CONSIGNES GÉNÉRALES

DOCUMENTS NON AUTORISES
CE CAHIER D'EXAMEN COMPORTE 25 PAGES + 2 PAGES SUPPLÉMENTAIRES
LES RÉPONSES DOIVENT ÊTRE ÉCRITES DANS
LES ESPACES RÉPONSES DÉDIÉES
EN CAS DE BESOIN UTILISER LES PAGES VIDES EN FIN DU CAHIER, DANS
CE CAS, IL FAUT LE SIGNALER DANS L'ESPACE RÉPONSE CORRESPONDANT
L'USAGE DES CALCULATRICES EST INTERDIT
Il FAUT RESPECTER IMPÉRATIVEMENT LES NOTATIONS DE L'ÉNONCÉ
VOUS POUVEZ ÉVENTUELLEMENT UTILISER L'ANNEXE

Le sujet comporte trois parties qui traitent les aspects suivants :

Partie I : Programmation orientée objet (Q1..Q21).

Partie II : Base de données relationnelle (Q22..Q31).

Partie I : Programmation Orientée Objet

Les systèmes de recommandation sont des algorithmes d'intelligence artificielle utilisés pour suggérer des éléments pertinents aux internautes par rapport à leurs préférences. Ils sont largement utilisés dans de nombreux domaines tels que le commerce électronique, les services de streaming de musique et de vidéo, les applications de voyage, l'éducation et bien d'autres.

Parmi les systèmes de recommandation on s'intéresse au modèle **Item-Based Collaborative Filtering (IBCF)** qui permet de recommander des items à des users en se basant sur l'historique de leurs interactions avec des items similaires. IBCF se base sur des évaluations (scores) émises par les users pour certains items.

Un exemple de système de recommandation est celui de cours en ligne (items) pour des étudiants (users). Le modèle IBCF exploite des données qui représentent les préférences des étudiants pour certains cours sous forme de scores. Les données peuvent être organisées dans une matrice où les lignes représentent les étudiants, les colonnes représentent les cours et les éléments de la matrice représentent les évaluations.

score =

	Math	History	Programming	English	Chemistry	Art
User1	4	3	5	-	2	1
User2	-	2	4	3	-	5
User3	5	-	4	2	3	-
User4	3	4	-	5	1	2
User5	2	-	3	1	-	4
User6	1	5	2	4	5	3
User7	-	3	-	2	4	1
User8	4	-	1	5	2	3

FIGURE 1 – Exemple de données d'entrées pour un Système de recommandation de cours en ligne

Dans cet exemple :

- Chaque ligne i représente un étudiant user_i.
- Chaque colonne j représente un cours item_j.
- Un score_{ij} est donné par un étudiant i pour un cours j sur une échelle de 1 à 5., où `'-'` indique aucune évaluation.

Dans la suite de l'énoncé, nous utiliserons les données de cet exemple pour la validation du travail demandé.

Espace de réponse supplémentaire

Espace de réponse supplémentaire

Classes à Implémenter

Dans le but de réaliser un système de recommandation selon le modèle IBCF, on propose d'implémenter un ensemble de classes permettant de mettre en place les fonctionnalités et les données nécessaires :

- `DataFrame` : Une classe pour stocker les données tabulaires représentant les scores attribués par les users aux items et offrant des méthodes pour les représenter et les manipuler.
- `SimilarityMatrix` : Une classe dérivée de `DataFrame` pour représenter la matrice de similarités entre les items.
- `MaskedDataFrame` : Une classe dérivée de `DataFrame` pour stocker et manipuler les données tabulaires avec des valeurs masquées (Dans la figure 1 l'user_7 n'a pas interagit avec l'item_1, ainsi le score associé est une valeur masquée).
- `SimilarityMeasure` : Une classe abstraite qui encapsule les données et les opérations nécessaires pour appliquer une mesure de similarité sur les items (Similarité Cosinus, Similarité de Jaccard et Similarité Euclidienne). Elle est dédiée à la spécialisation par héritage.
- `CosineSimilarityMeasure` : Une classe dérivée de la classe `SimilarityMeasure` pour calculer la similarité cosinus entre les items.
- `JaccardSimilarityMeasure` : Une classe dérivée de la classe `SimilarityMeasure` pour calculer la similarité de Jaccard entre les items.
- `EuclidianSimilarityMeasure` : Une classe dérivée de la classe `SimilarityMeasure` pour calculer la similarité Euclidienne entre les items.
- `SelectionStrategy` : Une classe abstraite qui encapsule les données et les opérations nécessaires pour appliquer une stratégie de sélection des items les plus similaires à un item donné pour un user spécifique (Stratégie plus proches voisins et Stratégie de seuillage). Elle est dédiée à la spécialisation par héritage.
- `KNNStrategy` : Une classe dérivée de la classe `SelectionStrategy` pour sélectionner les items selon la stratégie des plus proches voisins.
- `ThresholdStrategy` : Une classe dérivée de la classe `SelectionStrategy` pour sélectionner les items selon la stratégie de seuillage.
- `IBCFSystem` : Une classe pour encapsuler le processus d'apprentissage (calcul de la matrice de similarité des items à partir du `MaskedDataFrame`) et de prédiction (calcul du score le plus proche qu'un user aurait attribué à un item à suggérer) d'un système de recommandation IBCF.

