

# MOTEURS DE RECOMMANDATION (Recommender systems)

# INTRODUCTION

**Objectif** : proposer aux utilisateurs des choix pertinents.

**Applications nombreuses:**

- Produits culturels (Amazon, Netflix, ...)
- Publicité (Criteo, Facebook, ...)
- Réseaux sociaux (Facebook, LinkedIn, ...)
- Sites de rencontres (Meetic, Tinder, ...)

**Principe intuitif** : utilisation des caractéristiques de l'utilisateur et/ou des produits pour proposer des choix parmi un catalogue de choix possibles.

# EXEMPLE

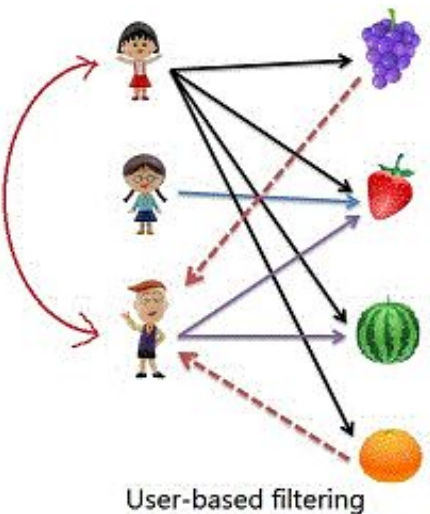
Quelle méthode utiliser pour proposer à des étudiants M2 info des cours qui les intéressent?

- Ceux qui ont pris ces cours ont pris ce cours...
- Fouille a des **mots-clés similaires** à ceux du cours X: si on aime X, on devrait aimer Fouille.
- L'utilisateur **ressemble** à un utilisateur qui a aimé le cours de Fouille, il devrait l'aimer aussi.
- L'utilisateur **aime les mêmes cours** qu'un autre qui a aimé Fouille, il devrait l'aimer aussi.

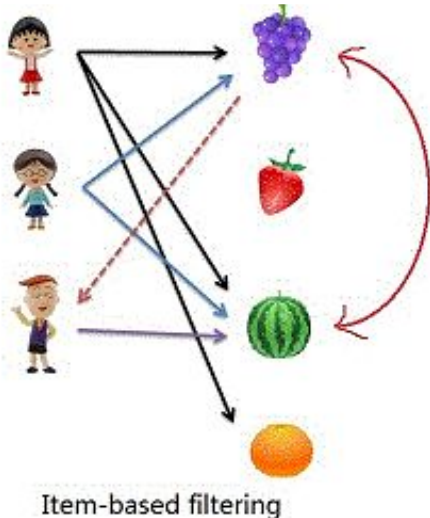
**Différents systèmes, tous pertinents mais pas forcément dans les mêmes contextes !**

# DEUX MODELES PRINCIPAUX

- **User-based** : basé sur les utilisateurs
- **Content- ou Item-based** : basé sur le contenu



Source : paxcel.net



# ALGORITHMES BASÉS SUR LE CONTENU

# PRINCIPE

0.98   0.75   -0.3



|         |   |   |   |   |
|---------|---|---|---|---|
| Camille | 4 | 5 | ? | 4 |
| Mehdi   | 1 | ? | 5 | ? |
| Julien  | 5 | ? | ? | 2 |

**Donnée: Profil-utilisateur**, i.e. contenus qu'il a noté (\*). On évalue la note des autres contenus grâce à leur similarité aux contenus connus.

# MODELISATION DES CONTENUS

On cherche à **caractériser** un contenu. C'est du feature engineering!

- Texte + **TF-IDF** : Titre; description; mots-clés (avec importances différentes!)
- Données catégorielles: Genre, ...
- Données numériques: Année de sortie, durée, box office, ...

De manière générale, un peu la même problématique que façonner une “bonne” fonction de distance!

# EXEMPLE

Un étudiant aime le cours de Fouille (*normal!*).  
Quel autre cours lui recommander ?

FDD, **comparé aux autres**, a de **fortes** valeurs TF-IDF pour les mots : données, fouille, apprentissage, statistiques, et de **faibles** valeurs TF-IDF pour: algorithme, systèmes, architecture.

On mesure la **similarité** entre le TF-IDF du cours "Fouille" avec celui de chacun des autres cours.

On propose à l'étudiant des cours similaires, comme "Intelligence Artificielle", qui a un **profil** de contenu similaire.



