Mise en œuvre d'un algorithme de recommandations
Révisions sur les bases de données

*L'apprentissage automatique (*machine learning*) est le domaine de l'intelligence artificielle qui concerne la conception, l'analyse, le développement et l'implémentation de méthodes permettant à une machine (au sens large) d'évoluer afin de remplir des tâches complexes : classification de données, aide à la décision, détection d'anomalies, reconnaissance de formes, etc. Les applications vont de la météo aux analyses financières en passant par les jeux vidéos, la médecine et l'industrie automobile (liste non exhaustive).*

Les systèmes de recommandations sont omniprésents, en particulier sur internet et les plateformes de vidéos. On peut par exemple penser à l'algorithme de suggestion de vidéos de YouTube. Mangaki **`https://mangaki.fr`** *est un site de recommandation de mangas et d'animes créé par une association de passionnés. Le principe : l'utilisateur attribue des notes à des films, animes ou mangas et le système lui en recommande d'autres.*

Le but de ce problème est d'étudier l'algorithme qui permet d'effectuer ces recommandations (partie 2). Avant cela, on commence par étudier la base de données qui enregistre les notations des utilisateurs (partie 1).

Ce problème est basé sur l'article
Jill-Jênn Vie – Comment coder un système de recommandations en Python : l'exemple de Mangaki. – GNU/Linux magazine hors série 94

I. Base de données

Cette première partie est constituée de rappels sur les bases de données en général ainsi que sur l'écriture de requêtes SQL.

1 Vocabulaire

◇ Une *base de données relationnelle* (BDD) :

- Permet d'enregistrer des données, mais aussi des *relations* entre ces données ;
- Est constituée d'une ou plusieurs *tables* (ou *relations*) présentées sous forme de tableaux à 2 dimensions.

◇ Une *table* (on dit aussi *relation*) :
- Est identifiée par son *nom* ;
- Possède un certain nombre d'attributs (ou *champs*), chacun identifié par son *nom* et son *domaine* (ou *type*). Chaque attribut est représenté par une colonne ;
- Est constituée d'un certain nombre d'*enregistrements*, décrivant les valeurs de chaque attribut. Ces enregistrements sont disposés en ligne.

Le *schéma* d'une table est une liste constituée de son nom puis de ses attributs chacun étant associé à son domaine.

◇ Dans une table, une *clé primaire* est un champ qui permet d'identifier de manière unique chaque ligne de cette table. Une *clé étrangère* est un champ d'une table qui référence un champ d'une autre table (en général, une clé primaire de cette table).

◇ Un *système de gestion de bases de données* (SGBD) permet :
- De créer, modifier, supprimer des tables ;
- D'ajouter, supprimer, modifier des enregistrements dans des tables ;
- D'effectuer des recherches dans la base suivant des critères et présenter les réponses obtenues.

Il rend indépendant la structure générale de la base de données de la manière dont elle est gérée physiquement. Il permet également de gérer des accès concurrents, les droits d'accès, éviter les pertes d'information.

2 La base de données **`mangaki`**

Cette base de données est constituée de trois tables dont on donne ici les noms et les schémas (les clés primaires sont soulignées).
- Table **`Notes`** : (**`IdU`** integer, **`IdO`** integer, **`Note`** integer).
- Table **`Titres`** : (**`IdO`** integer, **`Titre`** text, **`Type`** text).
- Table **`Tests`** : (**`IdU`** integer, **`IdO`** integer, **`Note`** integer).

La table **`Notes`** *enregistre les notes données à certaines œuvres par les utilisateurs. Les utilisateurs sont désignés par leur identifiant* **`IdU`**, *les œuvres sont également désignées par leur identifiant* **`IdO`** *et la note attribuée est un nombre entier allant de* 1 *à* 4 *(*4 *signifie que l'utilisateur adore cette œuvre,* 3 *qu'il l'aime bien,* 2 *qu'il est neutre et* 1 *qu'il ne l'aime pas). La table* **`Tests`** *enregistre également des notes sur le même principe, elle ne sera utilisée que dans la deuxième partie du problème (pour tester l'algorithme). La table* **`Titres`** *donne le titre et le type (manga, anime ou album) de chaque œuvre, désignée par son identifiant* **`IdO`**.
Pour fixer les idées, on donne ci-dessous des extraits *de ces trois tables.*