Interface de la Classe `DataFrame`

Rôle & responsabilités

Cette classe encapsule une matrice des scores données par des users pour des items. Cette matrice est à deux axes respectivement indexables par les noms des users et les noms des items.

Attributs

- `data` : La matrice des scores (instance de la classe `numpy.ndarray`).
- `rowMap` : Un dictionnaire qui relie les noms des users (clés) aux indices des lignes associés (valeurs).
- `colMap` : Un dictionnaire qui relie les noms des items (clés) aux indices des colonnes associés (valeurs).

Méthodes à implémenter

Q1. `__init__(...)` : Initialise une instance à partir des paramètres `data`, `rows` et `columns` représentant respectivement la matrice des scores, la liste des noms des utilisateurs et la liste des noms des items supposées énumérées dans l'ordre des lignes et des colonnes associées.

Espace de réponse pour Q1

```
import numpy as np
class DataFrame:
   def __init__(self, data, rows, columns):
```

Q2. `mapRow(...)` : reçoit un str `rowName` qui correspond au nom d'un user et retourne l'indice de la ligne correspondante.

Espace de réponse pour Q2

```
   def mapRow(self, rowName):
```

Q3. `mapCol(...)` : reçoit un `str colName` qui correspond au nom d'un item et retourne l'indice de la colonne correspondante.

Espace de réponse pour Q3

```
   def mapCol(self, colName):
```

Q4. `mapPos(...)` : reçoit un `tuple locName` contenant deux chaînes qui correspondent respectivement à un nom d'user et un nom d'item et retourne un `tuple` contenant l'indice de la ligne et l'indice de la colonne associées.

— $D_{\text{en ligne}}$ représente la durée cumulée où le datacenter était en ligne,
— D_{total} représente la somme de toutes les durées cumulées pour tous les statuts possibles.

Pour exprimer D_{total} sous forme d'une équation, on peut écrire :

$$D_{\text{total}} = \sum_{s \in \text{Statuts}} D_s$$

Espace de réponse pour Q31

```
def DCavailability(cur, dataCenterID):
```

Espace de réponse pour Q29

```
def ServerMonitoring(cur, serverID):
```

Q30. Écrire la fonction `DCStatutTimeSpan` qui prend deux paramètres `cur` et `dataCenterID` l'identifiant d'un centre de calcul et retourne un dictionnaire `ddcStat` où les clés sont les statuts qui ont été observés durant le monitoring des serveurs installés dans ce centre de calcul (`"En ligne"`, `"Hors ligne"`, `"En maintenance"`, `"En panne"`, `"En surcharge"`) et les valeurs sont les durées de temps cumulés de tous les serveurs pour chaque statut.

Espace de réponse pour Q30

```
def DCStatutTimeSpan(cur, dataCenterID):
```

Q31. Écrire la fonction `DCavailability` qui prend deux paramètres `cur`et `dataCenterID`, l'identifiant d'un centre de calcul, et retourne `disponibility`, le ratio de disponibilité de ce centre de calcul. Sachant que le ratio de disponibilité d'un centre de calcul est le rapport entre toutes les durées de temps cumulées de ses serveurs pour le statut `"En ligne"`, sur la somme des durées de temps cumulées pour tous les états possibles.

$$\text{Disponibility} = \frac{D_{\text{en ligne}}}{D_{\text{total}}}$$

Espace de réponse pour Q4

```
def mapPos(self, locName):
```

Q5. `getRowNames(...)` : qui retourne les noms des lignes sous forme d'une liste triée selon l'ordre des indices.

Espace de réponse pour Q5

```
def getRowNames(self):
```

Q6. `getColumnNames(...)` : qui retourne les noms des colonnes sous forme d'une liste triée selon l'ordre des indices.

Espace de réponse pour Q6

```
def getColumnNames(self):
```

Q7. `__getitem__(...)` : méthode spéciale qui reçoit un `tuple` `locName` contenant deux chaînes qui correspondent respectivement à un nom d'user et un nom d'item et retourne le score associé à partir de la matrice `data`.

Espace de réponse pour Q7

```
def __getitem__(self, locName):
```

Interface de la classe `SimilarityMatrix`

Rôle & responsabilités

Cette classe hérite de la classe `DataFrame`, elle encapsule une matrice carrée contenant les similarités entre toutes les paires d'items. Ainsi, les deux axes sont indexables par les noms des items.

Méthodes à implémenter

Q8. `__init__(...)` : Initialise une nouvelle instance à partir des paramètres `data`, `itemNames` représentant respectivement la matrice des similarités entre les paires des items et la liste des noms des items.

