Ingénierie IA

Kindi BALDE

Réalisez une application mobile de recommandation de contenu

Présentation du projet

Problématique métier

My Content est une start-up qui veut encourager la lecture en recommandant des contenus pertinents pour ses utilisateurs. Elle souhaite donc tester une solution de recommandation d'articles et de livres à des particuliers.

Solution : un client utilisateur qui viendrait sur notre plateforme, nous voulons pouvoir lui proposer cinq articles à lire. Et cela, en se basant sur :

- son historique d'utilisation de notre plateforme (Content Based Filtering approche);
- ou bien en se basant sur son interaction avec nos articles et de son comportement en tant que lecteur (Collaborative Filtering)

Challenge

- Développement d'un modèle de recommandation pour notre plateforme de contenus
- Prise en compte de l'ajout de nouveaux articles et de nouveaux utilisateurs dans nos modèles
- Intégration du modèle choisi dans une application mobile pré-programmé.

Mission

En tant que CTO, ma mission est de développer une première version de l'application mobile avec l'un des techniques de système de recommandation.

Sommaire

1. Préparation des données
2. Modèle de type Content Based Filtering
3. Modèle de type Collaborative Filtering
4. Déploiement des Fonctions Azure et test en production
5. Environnement Bookshelf et Azure Functions
6. Description architectures MVP et future
7. Démonstration fonctionnelle de l’application

1. Préparation des données

Nous allons utiliser un datasets de la plateforme **G1 News Portal** en ligne contenant 46 047 Articles, plus de **3 million de clics** observés sur plus de **1 million de sessions** par **314 000 utilisateurs**.

Data : clicks; article metadata; articles embeddings

Pas de **rating** explicite pour avoir la table **user-article-rating**

Rating : Nous allons donc construire un rating implicite en nous basant sur l'interaction user-article. Le rating que nous avons créé est une combinaison de trois effets:

- **Article Effect** : prendre en compte le nombre de clics par article par utilisateur
- **Category Effect** : prendre en compte le nombre de clics par article par catégorie
- **Session Duration Effect** : la durée de la session ramenée à la longueur de l'article (word count)

2. Modèle Content Based Filtering

Content Based Filtering

On se base sur l'historique de l'utilisateur pour lui proposer des produits la prochaine fois qu'il vient sur notre site. Nous allons utiliser la description du contenu des articles que l'utilisateur consulte.

Une matrice de représentation (de description) des articles nous a été fournie. Il nous reste à construire une matrice de représentation des utilisateurs dans cette dimension, pour ensuite déduire (calculer la similarité) pour chaque utilisateur, la liste des cinq articles auxquels il est le plus proche dans cette dimension.

Avantage : Content Based Filtering permet de partir rapidement d'une matrice des articles et d'une du profil des utilisateurs pour construire une recommandation. rapide et efficace.

Inconvénients: cette méthode ne permet pas de prendre en compte de nouveaux utilisateurs et d'articles.

2. Modèle Content Based Filtering

- **Illustration**

1

```
emb_matrix.shape
```

```
(364047, 250)
```

2

```
user_article_matrix_df.head()
```

	user_id	article_id	rating
1	0	68866	0.451868
1535241	0	87205	0.372258
1535240	0	87224	0.127742
1287361	0	96755	0.294782
0	0	157541	0.274066

3

4

```
user_articles_pivot.shape
```

```
(280897, 500)
```

5

```
users_articles.shape, articles_feats.shape
```

```
(TensorShape([280897, 500]), TensorShape([500, 250]))
```

```
users_feats = tf.matmul(users_articles, articles_feats)
```

6

```
users_feats.shape
```

```
TensorShape([280897, 250])
```

```
for i in range(num_users):
    feature_names = [features[int(index)] for index in top_articles[i]]
    print('For User {}, We recommend those 5 articles : {}'.format(users[i],feature_names))
    print()
```

```
For User 0, We recommend those 5 articles : [157541, 209236, 206120, 202370, 205845]

For User 1, We recommend those 5 articles : [282785, 205845, 283327, 206120, 234267]
```

7

3. Modèle de type Collaborative Filtering

Le modèle de Collaborative Filtering se base sur l'interaction des utilisateurs et des articles pour construire un profil pour chaque utilisateur et des features pour chaque article. Et cela, en prenant en compte la similarité des utilisateurs en terme de lecture et celle des articles en terme d'utilisation.