Notes (extrait)

IdU	IdO	Note
14	99	4
152	9	2
152	317	3
231	142	3
373	66	3
558	137	1

Titres (extrait)

IdO	Titre	Type
2	Dragon Ball	anime
14	Maou lover VS Le prince	manga
30	Armitage III Polymatrix	anime
34	Slayers Knight of Aqua Lord	manga
980	Detenu 042	manga
4678	UTAMONOGATARI?	album

Tests (extrait)

IdU	IdO	Note
66	250	1
387	222	3
356	258	3
513	175	2
86	22	3
802	140	3

3 Interroger une BDD

◇ Le langage *SQL* (*structured query language*) est un standard permettant d'interroger une base de données (il permet également de réaliser les opérations de création et modification de la base de données).

◇ Les commandes SQL les plus simples sont de la forme :

`SELECT` *nom des champs*
`FROM` *nom de la table*
`WHERE` *condition*

Question 1. Écrire en langage SQL les requêtes pour obtenir :
(a) Les identifiants des œuvres notées par l'utilisateur 231 ainsi que les notes attribuées.
(b) Les identifiants des utilisateurs ayant donné des notes supérieures ou égales à 3.

◇ Pour « croiser » les informations contenues dans deux tables, on utilise une requête du type :

`SELECT` *nom des champs*
`FROM` *nom de la première table*
`JOIN` *nom de la deuxième table*
`ON` *condition de jointure*

C'est ce que l'on appelle une *jointure* (si besoin, on peut combiner avec **`WHERE`**).

Question 2. Écrire les requêtes SQL permettant de :
(a) Déterminer les titres des œuvres notées par l'utilisateur 231.
(b) Déterminer les titres des mangas ayant obtenu une note égale à 4.
(c) Déterminer qui n'aime pas *Dragon Ball*.

◇ On peut également compter le nombre d'éléments d'une colonne (**`COUNT`**), calculer la somme et la moyenne (**`SUM`**, **`AVG`**), l'écart-type et la variance (**`STDDEV`**, **`VARIANCE`**), le minimum et le maximum (**`MIN`**, **`MAX`**). Ces fonctions sont appelées fonction d'*agrégation* de résultats.

Question 3. Écrire les requêtes SQL permettant de :
(a) Déterminer la moyenne des notes attribuées par l'utilisateur 231.
(b) Déterminer le plus grand numéro d'utilisateur apparaissant dans la table **`Notes`**.

Question 4. Écrire les requêtes SQL permettant de :
(a) Déterminer le plus grand identifiant apparaissant dans la table **`Titres`**.
(b) Déterminer le plus grand identifiant apparaissant dans la table **`Titres`** ainsi que le titre de l'œuvre associées.
(c) Déterminer le titre (uniquement) de l'œuvre dont l'identifiant est le plus grand dans la table **`Titres`**.

Remarque : ce type de question se retrouve souvent dans les sujets de concours.

Question 5. Pour aller plus loin.

(a) Pour limiter l'application d'une fonction d'agrégation à certaines valeurs d'un champ, on utiliser **`GROUP BY champ`**. Application : déterminer la moyenne des notes données par chaque utilisateur.

(b) On peut compléter une recherche dans une table par l'expression **`ORDER BY champ`** ou **`ORDER BY champ DESC`** pour que les résultats soient présentés triés suivant le champ précisé (dans l'ordre croissant ou décroissant). Application : reprendre la requête précédente et présenter les résultats par ordre décroissant des identifiants d'utilisateur. Reprendre cette requête et présenter les résultat par moyenne croissante.

(c) Dans une expression **`FROM table`** ou **`JOIN table`**, on peut compléter par **`AS t`** pour signaler que la table sera aussi désignée par le nom **`t`**. Application : déterminer s'il existe des œuvres auquelles le même utilisateur a attribué des notes différentes.

Question 6. Comment déterminer (à l'aide d'une ou plusieurs requêtes SQL) si toute œuvre a reçu au moins une note ?

Question 7. Comment déterminer (à l'aide d'une ou plusieurs requêtes SQL) s'il existe un utilisateur qui a attribué la note 1 à toutes les œuvres qu'il a notées ?