# CALCUL DE LA SIMILARITE

| TF-IDF   | algo | système | stats | données |
|----------|------|---------|-------|---------|
| Fouille  | 0.1  | 0.05    | 3.5   | 3       |
| I.A.     | 0.1  | 1.2     | 2     | 2       |
| Systèmes | 0.05 | 4       | 0     | 1       |

*(Données fictives)*

Par exemple, on peut mesurer une **distance** entre chaque couple de cours, e.g. distance euclidienne basée sur le TF-IDF.

On peut calculer d'autres **similarités**, que nous allons voir maintenant.

# SIMILARITE “COSINUS”

Si  $X$  et  $Y$  sont deux vecteurs (TF-IDF ou autre, ... tout feature vector), leur **similarité cosinus** est :

$$S(X, Y) = \frac{\langle X, Y \rangle}{\|X\| \times \|Y\|} = \frac{\sum_{i=1}^d x_i y_i}{\sqrt{\sum_{i=1}^d x_i^2 \times \sum_{i=1}^d y_i^2}}$$

Où:

- $\langle X, Y \rangle$  est le produit scalaire entre  $x$  et  $y$ :  
 $\langle \text{FDD}, \text{IA} \rangle = 0.1 \times 0.1 + 0.05 \times 1.2 + 3.5 \times 2 + 3 \times 2$
- $\|X\|$  est la norme de  $X$ , i.e. sa “taille”.
- Le numérateur est grand si  $X$  et  $Y$  ont de grandes valeurs aux mêmes endroits.
- Division par  $\|X\| \times \|Y\|$  pour éviter un effet

# SIMILARITE COSINUS - SUITE

---

Au tableau:

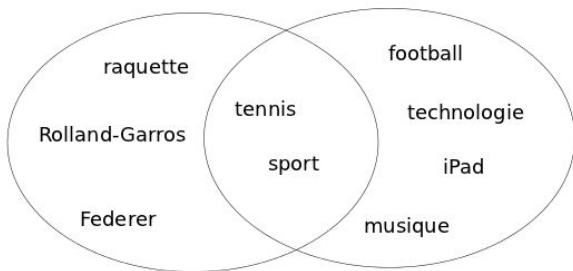
- Pourquoi ça s'appelle "Cosinus"
- Quelles valeurs prend-elle ?  
Comment interpréter ces valeurs ?
- Quand choisit-on la similarité Cosinus?

# SIMILARITE DE DICE-SORENSEN

Si A et B sont deux ensembles de mots-clés, la **similarité de Dice** entre eux est :

$$S(A, B) = \frac{2|A \cap B|}{|A| + |B|} = 2 \times \frac{\text{Nombre de mots en commun}}{\text{Nombre total de mots}}$$

Objet décrit par des mots-clé



Exemple: ici,

$$S(A, B) = \frac{2 \times 2}{5 + 6}$$

# MÉTHODE 0

$$note(d) = \frac{\sum_{d_i \in \text{Notés}} note(d_i) * similarité(b, b_i)}{\sum_{d_i \in \text{Notés}} similarité(b, b_i)}$$

Où:

- “d” est un nouveau contenu à noter,
- “Notés” est l’ensemble des contenus  $\{d_i\}$  déjà notés par l’utilisateur,
- $similarité(x, y)$  est la fonction de similarité choisie.

# MÉTHODE 0

$$note(d) = \frac{\sum_{d_i \in \text{Notés}} note(d_i) * similarité(b, b_i)}{\sum_{d_i \in \text{Notés}} similarité(b, b_i)}$$

Où:

- “d” est un nouveau contenu à noter,
- “Notés” est l’ensemble des contenus  $\{d_i\}$  déjà notés par l’utilisateur,
- $similarité(x, y)$  est la fonction de similarité choisie.

Problème: **Complexité**

# MÉTHODE 1

On veut créer un “profil utilisateur” sous forme de feature vector [pourquoi ??].

Nos données: il a "liké" quatre documents A, B, C et D. Faisons la **moyenne** de ce qu'il aime:

$$P = \frac{TFIDF(A) + TFIDF(B) + TFIDF(C) + TFIDF(D)}{4}$$

On obtient un vecteur P de ses **goûts moyens**.

Ce vecteur P est celui qui est comparé aux nouveaux contenus: les plus proches sont proposés (*recommandés*) à l'utilisateur.