Ce modèle est versatile en ce sens que l'on peut bien dire, que notre objectif est de recommander des utilisateurs à des articles

Très adapté à l'ajout des nouveaux utilisateurs et articles :

- pour un nouvel utilisateur, le modèle recommande les articles les plus populaires.
- pour un nouveau article, il y'a deux solutions:
    - utilisation de la matrice d'embedding pour représenter l'article dans une dimension dans laquelle il est comparable aux autres articles; ensuite on pourra affecter à ce nouvel article les ratings du produit qui lui est le plus proche en terme de similarité.
    - utiliser le modèle de recommandation pour prévoir les ratings de l'article par les utilisateurs.

3. Modèle de type Collaborative Filtering

- Illustration

```python
def reco_to_user_5_art(userId):
    """
    Arguments:
        userId: the user id from 0 to 46047
    Returns:
        a list of five articles to recommend to the User

    """
    reco_dict = {}
    # Please do not propose me articles that I already digested
    user_digest = df[df.user_id==userId].article_id.tolist()
    for art in art_list:
        if art not in user_digest:
            reco_dict[art]=model.predict(userId, art).est

    return list({k: v for k, v in sorted(reco_dict.items(),
                                         key=lambda item: item[1], reverse=True)}.keys())[:5]
```

1

```
The article 59929 is not in the set
Is the User 4 in test_set ?: False
Is the User 0 in test_set ?: True
```

2

```
reco_to_user_5_art(0)
```

```
[207122, 307561, 358085, 356647, 38708]
```

3

```
reco_to_user_5_art(4)
```

```
[83890, 207122, 37968, 100929, 70439]
```

4

```
reco_to_user_5_art(322899)
```

```
[207122, 119193, 48403, 183176, 38708]
```

5

4. Déploiement sur Azure Functions et test en production

Deux scénarios de déploiement validés :

1. Azure Functions - HttpTrigger, avec appel d'un moteur d'inférence:

   - Validation de la fonctionnalité du service de déploiement sur Azure ML : le service consomme bien un fichier json de clé **userId** et de valeur l'identité d'un utilisateur, et retourne une liste de produits recommandés.
   - Intégration du service sur l'application Azure Functions avec seulement du **HttpTrigger**: dès que l'URL est activé, il demande un json qu'il consomme aussitôt et produit à travers le moteur d'inférence une prédiction, la liste des articles à recommander.

2. Azure Functions - **HttpTrigger**, avec requête d'une Base de Données **Cosmos DB Table Storage** :

   - Création d'une BD Cosmos DB Table Storage et insertion des prévisions par utilisateur;
   - intégration des informations de connection de la BD sur l'application Azure Functions. Pour chaque activation de l'URL, une requête est envoyée à la BD Cosmos BD.

4. Déploiement sur Azure Functions et test en production

- Illustration

1

PartitionKey	RowKey	Timestamp	listValue
userId	0	Mon, 10 Jan 2022 14:43:25 GMT	[207122, 38708, 85043, 164213, 307561]
userId	1	Mon, 10 Jan 2022 14:43:25 GMT	[150028, 207122, 227545, 226632, 38708]
userId	2	Mon, 10 Jan 2022 14:43:25 GMT	[207122, 33463, 100956, 106649, 71298]
userId	3	Mon, 10 Jan 2022 14:43:25 GMT	[207122, 203863, 260466, 140488, 143444]

2

```
# AFTER CONFIG OF HTTP TRIGGER FUNCTION

!curl -X POST -H 'Content-type: application/json' \
     --data '{"userId":1}'\
     https://pythonfuncapplicationcosmos.azurewebsites.net/api/HttpCosmosDBTable
```

```
[150028, 207122, 227545, 226632, 38708]
```

3

```
print(service.scoring_uri)
```

```
http://118795b5-6f88-4795-a8c9-afdb53f2b858.eastus2.azurecontainer.io/score
```

4

```
# AFTER CONFIG OF HTTP TRIGGER FUNCTION
!curl -X POST -H 'Content-type: application/json' \
     --data '{"userId":0}'\
      https://pythonfuncapplication.azurewebsites.net/api/HttpTriggerFunctionOC
```

```
"[207122, 249782, 356647, 192079, 85224]"
```

5

```
test_samples = json.dumps({"userId": 0 })
pred = service.run(test_samples)
print(pred)
```

```
[207122, 249782, 356647, 192079, 85224]
```

5. Environnement Bookshelf et Azure Functions

Gestion des configurations de l'application Bookshelf

- Installation de **Android Developper**
- Installation (récuperation) de l'application à partir de GitHub en local ordinateur.
- Installation de **Node JS** et test du client **npm**.
- Déploiement en local à côté du dossier **Bookshelf**, le dossier de l'application Azure Functions.
- Configuration du **l'API_URL** de Azure Functions dans le fichier **config.json** de l'application Bookshelf.
- Emulation de l'application Android à partir du dossier BookShelf.