Espace de réponse pour Q8

```
class SimilarityMatrix(DataFrame):
    def __init__(self, data, itemNames):
```

Interface de la classe `MaskedDataFrame`

Rôle & responsabilités

Cette classe hérite de la classe `DataFrame`, elle encapsule une matrice des scores donnés par des users pour certains items. Cette matrice est à deux axes respectivement indexables par les noms des users et les noms des items.

- Une interaction user, item (produisant un score effectif) est marquée par la valeur `False` au niveau du masque.
- L'absence d'interaction user, item (décrite par le symbole `"-"` dans la figure 1) est marquée par la valeur `True` au niveau du masque.

Espace de réponse pour Q28

SQLITE

Dans cette partie, on suppose que la table suivante a été ajoutée à la base de données :

- `Monitoring`(`MonitoringID`, `DateSurv`, `Statut`, `Commentaires`, `DuréeStatut`, #`UserID`, #`ServeurID`)
  La table `Monitoring` enregistre les informations relatives à des prélèvements réguliers pour assurer la surveillance de performances et de disponibilité des serveurs pour les utilisateurs. C'est le basculement de statut d'un serveur qui provoque l'insertion d'un nouveau tuple dans cette base.
  - `MonitoringID` : utilisé pour identifier de manière unique chaque enregistrement de surveillance, de type entier clé primaire.
  - `DateSurv` : indique la date et l'heure auxquelles la surveillance a été effectuée, de type date et heure selon le format `"AAAA-MM-JJ HH:MM:SS"`.
  - `Statut` : Il indique l'état opérationnel du serveur au moment de la surveillance, ce qui peut être utile pour analyser les performances et la disponibilité du serveur au fil du temps. Ce champ pourrait inclure des valeurs telles que `"En ligne"`, `"Hors ligne"`, `"En maintenance"`, `"En panne"`, `"En surcharge"`, etc., de type chaîne de caractères.
  - `Commentaires` : champ permettant d'enregistrer des commentaires ou des notes sur le déroulement de la surveillance, de type chaîne de caractères.
  - `DuréeStatut` : La durée exprimée en millisecondes pendant laquelle le serveur reste dans le statut actuel avant de basculer vers un autre statut, de type entier.
  - `UserID` : identifiant de l'entreprise locataire de type entier clé étrangère qui fait référence à la table Utilisateurs.
  - `ServeurID` : identifiant d'un serveur sur lequel les données de l'utilisateur sont enregistrées de type entier, clé étrangère qui fait référence à la table Serveurs.

Dans la suite, les fonctions demandées doivent être écrites en Python en désignant par `cur` le curseur d'exécution des requêtes.

Q29. Écrire la fonction `ServerMonitoring` qui prend deux paramètres `cur` et `serverID` l'identifiant d'un serveur et retourne un dictionnaire `dStat` où les clés sont les statuts qui ont été observés durant le monitoring de ce serveur et les valeurs sont des ensembles contenant les dates et heures des observations de ces statuts.

Dans la suite on suppose que les tables de la base de données sont remplies en respectant les contraintes et les règles de gestion su citées.

Q25. Calculer la capacité totale de stockage pour chaque centre de calcul.

Espace de réponse pour Q25

Q26. Déterminer tous les détails relatifs aux logiciels qui pour le moment n'ont aucune licence.

Espace de réponse pour Q26

Q27. Déterminer toutes les informations relatives aux serveurs qui hébergent des logiciels avec des licences de type `"logiciel libre"`.

Espace de réponse pour Q27

Q28. Déterminer les identifiants, les capacités serveurs et les capacités disponibles (égale à la capacité serveur du DataCenter – nombre de serveurs effectivement affectés à ce DataCenter) le résultat sera trié par ordre décroissant des capacités disponibles.

Attributs

— `mask` : Une matrice de booléens (instance de la classe `numpy.ndarray`) ayant les même dimensions que l'attribut hérité `data` où les cases à `True` sont les cases masquées (les items que l'utilisateur n'a pas évalué).

Pour l'exemple de données illustrés dans la figure 1, le masque associé est présenté comme suit dans la figure 2.

mask =

	Math	History	Programming	English	Chemistry	Art
User1	False	False	False	True	False	False
User2	True	False	False	False	True	False
User3	False	True	False	False	False	True
User4	False	False	True	False	False	False
User5	False	True	False	False	True	False
User6	False	False	False	False	False	False
User7	True	False	True	False	False	False
User8	False	True	False	False	False	False

FIGURE 2 – Masque associé aux données de la figure 1

Méthodes à implémenter

Q9. `__init__(...)` : Initialise une instance à partir des paramètres data, `rows`, `columns` et `mask` représentant respectivement la matrice des scores, la liste des noms des utilisateurs, la liste des noms des items et la matrice du masque.

Espace de réponse pour Q9

```
class MaskedDataFrame(DataFrame):
   def __init__(self, data, rows, columns , mask):
```

Q10. `__getitem__(...)` : méthode spéciale qui reçoit un tuple `locName` contenant deux chaînes qui correspondent respectivement à un nom d'user et un nom d'item et retourne le score associé à partir de la matrice `data` si l'user a évalué l'item et `None`, sinon.

