Nom et prénom : QARA EL HOUCINE
Filière : TNPEIC S3

Système de Recommandation

1. Introduction

Ce projet implémente un système de recommandation permettant de suggérer des films aux utilisateurs en fonction de leurs préférences passées. Nous comparons deux approches : **Filtrage Collaboratif** et **Filtrage Basé sur le Contenu**.

1.1. Chargement du Dataset

Le dataset contient deux fichiers principaux :

- ratings.csv

```
UserID,ItemID,Rating
1,1,5
1,2,3
1,3,5
2,1,4
2,3,5
```

- movies.csv

```
ItemID,Title,Genre
1,Film 1 - Romance,Romance
2,Film 2 - Comedy,Comedy
3,Film 3 - Drama,Drama
```

Le fichier pretraitement_donnees.py pour le traitement de dataset :

- **Fonction charger_donnees()**

Cette fonction charge les données depuis un fichier CSV en utilisant pandas. La fonction pd.read_csv() lit le fichier et retourne un DataFrame contenant toutes les colonnes.

```python
def charger_donnees(chemin_fichier):
    return pd.read_csv(chemin_fichier)
```

- **Fonction nettoyer_donnees()**

Cette fonction nettoie les données en trois étapes :

- **dropna()** : Supprime toutes les lignes contenant des valeurs manquantes (NaN)
- **pd.Categorical().codes + 1** : Transforme les IDs désordonnés en IDs séquentiels commençant à 1
- **drop_duplicates()** : Supprime les doublons sur les colonnes UserID et ItemID, gardant la première occurrence

```python
def nettoyer_donnees(donnees):
    donnees = donnees.dropna()
    donnees['UserID'] = pd.Categorical(donnees['UserID']).codes + 1
    donnees['ItemID'] = pd.Categorical(donnees['ItemID']).codes + 1
    donnees = donnees.drop_duplicates(subset=['UserID', 'ItemID'], keep='first')
    return donnees
```

2. Méthodes de Recommandation

2.1. Filtrage Collaboratif (Fichier filtrage_collaboratif.py)

Principe : Les utilisateurs similaires ont des goûts similaires.

- **Méthode entrainer()**

Cette méthode crée une matrice utilisateur-item où chaque ligne représente un utilisateur, chaque colonne représente un item, et chaque cellule contient la note attribuée (ou 0 si non noté). La fonction **pivot_table()** transforme les données longues en format matriciel. Les listes d'utilisateurs et d'items sont extraites pour référence future.

```python
def entrainer(self, donnees_ratings):
    self.matrice = donnees_ratings.pivot_table(
        index='UserID',
        columns='ItemID',
        values='Rating',
        fill_value=0
    )
    self.utilisateurs = list(self.matrice.index)
    self.items = list(self.matrice.columns)
```

Résultat : Matrice Utilisateur-Item

User 1	Film 1	FIlm 2	Film 3
User 2	5	0	4
User 3	4	3	0
User 4	0	5	3

Chaque ligne = un utilisateur, chaque colonne = un film, chaque case = la note (0 si pas noté)

- **Méthode calculer_similarite()**

Cette méthode calcule la similarité cosinus entre deux utilisateurs. Elle récupère les vecteurs de notes des deux utilisateurs, calcule leur produit scalaire avec **np.dot()**, puis calcule les normes des vecteurs avec **np.sqrt(np.sum()).** La similarité est le produit scalaire divisé par le produit des normes. Si une norme est nulle, la similarité est 0.

$$\text{similarité} = (\text{vecteur1} \cdot \text{vecteur2}) \,/\, (||\text{vecteur1}|| \times ||\text{vecteur2}||)$$

```
def calculer_similarite(self, user1, user2):
    notes1 = self.matrice.loc[user1].values
    notes2 = self.matrice.loc[user2].values

    produit = np.dot(notes1, notes2)
    norme1 = np.sqrt(np.sum(notes1**2))
    norme2 = np.sqrt(np.sum(notes2**2))

    if norme1 == 0 or norme2 == 0:
        return 0

    return produit / (norme1 * norme2)
```

