vrai_notes.append(matrice[x][y])
    matrice[x][y] =0
    liste_prediction.append([prediction_note(y,x,matrice),vrai_notes[c]])
    matrice[x][y]=vrai_notes[c]
    c=c+1
mae=0
compteur=0
for i in range(len(liste_prediction)):
    if liste_prediction[i][0]!=(-10):
        mae=mae+abs(liste_prediction[i][0]-liste_prediction[i][1])
        compteur=compteur+1
mae=mae/compteur
print('le nombre de notes predit en vrai:',compteur)
print('le mae est:',mae)
```



## Extrait de code pour l'hybridation:

```
nombreuti=len(matrice)
nombreitem=len(matrice[0])
matrice_reelle=copy.deepcopy(matrice)  #JE COPIE LA MATRICE D'ORIGINE
#JE VAIS CALCULER LA MATRICE DE SIMILITUDE POUR EVITER LA COMPLEXITE
matrice_de_similitude=[]
for i in range(nombreuti):
    liste_sim_i=correlation_pearson(i,matrice)
    matrice_de_similitude.append(liste_sim_i)
#La matrice de similitude est remplie
matrice_de_notes_predite=[[0 for i in range(nombreitem)] for j in range(nombreuti)]
#on a besoin de cette matrice car si nous rajoutons a fur et a mesure cela fausse les calculs
c=0
liste_prediction=[]
import random
liste_id,vrai_notes = [],[]
NB_TEST = 400000
for i in range(NB_TEST):
    x,y = (random.randint(0,len(matrice)-1),random.randint(0,len(matrice[0])-1))
    while matrice[x][y] != 0 :
        x,y = (random.randint(0,len(matrice)-1),random.randint(0,len(matrice[0])-1))
    liste_id.append((x,y))
    vrai_notes.append(matrice[x][y])
    note_predite=pearson_note(y,x,matrice)
    matrice_de_notes_predite[x][y]=note_predite
    c=c+1
#On rajoute les notes predites a la matrice d'origine 'matrice'
compteur2=0
for i in range(nombreuti):
    for j in range(nombreitem):
        if matrice_de_notes_predite[i][j]>0:
            compteur2=compteur2+1
            matrice[i][j]=matrice_de_notes_predite[i][j]
```