Espace de réponse pour Q10

```
def __getitem__(self, loc_name):
```

Q11. `getInteractions(...)` : prend le nom d'un item `colName` et retourne un ensemble contenant les noms des users qui ont interagit avec cet item.

Espace de réponse pour Q11

```
def getInteractions(self, colName):
```

Q12. `getDomain(...)` : extrait un ensemble contenant les noms des items qui ont été évalués par l'user `rowName`.

Espace de réponse pour Q12

```
def getDomain(self, rowName):
```

Q13. `commonRows(...)` : prend deux noms d'items `colName1` et `colName2` et en considérant uniquement les users qui ont interagit avec les deux items en question, extrait un `tuple` de deux vecteurs (`numpy.ndarray`) contenant les scores attribués.

Travail demandé

Algèbre relationnelle

Exprimer en algèbre relationnelle les requêtes permettant de :

Q22. Lister les identifiants des serveurs avec leurs capacités de stockage.

Espace de réponse pour Q22

Q23. Lister les identifiants et les adresses IP des serveurs qui hébergent des logiciels utilisés par des entreprises qui œuvrent dans le domaine de l'assurance.

Espace de réponse pour Q23

SQL

Q24. Écrire la requête qui permet de créer la table `Affectations` en supposant que toutes les tables référencées sont déjà créées.

Espace de réponse pour Q24

- `Licences`(<u>`LicenceID`</u>, `TypeLicence`, `DateExp`, #`LogicielID`)
  La table `Licences` enregistre les informations relatives aux licences permises aux différents logiciels. Certains logiciels libres sont livrés sans licences.
  - `LicenceID` : utilisé pour identifier de manière unique chaque licence, de type entier clé primaire.
  - `TypeLicence` : représente le type de distribution d'une licence, par exemple, propriétaire, open source, libre, commercial, etc., de type chaîne de caractères.
  - `DateExp` : date à laquelle la licence expire, indiquant jusqu'à quand le logiciel peut être légalement utilisé avec cette licence, de type date selon le format `"AAAA-MM-JJ"`.
  - `LogicielID` : identifiant du logiciel associé à une licence donnée, clé étrangère qui fait référence à la table Logiciels.
- `Utilisateurs`(<u>`UserID`</u>, `NomUser`, `Mail`, `Rôle`)
  La table `Utilisateurs` enregistre les informations relatives aux entreprises qui louent des services de Cloud Computing.
  - `UserID` : identifiant de l'entreprise locataire d'un ou plusieurs services, de type entier clé primaire.
  - `NomUser` : nom de l'entreprise locataire, de type chaîne de caractères.
  - `Mail` : adresse mail de l'entreprise locataire, de type chaîne de caractères.
  - `Rôle` : activité économique de l'entreprise locataire, par exemple, une banque, une assurance, une boîte de développement, etc., de type chaîne de caractères.
- `Affectations`(<u>`AffectationID`</u>, `DateAffect`, `DuréeEstimée`, #`UserID`, #`ServeurID`, #`LicenceID`)
  La table `Affectations` enregistre les informations relatives à un acte de location qui se traduit par une affectation d'une licence logiciel installée sur un serveur pour un utilisateur donné.
  - `AffectationID` : utilisé pour identifier de manière unique chaque affectation de type entier, clé primaire.
  - `DateAffect` : Indique la date à laquelle l'affectation a été effectuée, de type date selon le format `"AAAA-MM-JJ"`.
  - `DuréeEstimée` : Représente la durée estimée de l'affectation, exprimée en heures, de type entier.
  - `UserID` : identifiant de l'entreprise locataire, de type entier clé étrangère qui fait référence à la table Utilisateurs.
  - `ServeurID` : identifiant d'un serveur sur lequel les données de l'utilisateur sont enregistrées, de type entier clé étrangère qui fait référence à la table Serveurs.
  - `LicenceID` : identifiant de la licence louée par un utilisateur, de type entier, clé étrangère qui fait référence à la table Licences.

Espace de réponse pour Q13

```
def commonRows(self, colName1, colName2):
```

Interface de la classe `SimilarityMeasure`

Rôle & responsabilités

Cette classe abstraite encapsule la logique de production d'une matrice de similarité selon une métrique spécifiée par le biais de l'héritage.

Attributs

- `mdf` : instance de la classe `MaskedDataFrame`, à partir de laquelle la matrice de similarité des paires des items est calculée.

Méthodes à implémenter

La méthode `__init__(...)` : initialise l'attribut `mdf` à partir du paramètre d'entrée.

```
class SimilarityMeasure:
    def __init__(self, mdf):
        self.mdf = mdf
```

La méthode `measure(...)` : implémente une métrique de similarité. Cette méthode sera redéfinie dans les classes filles.

```
def measure(self, colName1, colName2):
    # pour cette classe le script de cette méthode
    # n'est pas demandé
```

Q14. `getSimilarityMatrix(...)` : calcule et retourne une instance de la classe `SimilarityMatrix` qui est la matrice de similarité entre toutes les paires des colonnes de l'attribut `mdf`.