- **Méthode predire()**

Cette méthode prédit la note qu'un utilisateur donnerait à un item. Elle trouve tous les utilisateurs qui ont noté cet item, calcule leur similarité avec l'utilisateur cible, puis effectue une moyenne pondérée de leurs notes. Les utilisateurs plus similaires ont un poids plus important. Si aucun utilisateur similaire n'a noté l'item, elle retourne la moyenne des notes de l'item. Le résultat est limité entre 1 et 5.

```
def predire(self, id_utilisateur, id_item):
    if id_utilisateur not in self.utilisateurs:
        return 3.0

    if id_item not in self.items:
        return 3.0

    notes_item = self.matrice[id_item]
    utilisateurs_ayant_note = notes_item[notes_item > 0].index.tolist()

    if len(utilisateurs_ayant_note) == 0:
        return 3.0

    somme_sim = 0
    somme_notes = 0

    for autre_user in utilisateurs_ayant_note:
        if autre_user != id_utilisateur:
            similarite = self.calculer_similarite(id_utilisateur, autre_user)
            note = self.matrice.loc[autre_user, id_item]
            somme_sim += abs(similarite)
            somme_notes += similarite * note

    if somme_sim == 0:
        return notes_item.mean()

    prediction = somme_notes / somme_sim
    return max(1.0, min(5.0, prediction))
```

- **Méthode recommander()**

Cette méthode génère des recommandations pour un utilisateur. Elle prédit la note pour tous les items non notés par l'utilisateur, trie les résultats par score décroissant, et retourne les N meilleurs items. Si l'utilisateur n'existe pas, elle retourne les items les plus populaires.

```
def recommander(self, id_utilisateur, nombre_recommandations=10):
    if id_utilisateur not in self.utilisateurs:
        popularite = self.matrice.sum().sort_values(ascending=False)
        return [(item, score) for item, score in popularite.head(nombre_recommandations).items()]

    notes_utilisateur = self.matrice.loc[id_utilisateur]
    items_non_notes = notes_utilisateur[notes_utilisateur == 0].index.tolist()

    predictions = []
    for item in items_non_notes:
        score = self.predire(id_utilisateur, item)
        predictions.append((item, score))

    predictions.sort(key=lambda x: x[1], reverse=True)
    return predictions[:nombre_recommandations]
```

2.1. Filtrage Basé sur le Contenu

Principe : Recommander des films similaires à ceux que l'utilisateur a aimés, basé sur les genres.

- **Méthode entrainer()**

Cette méthode crée d'abord la matrice utilisateur-item comme le Filtrage Collaboratif. Ensuite, si les métadonnées des items (genres) sont disponibles, elle applique TF-IDF pour vectoriser les genres. TF-IDF donne plus de poids aux genres rares qu'aux genres communs. Enfin, elle calcule la similarité cosinus entre tous les items pour créer une matrice de similarité pré-calculée.

```
def entrainer(self, donnees_ratings, donnees_items=None):
    self.matrice_utilisateur_item = donnees_ratings.pivot_table(
        index='UserID',
        columns='ItemID',
        values='Rating',
        fill_value=0
    )
    self.utilisateurs = list(self.matrice_utilisateur_item.index)
    self.items = list(self.matrice_utilisateur_item.columns)

    if donnees_items is not None:
        self.genres_items = {}
        for _, row in donnees_items.iterrows():
            item_id = row['ItemID']
            genre = str(row.get('Genre', ''))
            if item_id in self.items:
                self.genres_items[item_id] = genre

        genres_liste = [self.genres_items.get(item, '') for item in self.items]

        vectorizer = TfidfVectorizer()
        self.matrice_tfidf = vectorizer.fit_transform(genres_liste)

        self.similarite_items = cosine_similarity(self.matrice_tfidf)
    else:
        self.similarite_items = cosine_similarity(self.matrice_utilisateur_item.values.T)
```

- **Méthode calculer_similarite_items()**

Cette méthode récupère la similarité entre deux items depuis la matrice de similarité pré-calculée. Elle trouve les indices des items dans la liste, puis retourne la valeur correspondante dans la matrice.

```
def calculer_similarite_items(self, item1, item2):
    if item1 not in self.items or item2 not in self.items:
        return 0.0

    idx1 = self.items.index(item1)
    idx2 = self.items.index(item2)

    return self.similarite_items[idx1][idx2]
```