# MÉTHODE 1

On veut créer un "profil utilisateur" sous forme de feature vector [pourquoi ??].

Nos données: il a "liké" quatre documents A, B, C et D. Faisons la **moyenne** de ce qu'il aime:

$$P = \frac{TFIDF(A) + TFIDF(B) + TFIDF(C) + TFIDF(D)}{4}$$

On obtient un vecteur P de ses **goûts moyens**.  
Ce vecteur P est celui qui est comparé aux nouveaux contenus: les plus proches sont proposés (*recommandés*) à l'utilisateur.

Problème: Si A, B, C et D sont **très différents**?



## MÉTHODE 2

On commence par **clusteriser** les contenus passés.  
Par exemple, on trouve les clusters  
 $\{A, B\}$  et  $\{C, D\}$ .

Pour chaque nouveau contenu, on regarde la similarité avec chaque **cluster**, en utilisant la méthode que l'on veut : distance min, max, moyenne, Ward... ou encore similarité cosinus avec la moyenne (centroïde) de chaque cluster.

- On moyenne des contenus **comparables**
- On réduit la complexité (VS méthode 0)

# COMMENT CHOISIR LA SIMILARITÉ ? ET L'ENCODAGE ?

---

Comme toujours en Apprentissage : Grâce à l'**évaluation** sur des vraies données!

(on essaye différentes recettes, on voit ce qui marche bien et moins bien, on itère)

Ici: on peut évaluer à quel point une fonction de similarité donne de bons résultats pour la prédiction

# RÉ-ALIMENTATION; COLD START

**Algo:** Recommandation basée sur le contenu....

1. Initialisation:  $X = \{\text{vecteurs}\}$  représentant le profil:  
TF-IDF des contenus passés aimés par l'utilisateur
2. **for each** visite sur le site (Netflix, Amazon, ...)
3. Recommander des produits à l'utilisateur avec la méthode 0, 1, 2, ou autre, en fonction de  $X$ .
4. Pour tous ceux que l'utilisateur a noté (\*), les ajouter à  $X$

**Problème:** Étape 1, le "*Cold Start*".

# RECOMMANDATION PAR CONTENU:

## CONCLUSION

### Avantages :

- fait des propositions personnalisées.
- ne requiert pas la participation de milliers d'utilisateurs.

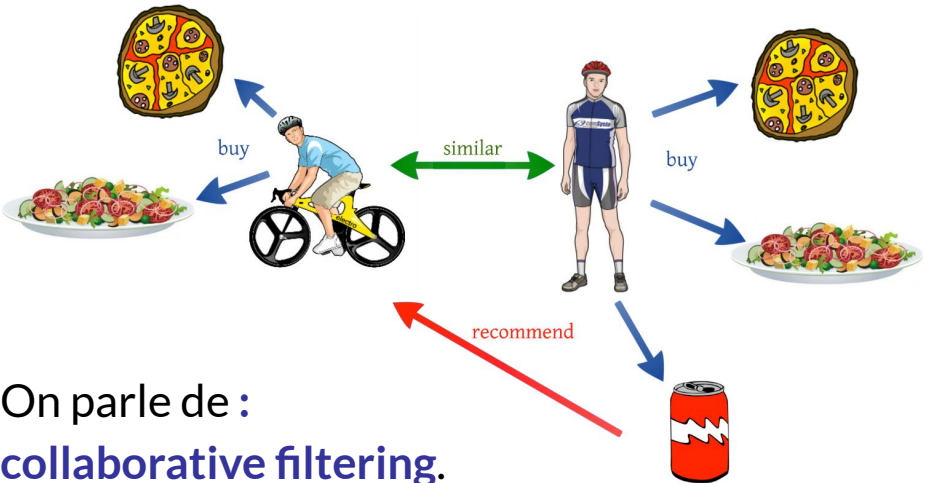
### Inconvénients :

- ne se base que sur le profil-utilisateur, ne prend pas en compte les autres utilisateurs.
- ne répond pas au problème du cold-start.
- Certains types de contenus sont difficiles à comparer, ou, pire, à convertir en *feature*.

# ALGORITHMES BASÉS SUR LES UTILISATEURS

# PRINCIPE

Les utilisateurs ayant aimé les mêmes choses que vous dans le passé continuent à aimer les mêmes choses que vous dans le futur.