Espace de réponse pour Q14

```
def getSimilarityMatrix(self):
```

Interface de la classe `CosineSimilarityMeasure`

Rôle & responsabilités

Cette classe concrète encapsule la logique de production d'une matrice de similarité selon le cosinus de l'angle entre les deux vecteurs v_i et v_j représentants les scores de deux items en considérant uniquement les users qui ont simultanément interagit avec ces deux items en question.

$$\text{CosineSimilarity}(\text{item}_i, \text{item}_j) = \frac{\sum_{u \in \mathcal{U}(\text{item}_i, \text{item}_j)} \text{score}_{u,i}\ \text{score}_{u,j}}{\sqrt{\sum_{u \in \mathcal{U}(\text{item}_i, \text{item}_j)} \text{score}_{u,i}^2} \sqrt{\sum_{u \in \mathcal{U}(\text{item}_i, \text{item}_j)} \text{score}_{u,j}^2}}$$

Cette formule est équivalente à :

$$\text{CosineSimilarity}(v_i, v_j) = \frac{v_i . v_j}{\|v_i\|\ \|v_j\|}$$

Équation 1 – Similarité Cosinus

Où :

- $\mathcal{U}(\text{item}_i, \text{item}_j)$ est l'ensemble des users qui ont simultanément interagit avec les items item_i et item_j.
- v_i, v_j sont deux vecteurs contenant les scores de deux items item_i et item_j en considérant uniquement les users qui ont simultanément interagit avec les deux items.
- $v_i.v_j$ représente le produit scalaire entre v_i et v_j.
- $\|v\|$ représente la norme euclidienne du vecteur v.

Méthodes à implémenter

Q15. `measure(...)` : calcule et retourne un réel représentant la similarité cosinus entre les deux items dont les noms sont passés dans les paramètres `colName1` et `colName2` comme illustré par l'équation 1.

Partie II : Base de Données Relationnelle

Le cloud computing fait référence à l'utilisation de la mémoire et des capacités de calcul des ordinateurs et des serveurs répartis dans le monde entier et liés par un réseau. Les applications et les données ne se trouvent plus sur un ordinateur déterminé d'une entreprise donnée mais dans un cloud composé de nombreux serveurs distants interconnectés formant un centre de calcul réparti. Les entreprises louent donc du matériel, des applications et des services chez des fournisseurs cloud computing pour réduire leurs charges de développement IT.

L'objectif de cette partie est de gérer la consommation des entreprises clientes en terme de services et de ressources matérielles.

Le schéma relationnel de la base de données décrit ci-dessous a été élaboré pour les besoins de l'énoncé.

- `DataCenters(DataCenterID, NomDC, Emplacement, CapacitéServ)`
  La table `DataCenters` enregistre les informations relatives aux centres de calcul qui offrent leurs services et ressources.
  - `DataCenterID` : identifiant d'un centre de calcul, de type entier clé primaire.
  - `NomDC` : nom du centre de calcul, de type chaîne de caractères.
  - `Emplacement` : désigne l'emplacement physique du centre de calcul, de type chaîne de caractères.
  - `CapacitéServ` : le nombre maximum de serveurs qu'un centre de calcul peut contenir, de type entier.
- `Serveurs(ServeurID, NomSrv, AdresseIP, CapacitéStk, Statut, #DataCenterID)`
  La table `Serveurs` enregistre les informations relatives aux serveurs installés dans les différents centres de données.
  - `ServeurID` : utilisé pour identifier de manière unique chaque serveur, de type entier clé primaire.
  - `NomSrv` : nom du serveur, de type chaîne de caractères.
  - `AdresseIP` : c'est l'adresse IP du serveur, de type chaîne de caractères.
  - `CapacitéStk` : indique la capacité de stockage du serveur, exprimée en téraoctets, de type entier.
  - `Statut` : représente l'état actuel du serveur, par exemple, s'il est en ligne, hors ligne, en maintenance, etc., de type chaîne de caractères.
  - `DataCenterID` : identifiant d'un centre de calcul de type entier, clé étrangère qui fait référence à la table DataCenter.
- `Logiciels(LogicielID, NomLog, Version, Editeur, Type)`
  La table `Logiciels` enregistre les informations relatives aux applications offertes par le centre de calcul.
  - `LogicielID` : utilisé pour identifier de manière unique chaque logiciel, de type entier clé primaire.
  - `NomLog` : nom du logiciel, de type chaîne de caractères.
  - `Version` : version spécifique actuelle du logiciel, de type chaîne de caractères.
  - `Editeur` : nom de l'entreprise ou de l'individu qui a développé ou édité le logiciel, de type chaîne de caractères.
  - `Type` : type du logiciel, par exemple, système d'exploitation, application de commerce électronique, application de gestion de stock, utilitaire, etc., de type chaîne de caractères.