- **Méthode predire()**

Cette méthode prédit la note qu'un utilisateur donnerait à un item basé sur le contenu. Elle trouve les items que l'utilisateur a notés, calcule leur similarité avec l'item cible, puis effectue une moyenne pondérée des notes données par l'utilisateur à ces items similaires. Contrairement au Filtrage Collaboratif qui compare des utilisateurs, cette méthode compare des items.

```
def predire(self, id_utilisateur, id_item):
    if id_utilisateur not in self.utilisateurs:
        return 3.0

    if id_item not in self.items:
        return 3.0

    notes_utilisateur = self.matrice_utilisateur_item.loc[id_utilisateur]
    items_notes = notes_utilisateur[notes_utilisateur > 0].index.tolist()

    if len(items_notes) == 0:
        return 3.0

    somme_sim = 0
    somme_notes = 0

    for item_note in items_notes:
        similarite = self.calculer_similarite_items(id_item, item_note)
        note = self.matrice_utilisateur_item.loc[id_utilisateur, item_note]
        somme_sim += abs(similarite)
        somme_notes += similarite * note

    if somme_sim == 0:
        return notes_utilisateur.mean()

    prediction = somme_notes / somme_sim
    return max(1.0, min(5.0, prediction))
```

3. Évaluation des Modèles(evaluation.py)

3.1. Fonction erreur_quadratique_moyenne_racine() – RMSE

Cette fonction calcule le RMSE (Root Mean Squared Error). Elle calcule d'abord les erreurs au carré pour chaque prédiction, puis prend la moyenne et la racine carrée. Le RMSE pénalise plus les grandes erreurs que les petites erreurs.

$$\text{RMSE} = \sqrt{\Sigma(\text{prediction} - \text{réel})^2 / n}$$

```
def erreur_quadratique_moyenne_racine(valeurs_reelles, valeurs_predites):
    erreurs = [(r - p)**2 for r, p in zip(valeurs_reelles, valeurs_predites)]
    return np.sqrt(np.mean(erreurs))
```

3.2. Fonction erreur_absolue_moyenne() – MAE

Cette fonction calcule le MAE (Mean Absolute Error). Elle calcule la valeur absolue de chaque erreur, puis prend la moyenne. Contrairement au RMSE, le MAE traite toutes les erreurs de manière égale.

```
def erreur_absolue_moyenne(valeurs_reelles, valeurs_predites):
    erreurs = [abs(r - p) for r, p in zip(valeurs_reelles, valeurs_predites)]
    return np.mean(erreurs)
```

3.3. Fonction precision_a_k()

Cette fonction calcule la Précision@k, qui mesure la proportion d'items pertinents dans les k premières recommandations. Un item est considéré pertinent si son rating est supérieur ou égal au seuil (par défaut 3). La fonction prend les k premiers items recommandés et compte combien sont pertinents.

```
def precision_a_k(items_pertinents, items_recommandes, k=10):
    if len(items_recommandes) == 0:
        return 0.0

    items_pertinents_set = set(items_pertinents)
    items_rec_k = items_recommandes[:k]

    nb_pertinents = sum(1 for item in items_rec_k if item in items_pertinents_set)

    return nb_pertinents / len(items_rec_k)
```

3.4. Fonction rappel_a_k()

Cette fonction calcule le Recall@k, qui mesure la proportion d'items pertinents trouvés parmi tous les items pertinents. Elle compte combien d'items pertinents sont présents dans les k premières recommandations, puis divise par le nombre total d'items pertinents.

```
def rappel_a_k(items_pertinents, items_recommandes, k=10):
    if len(items_pertinents) == 0:
        return 0.0

    items_pertinents_set = set(items_pertinents)
    items_rec_k = set(items_recommandes[:k])

    nb_trouves = len(items_pertinents_set & items_rec_k)

    return nb_trouves / len(items_pertinents_set)
```