# PROBLEME DE DEPART



|         |   |   |   |   |
|---------|---|---|---|---|
| Camille | 4 | 5 | ? | 4 |
| Mehdi   | 1 | ? | 5 | ? |
| Julien  | 5 | ? | ? | 2 |

On cherche à remplir les cases **inconnues**.

# SIMILARITE DES UTILISATEURS

Si x et y sont 2 utilisateurs représentés par leur vecteur de goûts, alors la similarité entre eux est mesurée par le **coefficient de corrélation** :

$$\rho(x, y) = \frac{\sigma_{xy}}{\sigma_x \sigma_y} = \frac{\sum_i (x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_i (x_i - \bar{x})^2} \sqrt{\sum_i (y_i - \bar{y})^2}}$$

- $\sigma_x$  et  $\sigma_y$  sont les **écarts-type** de x et y.
- $\sigma_{xy}$  est la **covariance** entre x et y: elle décrit le comportement d'une des deux variables en fonction de l'autre.
- La **corrélation**  $\rho$  est entre -1 (opposés absolus) et 1 (identiques).



# GESTION DES DONNEES MANQUANTES

Comment mesurer la corrélation entre deux..... utilisateurs si certaines "cases" sont vides?

**Solution 1:** ne prendre en compte que les "lignes" complètes pour les deux utilisateurs.

**Solution 2:** remplacer les cases manquantes par la moyenne (⚠).

|         |  |  |  |  |
|---------|--|---|---|---|
| Camille | 4  | 5   | ?   | 4   |
| Mehdi   | 1  | ?   | 5   | ?   |
| Julien  | 5  | ?   | ?   | 2   |

D'autres solutions existent, mais + compliquées !

# PROPOSITION DE CONTENUS

On commence par **calculer la corrélation** entre l'utilisateur-cible et tous les autres.

Les valeurs de corrélation sont alors utilisées comme des poids pour calculer une **moyenne pondérée** de leurs notes (ratings) pour chaque nouveau contenu.

Cette moyenne pondérée est utilisée pour **prédire une note** utilisateur-cible/nouveau contenu.

# PROPOSITION DE CONTENUS

---

$$note(d, u) = \frac{\sum_{u' \in \text{Users}} note(d, u') \times \rho(u, u')}{\sum_{u' \in \text{Users}} \rho(u, u')}$$

Problème de complexité ?

→ mêmes techniques qu'avant (Méthode 2, clustering), ou **sampling** (on en reparlera).

# EXEMPLE

| Like?    | Étudiant 0 | Étudiant 1 | Étudiant 2 |
|----------|------------|------------|------------|
| Systèmes | 0          | 1          | 0          |
| BDD      | 1          | 1          | 1          |
| Fouille  | 1          | 0          | ?          |
| I.A.     | 1          | 0          | 1          |
| XML      | 1          | 0          | 0          |

L'étudiant 2 va-t-il aimer le cours de Fouille ?

```
import numpy as np
E = np.array([[0, 1, 1, 1],
              [1, 1, 0, 0], [0, 1, 1, 0]])
C = np.corrcoef(E)
rating = C[0, 2]*1+C[1, 2]*0
```

→ 0.6

Correlation matrix  $C$  (green text):

$$C = \begin{bmatrix} 1. & -0.6 & 0.6 \\ -0.6 & 1. & 0. \\ 0.6 & 0. & 1. \end{bmatrix}$$

# EXEMPLE

| Like?    | Étudiant 0 | Étudiant 1 | Étudiant 2 |
|----------|------------|------------|------------|
| Systèmes | 1          | 3          | 0          |
| BDD      | 5          | 4          | 4          |
| Fouille  | 4          | 1          | ?          |
| I.A.     | 4          | 2          | 1          |
| XML      | 5          | 1          | 1          |

L'étudiant 2 va-t-il aimer le cours de Fouille ?

```
import numpy as np
E = np.array([[1, 5, 4, 5],
              [3, 4, 2, 1],
              [0, 4, 1, 1]])
C = np.corrcoef(E)
rating = C[0,2]*4+C[1,2]*1
```

→

$$\begin{bmatrix} 1. & -0.2 & 0.7 \\ -0.2 & 1. & 0.6 \\ 0.6 & 0.7 & 1. \end{bmatrix}$$

→ 3.2

# PROBLEME DE COLD START

Ici, le problème de **cold-start** revient à avoir une ligne entière vide (cold-start item) ou une colonne entière vide (cold-start user).