Espace de réponse pour Q21

```
def predict(self, rowName, colName):
```

Espace de réponse pour Q15

```
class CosineSimilarityMeasure(SimilarityMeasure):
   def measure(self, colName1,  colName2):
```

Interface de la classe `JaccardSimilarityMeasure`

Rôle & responsabilités

Cette classe concrète encapsule la logique de production d'une matrice de similarité selon l'indice de Jaccard qui capture le rapport entre l'intersection et l'union entre les deux vecteurs v_i et v_j représentants les scores de deux items en considérant uniquement les users qui ont simultanément interagit avec ces deux items en question.

$$\text{JaccardSimilarity}(\text{item}_i, \text{item}_j) = \frac{\sum_{u \in \mathcal{U}(\text{item}_i, \text{item}_j)} \min\left(\text{score}_{u,i}, \text{score}_{u,j}\right)}{\sum_{u \in \mathcal{U}(\text{item}_i, \text{item}_j)} \max\left(\text{score}_{u,i}, \text{score}_{u,j}\right)}$$

Cette formule est équivalente à :

$$\text{JaccardSimilarity}(v_i, v_j) = \frac{\sum_u \min\left(v_{i_u}, v_{j_u}\right)}{\sum_u \max\left(v_{i_u}, v_{j_u}\right)}$$

Équation 2 – Similarité de Jaccard

Où :

- $\mathcal{U}(\text{item}_i, \text{item}_j)$ est l'ensemble des users qui ont simultanément interagit avec les items item_i et item_j.
- v_i, v_j sont deux vecteurs contenant les scores de deux items item_i et item_j en considérant uniquement les users qui ont simultanément interagit avec les deux items.
- v_{i_u} est l'élément qui correspond à l'utilisateur u dans le vecteur v_i.

Méthodes à implémenter

Q16. `measure(...)` : calcule et retourne un réel représentant la similarité de Jaccard entre les deux items dont les noms sont passés dans les paramètres `colName1` et `colName2` comme illustré par l'équation 2.

Espace de réponse pour Q16

```
class JaccardSimilarityMeasure(SimilarityMeasure):
   def measure(self, colName1, colName2):
```

Interface de la classe `EuclidianSimilarityMeasure`

Rôle & responsabilités

Cette classe concrète encapsule la logique de production d'une matrice de similarité basée sur la distance euclidienne qui représente la longueur de la droite qui relie deux vecteurs v_i et v_j représentants les scores de deux items en considérant uniquement les users qui ont simultanément interagit avec ces deux items en question.

$$\text{EuclidianSimilarity}(\text{item}_i, \text{item}_j) = \frac{1}{1 + \sqrt{\sum_{u \in \mathcal{U}(\text{item}_i, \text{item}_j)} (\text{score}_{u,i} - \text{score}_{u,j})^2}}$$

Cette formule est équivalente à :

$$\text{EuclidianSimilarity}(v_i, v_j) = \frac{1}{1 + \sqrt{\sum_{u} (v_{i_u} - v_{j_u})^2}}$$

Équation 3 – Similarité euclidienne

Où :

- $\mathcal{U}(\text{item}_i, \text{item}_j)$ est l'ensemble des users qui ont simultanément interagit avec les items item_i et item_j.
- v_i, v_j sont deux vecteurs contenant les scores de deux items item_i et item_j en considérant uniquement les users qui ont simultanément interagit avec les deux items.
- v_{i_u} est l'élément qui correspond à l'utilisateur u dans le vecteur v_i.

Espace de réponse pour Q20

```
def fit(self, mdf):
```

Q21. La méthode `predict(...)` : implémente la phase de prédiction du score qu'un user `rowName` aurait donnée à un item `colName`. Cette méthode doit effectuer les étapes suivantes :

1. En utilisant l'attribut `mdf` tester si l'user a déjà interagit avec l'item et retourner le score attribué si c'est le cas.
2. Sinon, construire `colDomain` l'ensemble des items que l'user `rowName` a interagit avec.
3. Si `colDomain` est vide, alors retourner `None`.
4. Sinon, appliquer la stratégie de sélection pour construire une liste `selected` contenant les items les plus similaires à `colName` qui ont été appréciés par l'user `rowName`.
5. Calculer puis retourner $\hat{score}_{\text{rowName,colName}}$ le score que l'user `rowName` aurait attribué à l'item `colName` en appliquant l'équation suivante :

$$\hat{\text{score}}_{\text{rowName,colName}} = \frac{\sum_{i \in \text{selected}} \text{sMatrix}_{i,\text{colName}} \, \text{score}_{\text{rowName},i}}{\sum_{i \in \text{selected}} |\text{sMatrix}_{i,\text{colName}}|}$$

Équation 4 – Prédiction du score

Interface de la classe `IBCFSystem`

Rôle & responsabilités

Cette classe joue le rôle d'une façade qui exploite les classes précédentes pour implémenter un système de recommandation basé sur le modèle **IBCF** son interface offre deux méthodes :

- `fit(...)` : qui assure l'apprentissage des similarités à partir d'un `MaskedDataFrame`.
- `predict(...)` : qui exploite les données apprises pour prédire un score potentiel qu'un utilisateur aurait attribué à un item à suggérer.