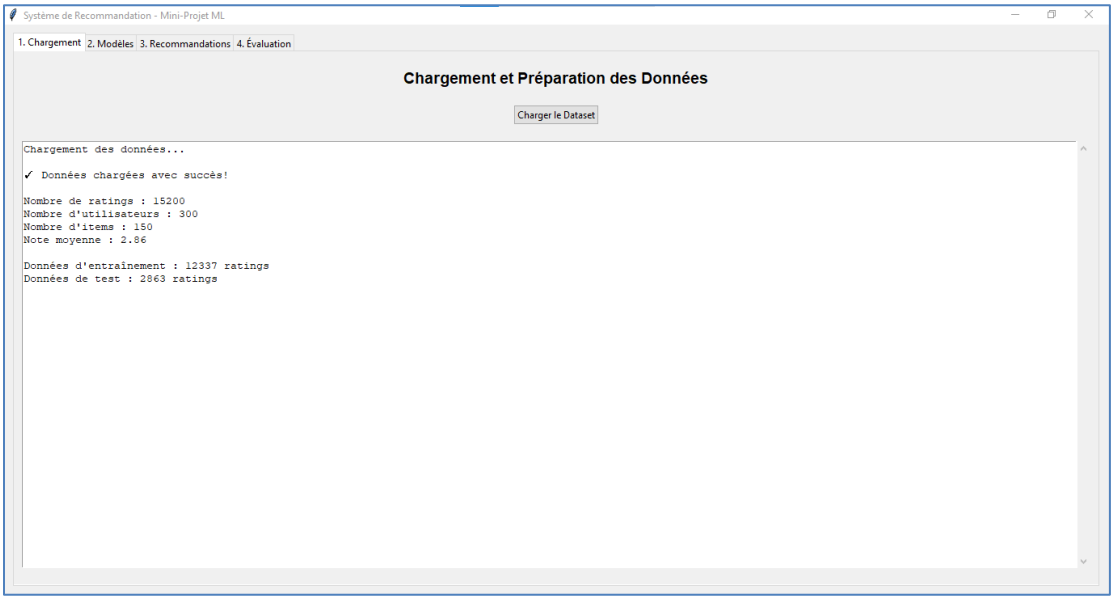
4. Interface Graphique (Tkinter)

L'interface graphique du système de recommandation a été développée avec Tkinter pour offrir une interaction intuitive avec le système. Elle est organisée en quatre onglets principaux : Chargement (pour charger et préparer le dataset), Modèles (pour entraîner les deux modèles de recommandation), Recommandations (pour obtenir des recommandations personnalisées pour un utilisateur spécifique), et Évaluation (pour mesurer les performances des modèles). Chaque onglet contient des boutons et des zones de texte pour afficher les résultats. L'interface utilise le widget **ttk.Notebook** pour créer le système d'onglets, et la classe InterfaceRecommandation encapsule toutes les fonctionnalités en maintenant l'état de l'application (données chargées, modèles entraînés, résultats).

Pour éviter que l'interface se bloque pendant les calculs longs, notamment lors de l'évaluation des modèles, le système utilise le threading. L'évaluation est exécutée dans un thread séparé, garantissant que l'interface reste réactive. L'interface intègre également une gestion complète des erreurs avec validation des données et des entrées utilisateur, ainsi que des messages informatifs via des boîtes de dialogue. Cette interface graphique rend le système accessible aux utilisateurs non techniques et facilite l'interaction avec les modèles complexes sans avoir à manipuler directement le code Python.