On peut:

- Demander à un nouvel utilisateur de donner des ratings **explicites**
- Proposer un nouvel item à des utilisateurs au



# PROBLEMES DE CALCUL

Si, comme Netflix, on a 100M+ utilisateurs, le calcul **en temps réel** devient trop lourd:

- On peut **précalculer off-line** les similarités
- On choisit les K utilisateurs **plus proches voisins** (K-nearest neighbors), c'est-à-dire les K utilisateurs ayant les plus fortes corrélations avec l'utilisateur-cible. On ne calcule le rating qu'à partir de ces K utilisateurs.
- **Variante** : on sélectionne les utilisateurs ayant une corrélation  $\rho > C_{min}$  avec l'utilisateur-cible.
- **Plus compliqué**: Clusters d'utilisateurs.


# USER-CENTRIC VS ITEM-CENTRIC

L'approche qu'on vient de voir compare d'abord les utilisateurs. On l'appelle **user-centric**.

On peut, à l'inverse, calculer les similarités sur les contenus (**item-centric**) : 2 contenus avec les mêmes ratings utilisateurs ont une similarité 1.

→ On calculera  $\text{note}(d, u)$  en fonction de  $\{\text{notes}(\mathbf{d}', u)\}$  (au lieu de  $\{\text{notes}(d, \mathbf{u}')\}$ ).

+ économe: temps réel. Popularisée par **Amazon**.

 : Cette approche est différente de la recommandation par contenu seulement qui n'utilise pas les notes des **autres** utilisateurs.





# COLLABORATIVE FILTERING:

## CONCLUSION

### Avantages:

- Sans connaissance sur le contenu (distances, features) : on ne se base que sur les notes des utilisateurs.
- Donc plus facile à implémenter. Et **puissant**.

### Inconvénients:

- Calculs lourds.
- Problèmes de cold-start non résolus.
- **Sparsity** (parcimonie, rareté). E.g. Amazon: des millions de produits, mais chaque utilisateur ne notera qu'une minuscule partie →  algo 

# AUTRES MÉTHODES

# RECOMMANDATION PERSONALISEE

Approche basée sur le comportement passé de l'utilisateur (clics, cookies, likes...).

On va simplement lui recommander des contenus en fonction de ce qu'il a déjà cherché.

→ Approches plus ad-hoc, algorithmiquement similaires à la recommandation par contenu mais où l'input n'est pas des notes de contenus existants.

**Exemples** : Criteo, Facebook Ads, AdSense, ...

# RECOMMANDATION HYBRIDE

La **recommandation hybride** se base sur les 3... approches précédentes, et combine donc actions passées de l'utilisateur, similarités entre contenus et collaborative filtering.

Plus performante que les autres prises seules, et a plus d'armes pour répondre aux problèmes de cold-start et de sparsity.

**En pratique:** par exemple pré-remplir la matrice de collaborative filtering avec un algo basé sur le contenu et l'historique de l'utilisateur.

→ Approche utilisée par Amazon, Netflix, ...

# EXAMPLE: YOUTUBE OU NETFLIX

**Recommandation personnalisée** : en fonction de l'historique et de nos caractéristiques sociales (localisation, âge, etc.)

**Recommandation basée sur le contenu** : les vidéos ont des tags similaires

**Collaborative filtering** : les utilisateurs qui ont aimé la vidéo que vous regardez ont aussi aimé ces autres vidéos.

**Concrètement, on peut mettre des poids à chacun de ces types pour le ranking final.**

# LE PRIX NETFLIX (2007-2009)

cf [[wikipedia](https://fr.wikipedia.org/wiki/Le_Prix_Netflix)]. Prix: **1'000'000 \$** si  $\geq 10\%$  mieux

**480K** users, **18K** movies, **100M** ratings.

Rating: (user, movie, date, grade), grade  $\in \{1..5\}$ .

Train: 99M, Test: 1.5M. Evaluation: simple!