II. Travaux pratiques

On veut utiliser conjointement PYTHON *et la base de données* ***`mangaki.sqlite`*** *afin de mettre en place l'algorithme de recommandations.*

Question 8. Récupérer les deux fichiers

`mangaki.sqlite` et **`pb_bdd-2019-2020.py`**

et les placer dans le répertoire où vous enregistrez vos programmes PYTHON.

Question 9. Vérifier qu'en lançant le programme fourni vous obtenez l'affichage suivant :

```
[(844, 250, 3), (274, 189, 3), (543, 250, 1)]
```

Regarder les requêtes SQL apparaissant au début du programme et en déduire :

- Le contenu de la liste **`Notes`** (l'affichage ci-dessus montre le début de cette liste) ;
- Le contenu des variables **`NbU`** et **`NbO`**.

1 Algorithme de prédiction des notes

Considérons un utilisateur repéré par son identifiant i avec $0 \leqslant i \leqslant$ ***`NbU`*** -1 *ainsi qu'une œuvre repérée par son identifiant j avec $0 \leqslant j \leqslant$* ***`NbO`*** -1. *Le principe de l'algorithme de recommandation est d'associer à l'utilisateur i un vecteur U_i et d'associer à l'œuvre j un vecteur V_j choisis de sorte que le produit scalaire $\langle U_i, V_j \rangle$ représente (approximativement) la note attribuée par l'utilisateur i à l'œuvre j. Initialement, les vecteurs $U_0, \ldots, U_{NbU-1}$ ainsi que $V_0, \ldots, V_{NbO-1}$ sont choisis aléatoirement puis ajustés à l'aide des données contenues dans*

la liste **`Notes`**. *Plus précisément, pour chaque triplet* (i, j, r) *apparaissant dans* **`Notes`**, *on commence par calculer :*

$$e = \langle U_i, V_j \rangle - r$$

(c'est l'erreur commise par le système). On ajuste alors U_i *et* V_j *en fonction de cette erreur en posant :*

$$U_i \leftarrow U_i - \alpha(eV_j + \beta U_i)$$
$$V_j \leftarrow V_j - \alpha(eU_i + \beta V_j)$$

où α *et* β *sont des paramètres de l'algorithme. Un autre paramètre de cet algorithme est le nombre de composantes dans les vecteurs* U_i *et* V_j. *On la notera* **`NbC`**.

Question 10. Écrire une fonction **`initialiser(NbU,NbO,NbC)`** qui construit et renvoie le couple **`(U,V)`** constitué des deux listes de taille **`NbU`** et **`NbO`** et dont les éléments sont des vecteurs de taille **`NbC`** choisis aléatoirement en appelant **`np.random.random(NbC)`**.

Question 11. Écrire une fonction **`ajuster(U,V,Notes,alpha,beta)`** qui parcourt la liste **`Notes`** et réalise les ajustements décrits plus haut. Cette fonction ne renvoie pas de résultat mais modifie directement les listes **`U`** et **`V`**. On pourra remarquer que si **`u`** et **`v`** sont deux vecteurs NUMPY de même taille, leur produit scalaire (canonique) est **`np.sum(u*v)`**.

Question 12. Utiliser les fonctions précédentes pour construire les listes **`U`** et **`V`** à partir des listes initiales aléatoires en appliquant 10 fois la fonction ajuster. On prendra **`NbC=10`**, **`alpha=0.01`** et **`beta=0.1`**.

2 Prédiction des notes

Question 13. Pour un utilisateur i et une œuvre j, la note proposée par l'algorithme de recommandation est $\langle U_i, V_j \rangle$, arrondi à l'entier le plus proche (on utilisera **`np.round`**). De plus, si le résultat obtenu est inférieur à 1 (respectivement supérieur à 4), on le remonte à 1 (respectivement on le redescend à 4). Écrire une fonction **`prediction_note(i,j,U,V)`** qui calcule et renvoie cette note.

3 Test de l'algorithme

Le principe est de regarder les notes prédites par l'algorithme et de les comparer aux notes données par les utilisateurs et qui sont enregistrées dans la table **`Tests`**.

Question 14. En s'inspirant de la commande qui a construit la liste **`Notes`**, construire de même la liste **`Tests`**.