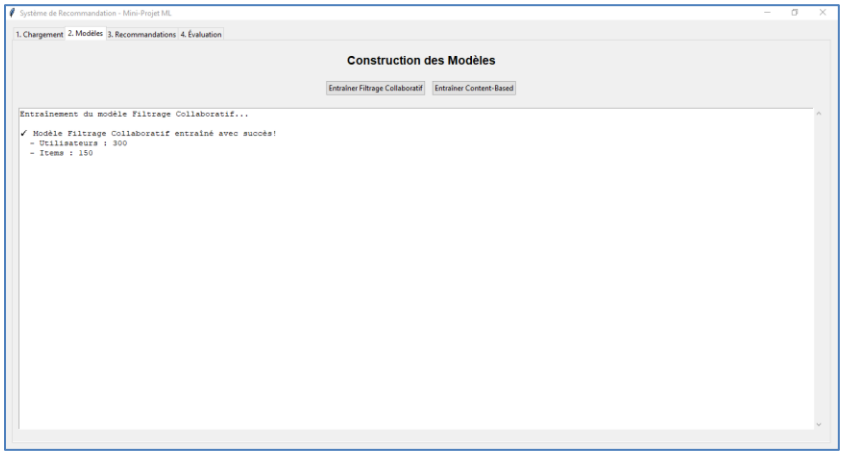
Onglet 1 : Chargement

Cet onglet permet de charger le dataset depuis les fichiers CSV avec un simple clic sur le bouton "Charger le Dataset". Le système vérifie l'existence des fichiers, charge et nettoie les données, divise automatiquement les données en ensembles d'entraînement (80%) et de test (20%), puis affiche les statistiques du dataset (nombre de ratings, utilisateurs, items, note moyenne) dans une zone de texte scrollable. Les erreurs sont gérées avec des boîtes de dialogue pour informer l'utilisateur en cas de problème.



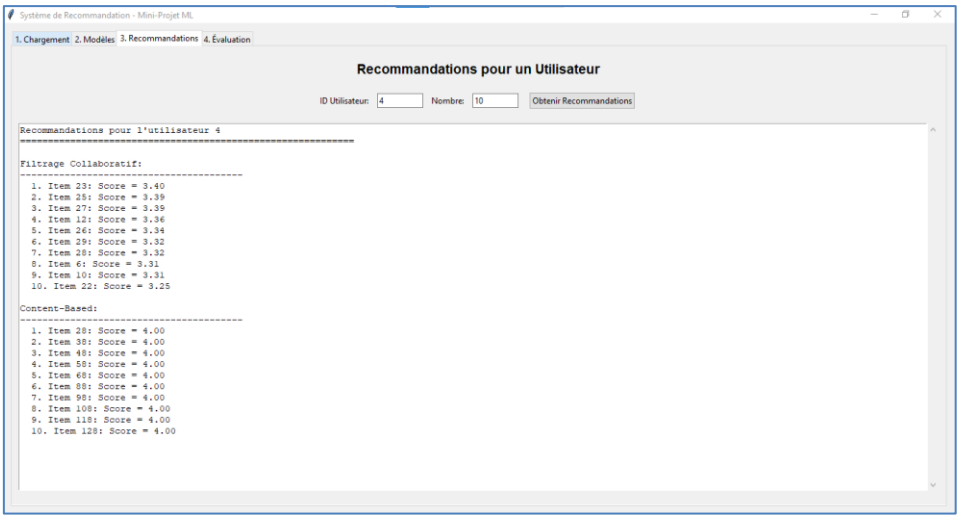
Onglet 2 : Modèles

Cet onglet permet d'entraîner les deux modèles de recommandation séparément grâce à deux boutons indépendants : "Entraîner Filtrage Collaboratif" et "Entraîner Content-Based". Chaque bouton vérifie d'abord que les données ont été chargées, puis procède à l'entraînement du modèle correspondant. Une fois l'entraînement terminé, l'interface affiche un message de confirmation ainsi que le nombre d'utilisateurs et d'items dans le modèle. Cette séparation permet à l'utilisateur d'entraîner uniquement les modèles dont il a besoin.



Onglet 3 : Recommandations

Cet onglet permet d'obtenir des recommandations personnalisées pour un utilisateur spécifique. L'utilisateur saisit l'ID de l'utilisateur et le nombre de recommandations souhaitées, puis clique sur "Obtenir Recommandations". Le système vérifie que les modèles sont entraînés, appelle la méthode **recommander()** pour chaque modèle, et affiche les résultats sous forme de liste numérotée avec les scores de prédiction, permettant une comparaison visuelle des recommandations des deux modèles côte à côte.



Onglet 4 : Évaluation

Cet onglet permet de mesurer les performances des modèles sur l'ensemble de test. Pour éviter que l'interface se bloque pendant les calculs longs, l'évaluation est exécutée dans un thread séparé, garantissant que l'interface reste réactive. Le système calcule les métriques RMSE (Root Mean Squared Error) et MAE (Mean Absolute Error) pour chaque modèle et affiche les résultats dans un tableau comparatif, permettant de visualiser facilement les différences de performance entre les deux modèles.