Vainqueurs: *"Our experience is that most efforts should be concentrated in deriving substantially different approaches, rather than refining a single technique. Consequently, our solution is an ensemble of many methods."*

# ÉVALUATION

# EVALUATION

---

- Cas “simple”: prédiction de la note  
 $\text{note} = f(\text{user}, \text{movie})$   
→ au tableau
- Cas compliqué: on a pas de ‘note’, on veut simplement **évaluer la liste** de recommandations



# EVALUATION DE LISTE

Deux métriques nous intéressent:

**RECALL:** Parmi les contenus que user  $U$  a aimés, combien étaient dans la liste recommandée ?

**PRÉCISION:** Dans la liste recommandée, combien l'utilisateur  $U$  en a-t-il aimé?

L'**algorithme parfait** a précision=1, recall=1: non seulement la personne adore tout ce que vous lui proposez (précision maximale) mais en plus elle n'aime rien d'autre (recall maximal).

**Problème:** pour faire augmenter l'un, on prend le risque de baisser l'autre.

# EVALUATION DE LISTE

| Je propose:        | Score: | L'utilisateur clique sur: |
|--------------------|--------|---------------------------|
| - chat mignon      | 0.95   | - chat au foot            |
| - chat possédé     | 0.92   | - chat mignon             |
| - chat qui tombe   | 0.9    | - chat et chien           |
| - chat qui vole    | 0.8    | - chat qui tombe          |
| - chat au foot     | 0.7    | ... 10 autres vidéos      |
| ... 100 autres ... | >0.6   | hors de ma liste          |
| - chat et chien    | 0.6    | ou de score <0.6          |

|               |                  |                     |
|---------------|------------------|---------------------|
| Au seuil 0.9, | recall = 2 / 14, | précision = 2 / 3   |
| Au seuil 0.7, | recall = 3 / 14, | précision = 3 / 5   |
| Au seuil 0.6, | recall = 4 / 14, | précision = 4 / 106 |

# EVALUATION DE LISTE

---

On peut prendre le problème à l'envers:

Pour avoir une précision de 0.9, combien faut-il que je recommande d'objets?

Et est-ce possible dans l'absolu? Cela peut déterminer la taille de votre liste.

# EVALUATION DE LISTE ORDONNEE

D'autres métriques sont utilisées pour renforcer l'idée que le haut de la liste est plus important:



$$\text{AP (Average Precision)} = \frac{1}{3}(1/1 + 2/2 + 3/5) = 0.87$$

**NDCG (Normalized Discounted Cumulative Gain):**

$$\text{DCG} = 1 + 1/\log_2(2) + 1/\log_2(5) = 2.43$$

$$\text{IDCG} = 1 + 1/\log_2(2) + 1/\log_2(3) = 2.63 \text{ (Ideal)}$$

$$\text{NDCG} = \text{DCG} / \text{IDCG} = 2.43 / 2.63 = \mathbf{0.92}$$

Si on a des notes, remplacer "1" par notes.

# EVALUATION DE LISTE - EN PRATIQUE

2 solutions, ou plutôt 2 étapes:

- Laisser de côté (**train/test**) certains contenus aimés de l'utilisateur pour créer votre modèle → **Exemple au tableau!!**

Pour évaluer: calculer recall/précision grâce à **test** sur les recommandations faites à partir de **train**

*(tout à fait jouable pour votre projet)*

- Votre moteur est en ligne. Attendre le feedback de l'utilisateur sur vos recos pour évaluer (et enrichir!) votre modèle.  
*(certainement impossible pour votre projet)*

# CONCLUSION

En fonction du problème et surtout du type de données, on choisit un algo approprié.

- **Croyances...** Est-il pertinent de proposer des contenus en fonction de ce qu'ont aimé les autres? Oui → collaborative filtering 👍. Sinon, autre chose. Dépend aussi du **type** de contenu.
- **Algorithmes fructueux** : Amazon 2009: 30% du CA obtenu via recommandation! [source:McKinsey] 2015: 35%! Netflix 2017: 70% watch
- **Algorithmes customisables** : le bon sens et les idées "non-mathématiques" en général vont jouer un rôle important.

# CLUSTERING VS RECOMMENDATION

Attention : ce ne sont pas les mêmes choses ! Il ne faut pas les confondre.

Ces deux types d'algorithmes n'ont en commun que l'utilisation de **distances** entre objets. (Et parfois le clustering est utilisé dans le cadre de la recommandation):

- **Clustering** : algo **non supervisé** dont le but est de grouper des objets dans des classes.
- **Recommandation** : algo qui utilise des info sur les produits, les utilisateurs et/ou les contenu aimés par les utilisateurs pour leur proposer d'autres choix.