Question 15. Déterminer le nombre d'erreurs effectuées par l'algorithme de recommandation sur la liste **`Tests`**. On distinguera trois situations :

- Lorsque la note prédite par l'algorithme est égale à celle attribuée par l'utilisateur, il s'agit d'un *succès* pour l'algorithme ;
- Lorsque la note attribuée par l'utilisateur et la note prédite par l'algorithme diffèrent (en valeur absolue) de 1, on dit qu'il s'agit d'une *erreur acceptable* de l'algorithme ;
- Dans les autres cas, on dit qu'il s'agit d'un *échec* de l'algorithme.

Comptabiliser les succès, les erreurs acceptables et les échecs et commenter le résultat obtenu.

Question 16. On peut donner une autre représentation des erreurs et succès obtenus par l'algorithme en utilisant la *matrice de confusion*. C'est la matrice M de taille 4×4 telle que $M[r, r']$ représente le nombre de fois où l'algorithme a prédit la note r' alors que la note attribuée était r. Écrire le code permettant d'obtenir la matrice de confusion (attention, en PYTHON la numérotation des lignes et colonnes d'une matrice commence à 0) et commenter le résultat obtenu.

Quelques explications sur l'algorithme utilisé

◇ On a décidé d'associer aux utilisateurs et aux œuvres des vecteurs U_i et V_j de sorte que la note attribuée par l'utilisateur i à l'œuvre j soit approximativement $\langle U_i, V_j \rangle$. Ainsi, si l'utilisateur i a aimé l'œuvre j, alors $\langle U_i, V_j \rangle$ sera assez élevé. Par ailleurs, si l'utilisateur i' a des goûts assez proches de i, on espère que ceci se traduira par des vecteurs U_i et $U_{i'}$ assez proches. Par conséquent, si i' n'a pas noté l'œuvre j, le système pourra lui proposer en recommandation puisque $\langle U_{i'}, V_j \rangle$ sera assez élevé (car proche de $\langle U_i, V_j \rangle$).

◇ Ce modèle constitué des vecteurs U_i et V_j reflète bien chaque information (i, j, r) contenue dans la table **`Notes`** lorsque les quantités :

$$\frac{1}{2}\big(\langle U_i, V_j \rangle - r\big)^2$$

sont petites (le facteur 1/2 sert pour simplifier la suite des calculs). Pour simplifier, imaginons que U_i et V_j ne sont pas des vecteurs mais simplement des nombres réels u_i et v_j. On considère alors :

$$f(u_i, v_j) = \frac{1}{2}(u_i v_j - r)^2$$

La dérivée de cette quantité par rapport à u_i est :

$$\frac{\partial f}{\partial u_i} = v_j(u_i v_j - r) = v_j e \quad \text{en posant} \quad e = u_i v_j - r$$

L'idée est d'ajuster u_i en le remplaçant par :

$$u_i - \alpha \frac{\partial f}{\partial u_i}$$

où α est un nombre strictement positif. En effet, deux cas sont possibles :

— Si la dérivée est négative, alors f doit être décroissante par rapport à u_i et l'opération

$$u_i - \alpha \frac{\partial f}{\partial u_i}$$

revient à augmenter un peu u_i, ce qui doit faire diminuer f.

— Si la dérivée est positive, alors f doit être croissante par rapport à u_i et l'opération

$$u_i - \alpha \frac{\partial f}{\partial u_i}$$

revient à diminuer un peu u_i, ce qui doit faire aussi diminuer f.

C'est pourquoi on voit apparaitre dans l'algorithme l'expression :

$$u_i = u_i - \alpha v_j e$$

et de manière similaire :

$$v_j = v_j - \alpha u_i e$$

⋄ Les termes supplémentaires en β qui apparaissent dans l'algorithme sont plus délicats à justifier. Il sont là pour éviter le phénomène de *sur-apprentissage* : ce phénomène se produit lorsque le modèle est tellement adapté aux données qui ont été utilisées pour l'apprentissage qu'il en devient un modèle spécialisé pour ce jeu de données et répond alors mal lorsqu'il est interrogé sur d'autres données.