Attributs

- `sMeasure` : une chaîne de caractère décrivant la mesure de similarité que le système doit adopter, elle supporte les valeurs `"cosine"`, `"jaccard"` ou bien `"euclidian"`.
- `selectParameter` : une valeur numérique décrivant le paramètre de la stratégie de sélection à adopter. Elle peut prendre deux formes : `KNNStrategy` pour un paramètre de type `int` (paramètre formel `n`) et `ThresholdStrategy` pour un paramètre de type `float` (paramètre formel `alpha`).
- `mdf` : instance de la classe `MaskedDataFrame`.
- `sMatrix` : instance de la classe `SimilarityMatrix`.
- `strategy` : instance de la classe `SelectionStrategy`.
- `mdf`, `sMatrix` et `strategy` sont des attributs qui seront initialisés dans la méthode `fit` (à traiter dans Q20).

Méthodes à implémenter

La méthode `__init__(...)` : initialise les attributs à partir des deux paramètres d'entrée. Cette méthode est implémentée comme suit :

```
class IBCFSystem:
    def __init__(self, similarityMeasure, selectParameter):
        self.sMeasure = similarityMeasure
        self.selectParameter = selectParameter
```

Q20. La méthode `fit(...)` : implémente la phase d'apprentissage des similarités à partir du paramètre `mdf` instance de la classe `MaskedDataFrame`. Cette méthode doit appliquer la démarche suivante :

1. Créer un attribut `mdf` à partir du paramètre d'entrée.
2. Selon l'attribut `sMeasure` et en utilisant l'attribut `mdf`, instancier la classe dérivée représentant la Mesure de similarité adéquate et l'affecter à l'attribut `sMeasure`.
3. Construire la matrice de similarités et l'affecter à l'attribut `sMatrix`.
4. Selon la nature du paramètre `selectParameter`, instancier la classe dérivée représentant la Stratégie de sélection adéquate et l'affecter à l'attribut `strategy`.

Méthodes à implémenter

Q17. `measure(...)` : calcule et retourne un réel représentant la similarité euclidienne entre les deux items dont les noms sont passés dans les paramètres `colName1` et `colName2` comme illustré par l'équation 3.

Espace de réponse pour Q17

```
class EuclidianSimilarityMeasure(SimilarityMeasure):
    def measure(self, colName1, colName2):
```

Interface de la classe `SelectionStrategy`

Rôle & responsabilités

Cette classe abstraite encapsule la stratégie d'extraction des items les plus similaires à un item donné à partir d'un domaine d'items qui contient les items avec lesquels un utilisateur donné a interagi. La similarité entre les items est capturée dans une matrice de similarité attachée aux instances de cette classe.

Attributs

- `sMatrix` : instance de la classe `SimilarityMatrix`, contenant les similarités entre toutes les paires possibles d'items.

Méthodes à implémenter

La méthode `__init__(...)` : initialise l'attribut `sMatrix` à partir du paramètre d'entrée.

```
class SelectionStrategy:
    def __init__(self, sMatrix):
        self.sMatrix = sMatrix
```

La méthode `select(...)` : implémente une stratégie de sélection des items les plus similaires à un item donné `colName` à partir d'un ensemble d'items `colDomain`. Cette méthode sera redéfinie dans les classes filles.

```
def select(self, colName, colDomain):
    # pour cette classe le script de cette méthode
    # n'est pas demandé
```

Interface de la classe `KNNStrategy`

Rôle & responsabilités

Cette classe concrète implémente une stratégie de sélection qui extrait les `n` plus proches voisins à un item donné à partir du domaine d'items. La notion de proximité entre les items est proportionnelle à la similarité entre les items représentée par l'attribut `sMatrix`.

Attributs

— `n` : entier positif qui représente le nombre de voisins les plus similaires relativement à `colName` que la méthode `select` doit extraire à partir du domaine d'items `colDomain`.

Méthodes à implémenter

La méthode `__init__(...)` : initialise à partir des paramètres, les attributs hérités et l'attribut `n`. Cette méthode est implémentée comme suit :

```
class KNNStrategy(SelectionStrategy):
    def __init__(self, smatrix, n):
        super().__init__(smatrix)
        self.n = n
```

Q18. La méthode `select(...)` : implémente une stratégie de sélection des items les plus similaires à un item donné `colName` à partir d'un ensemble d'items `colDomain`. Cette méthode doit trier les items de `colDomain` selon leurs similarités relatives à l'item `colName` puis retourne les `n` premiers items de cette liste triée.

Espace de réponse pour Q18

```
def select(self, colName, colDomain):
```


Interface de la classe `ThresholdStrategy`

Rôle & responsabilités

Cette classe concrète implémente une stratégie de sélection qui extrait du domaine d'items `colDomain` tous les items dont la similarité relative à l'item `colName` est supérieure ou égale à un seuil `alpha`.

Attributs

— `alpha` : réel positif représentant le seuil de similarité minimale, permettant de déterminer les items à extraire du domaine d'items `colDomain`, en fonction de leurs similarités relatives à l'item `colName`.