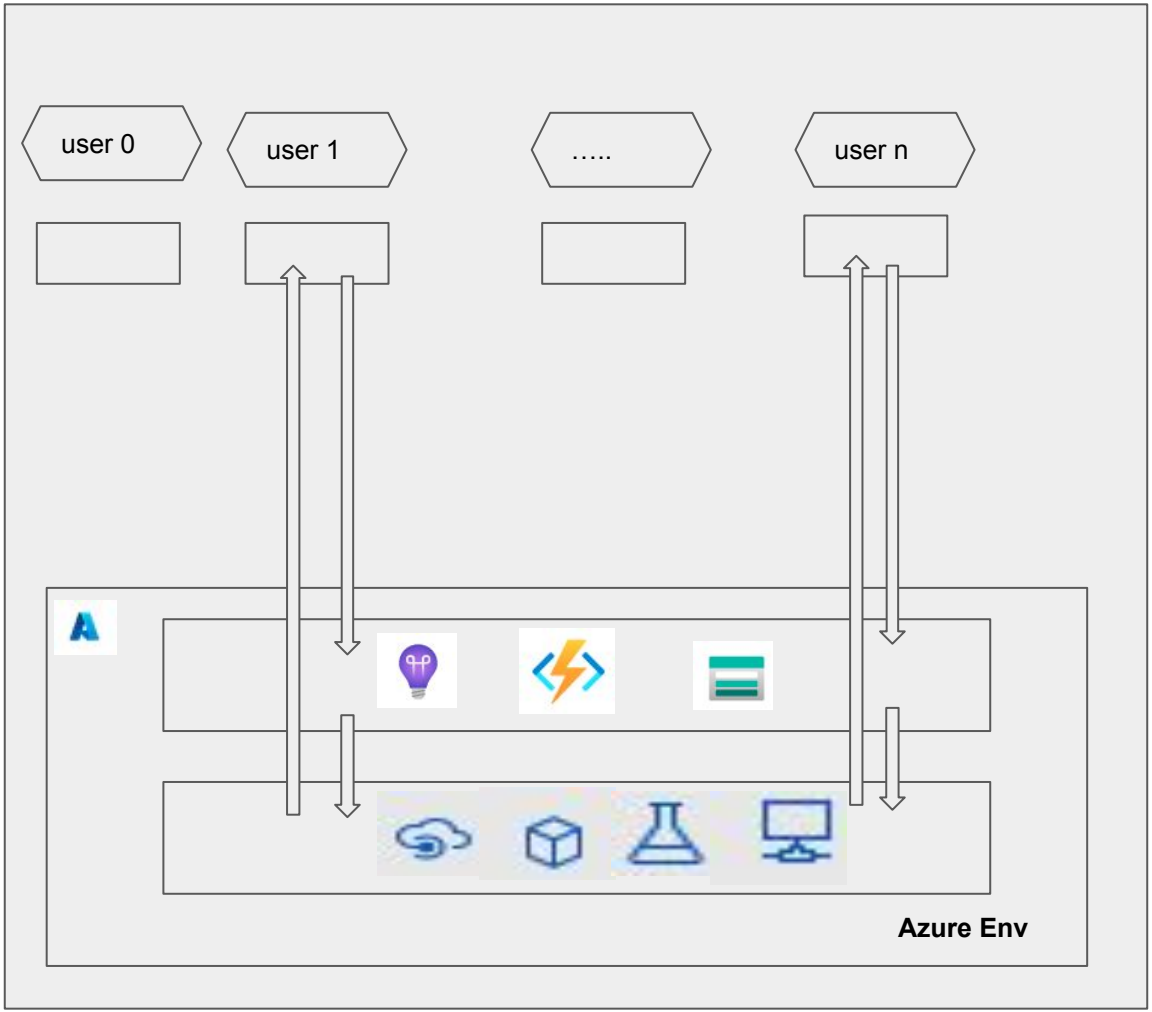
```
{
  "API_URL":"https://pythonfuncapplicationcosmos.azurewebsites.net/api/HttpCosmosDBTable"
}
```

6. Architectures

MVP

6. Architectures

Future



7. Démonstration fonctionnelle de l’application