Corrections

Q 1.
```
SELECT IdO,Note FROM Notes WHERE IdU=231
```

```
SELECT IdU FROM Notes WHERE Note>=3
```

Q 2.
```
SELECT Titre FROM Titres
JOIN Notes On Notes.IdO=Titres.IdO
WHERE IdU=231
```

```
SELECT Titre FROM Titres
JOIN Notes On Notes.IdO=Titres.IdO
WHERE Note=4
```

```
SELECT IdU FROM Notes
JOIN Titres ON Titres.IdO=Notes.IdO
WHERE Titre="Dragon Ball" AND Note=1
```

Q 3.
```
SELECT AVG(Note) FROM Notes WHERE IdU=231
```

```
SELECT MAX(IdU) FROM Notes
```

Q 4.
```
SELECT MAX(IdO) FROM Titres
```

```
SELECT MAX(IdO),Titre FROM Titres
```

```
SELECT Titre FROM Titres
WHERE IdO=(SELECT MAX(IdO) FROM Titres)
```

Q 5. (a)

```
SELECT IdU, AVG(Note) FROM Notes GROUP BY IdU
```

(b) Première requête :

```
SELECT IdU, AVG(Note) FROM Notes GROUP BY IdU ORDER BY IdU DESC
```

Deuxième requête :

```
SELECT IdU, AVG(Note) FROM Notes GROUP BY IdU ORDER BY AVG(Note)
```

Ou mieux :

```
SELECT IdU, AVG(Note) AS N FROM Notes GROUP BY IdU ORDER BY N
```

(c)

```
SELECT N1.IdO FROM Notes AS N1
JOIN Notes AS N2 ON N1.IdU=N2.IdU AND N1.IdO=N2.IdO
WHERE N1.Note<>N2.Note
```

Q 6. Une possibilité :

```
SELECT IdO FROM Titres
EXCEPT
SELECT IdO FROM Notes
```

Une autre possibilité, comparer les résultats des requêtes :

```
SELECT COUNT(*) FROM Titres
```

et :

```
SELECT COUNT(DISTINCT IdO) FROM Notes
```

Q 7. Une possibilité :

```
SELECT IdU FROM Notes
EXCEPT
SELECT IdU FROM Notes WHERE note>=2
```

Q 10.

```
import sqlite3
import numpy as np

db = sqlite3.connect('/home/ba/Enseignement/PC/Python/Data/mangaki.sqlite')

Notes = db.execute("SELECT * FROM Notes").fetchall()
Tests = db.execute("SELECT * FROM Tests").fetchall()
NbU = db.execute("SELECT MAX(IdU)+1 FROM Notes").fetchone()[0]
NbO = db.execute("SELECT MAX(IdO)+1 FROM Notes").fetchone()[0]

db.close()

def initialiser(NbU,NbO,NbC):
    U = [np.random.random(NbC) for i in range(0,NbU)]
    V = [np.random.random(NbC) for j in range(0,NbO)]
    return (U,V)
```

Q 11.

```
def ajuster(U,V,Notes,alpha,beta):
    for (i,j,r) in Notes:
        e = np.sum(U[i]*V[j])-r
        U[i] = U[i]-alpha*(e*V[j]+beta*U[i])
        V[j] = V[j]-alpha*(e*U[i]+beta*V[j])
```

Q 12.

```
NbC = 10
alpha = 0.01
beta = 0.1

(U,V) = initialiser(NbU,NbO,NbC)

for k in range(0,10):
    ajuster(U,V,Notes,alpha,beta)
```

Q 13.

```
def prediction_note(i,j,U,V):
    r = np.round(np.sum(U[i]*V[j]))
    r = max(r,1)
    r = min(r,4)
    return r

print(prediction_note(231,142,U,V))
```

```
3.0
```

Q 15.

```
total = len(Tests)
succes = 0
echecs = 0
for (i,j,r) in Tests:
    r1 = prediction_note(i,j,U,V)
    if r==r1:
        succes = succes+1
    elif abs(r-r1)>1:
        echecs = echecs+1

print(total,succes,echecs)
```

```
3882 1858 291
```

```
print(f"{echecs/total*100} pourcent d'échecs")
```

```
7.496136012364761 pourcent d'échecs
```

Q 16.

```
M = np.zeros((4,4),dtype=int)
for (i,j,r) in Tests:
    r1 = int(prediction_note(i,j,U,V))
    M[r-1,r1-1] += 1

print(M)
```

```
[[ 37 262 160 6]
[ 22 447 362 3]
[ 29 894 1367 33]
[ 1 92 160 7]]
```