Méthodes à implémenter

La méthode `__init__(...)` : initialise à partir des paramètres, les attributs hérités et également l'attribut `alpha`. Cette méthode est implémentée comme suit :

```
class ThresholdStrategy(SelectionStrategy):
    def __init__(self, sMatrix, alpha):
        super().__init__(sMatrix)
        self.alpha = alpha
```

Q19. La méthode `select(...)` : implémente une stratégie de sélection des items les plus similaires à un item donné `colName` à partir d'un ensemble d'items `colDomain`. Elle retourne la liste des items de `colDomain` ayant une similarité relative à `colName` supérieure ou égale à `alpha`.

Espace de réponse pour Q19

```
def select(self, colName, colDomain):
```

Annexe

Quelques Fonctions/Méthodes Python/Schéma Relationnel BD

Sans aucune obligation, les fonctions suivantes pourraient vous être utiles.

Opérations sur les itérables (`str, tuple, list, dict, etc.`)

- `len(it)` retourne le nombre d'éléments de l'itérable it.
- `range(d,f,p)` retourne la séquence des valeurs entières successives comprises entre d et f, f exclu, par $pas = p$.
- `sum(it)` retourne la somme de tous les éléments de l'itérable it.
- `min(it)` retourne la valeur minimale de l'itérable it.
- `min(it, key = fct)` retourne la valeur minimale de l'itérable it où la fonction key définit le critère de comparaison.
- `max(it)` retourne la valeur maximale de l'itérable it.
- `max(it, key = fct)` retourne la valeur maximale de l'itérable it où la fonction key définit le critère de comparaison.
- `sum(it)` retourne la somme des éléments de l'itérable it.
- `x in it` vérifie si x appartient à it.
- `sorted(it)` retourne une liste contenant les éléments de it dans l'ordre croissant.
- `sorted(it, key = fct, reverse = True)` retourne une liste contenant les éléments de it dans l'ordre décroissant où la fonction key définit le critère de comparaison.
- `lst.sort()` trie la liste lst dans l'ordre croissant.
- `lst.sort(key = fct, reverse = True)` trie la liste lst en ordre décroissant en utilisant key comme critère de comparaison.
- `lst.count(val)` retourne le nombre d'occurrences de val dans la liste lst.
- `lst.append(val)` ajoute val à la fin de la liste lst.
- `lst.remove(val)` supprime la première occurrence val de la liste lst.
- `lst.pop(idx)` supprime puis retourne la valeur située à la position `idx` de la liste lst.
- `lst.index(val)` retourne l'indice de la première occurrence val de la liste lst.
- `source.split(motif)` retourne une liste formée par des chaînes de caractères résultantes du découpage de la chaîne source autour de la chaîne motif.
- `motif.join(itérable de chaînes)` retourne une chaîne de caractères résultante de la concaténation des éléments de l'itérable intercalés par le motif.
- `motif.format(paramètres)` retourne une chaîne de caractères obtenue en substituant dans l'ordre chaque caractère dans motif par un objet dans paramètres.
- `d.values()` retourne un itérable formé par les valeurs du dictionnaire d.

— `d.items()` retourne un itérable de couples (k,v) ou k est une clé du dictionnaire d et v est la valeur associée.

Module numpy

— `M.shape` ou `np.shape(M)` retourne un `tuple` formé par le nombre de lignes et le nombre de colonnes d'une matrice M.

— `np.array(lst)` construit une instance de la classe `np.ndarray` dont les composantes sont initialisées à partir de la liste passée en entrée.

— `np.zeros(tpl)` construit une instance de la classe `np.ndarray` dont les composantes sont initialisées par zéro ayant un `shape` comme décrit par le `tuple tpl`.

— `np.identity(n)` construit une matrice identité de taille $n \times n$.

— `np.dot(a, b)` effectue le produit matriciel/scalaire des tableaux `a` et `b`.

— `np.linalg.norm(x)` calcule la norme du vecteur ou de la matrice `x`.

Module sqlite3

— `cur.execute(req)` exécuter la requête SQL décrite par le `str req` à partir du curseur `cur`.

— `cur.execute(req, seq)` exécuter la requête paramétrée décrite en SQL par le `str req` à partir du curseur `cur`.

— `cur.fetchall()` récupère tous les résultats de la dernière requête exécutée avec le curseur `cur` et les retourne sous forme de liste de tuples.

Schéma relationnel BD

- DataCenters(DataCenterID, NomDC, Emplacement, CapacitéServ)
- Serveurs(ServeurID, NomSrv, AdresseIP, CapacitéStk, Statut, #DataCenterID)
- Logiciels(LogicielID, NomLog, Version, Editeur, Type)
- Licences(LicenceID, TypeLicence, DateExp, #LogicielID)
- Utilisateurs(UserID, NomUser, Mail, Rôle)
- Affectations(AffectationID, DateAffect, DuréeEstimée, #UserID, #ServeurID, #LicenceID)
- Monitoring(MonitoringID, DateSurv, Statut, Commentaires, DuréeStatut, #UserID, #ServeurID)