Projet 10

Réalisez une application de recommandation de contenu

Introduction

Notre objectif principal est d'encourager notre audience à lire davantage en leur proposant des recommandations personnalisées. Concrètement, notre système vise à suggérer 5 lectures pertinentes à chaque utilisateur.

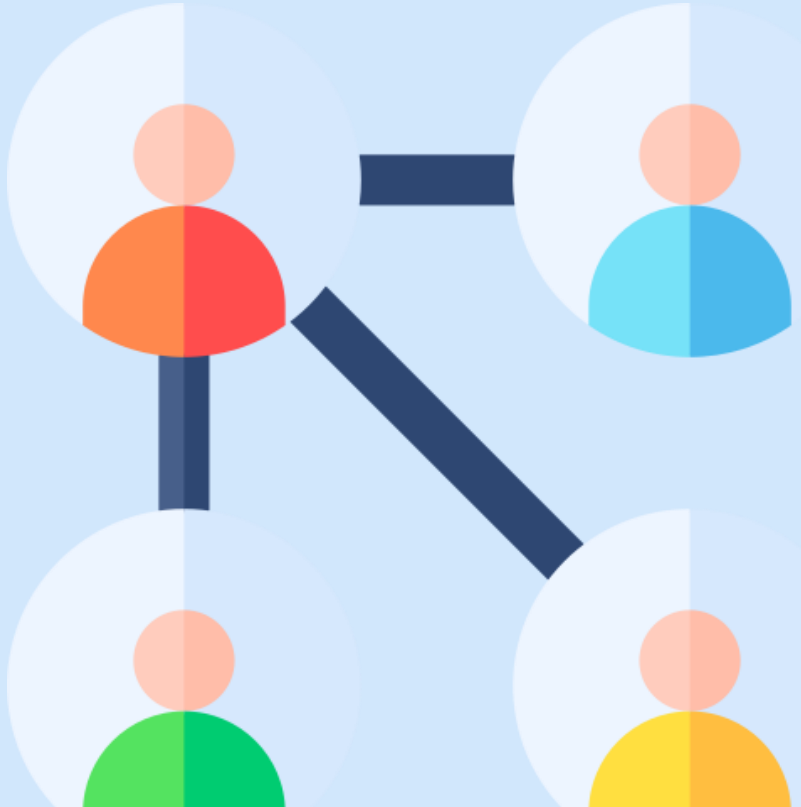
L'application que nous avons développée sera déployée sous forme d'API, accessible via un déclencheur HTTP. Cette approche garantit une intégration facile et une utilisation flexible dans divers contextes.

Pour ce MVP (Minimum Viable Product), nous avons travaillé avec un jeu de données open source, ce qui a apporté certaines contraintes. Au cours de cette présentation, nous expliquerons comment nous avons adapté notre développement pour relever ces défis spécifiques.

Enfin, nous aborderons les perspectives d'évolution de l'application, en explorant les possibilités d'amélioration et d'extension futures.

Cette présentation vous donnera un aperçu complet de notre démarche, des choix techniques effectués, et des résultats obtenus.

1. Contexte et importance des systèmes de recommandation
2. Modèles développés
3. Architecture retenue
4. Système de recommandation utilisé
5. Evolutions de l’architecture
6. Conclusion



Contexte et importance des systèmes de recommandation

Les systèmes de recommandation sont omniprésents dans le monde numérique d'aujourd'hui. Ils visent à attirer notre attention et à nous inciter à consommer certains produits, qu'il s'agisse de vidéos YouTube, de publications Instagram, d'offres Amazon Prime, de films Netflix, etc. Ces systèmes fonctionnent en analysant notre comportement passé pour prédire ce que nous aimerions.

Filtrage collaboratif

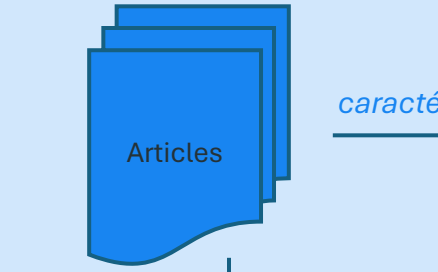
Filtrage basé sur le contenu

NLP

Modèles développés

Filtrage basé sur le contenu

Cette approche recommande des éléments similaires à ceux que l'utilisateur **a aimés** dans le passé en fonction des **caractéristiques des éléments** eux-mêmes. Par exemple, un système de recommandation musicale basé sur le contenu pourrait recommander des chansons du même genre ou du même artiste que celles que l'utilisateur a déjà écoutées.



caractérisés

notés

Note/avis utilisateur

Matrice d' embedding des articles

caractéristiques

articles

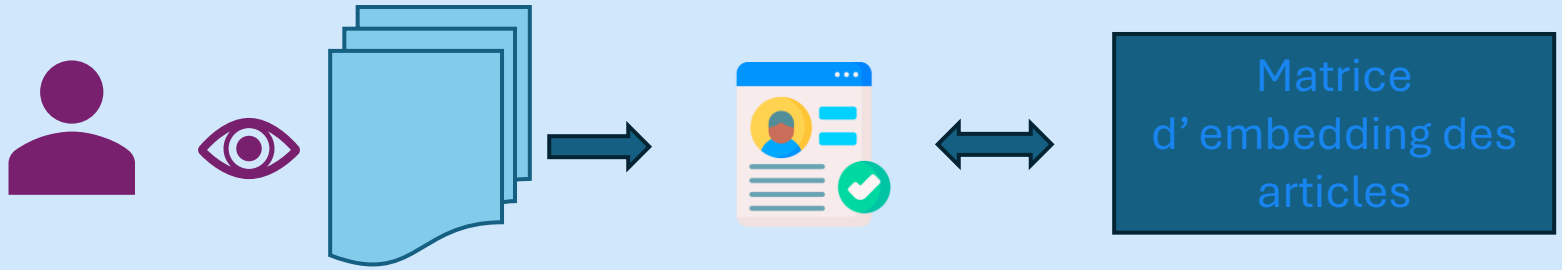
	0	1	2	3	4	247	248	249
0	-0.161183	-0.957233	-0.137944	0.050855	0.286870	-0.231686	0.597416	0.409623
1	-0.523216	-0.974058	0.738608	0.155234	-0.594137	0.182828	0.397090	-0.834364
2	-0.619619	-0.972960	-0.207360	-0.128861	-0.810877	-0.447580	0.805932	-0.285284
...	-0.740843	-0.975749	0.391698	0.641738	-0.910286	-0.537838	0.243541	-0.885329
364047	-0.279052	-0.972315	0.685374	0.113056	-0.917760	-0.424061	0.185484	-0.580292

Modèles développés

Filtrage basé sur le contenu

Avis explicites non disponibles dans notre jeu de données.

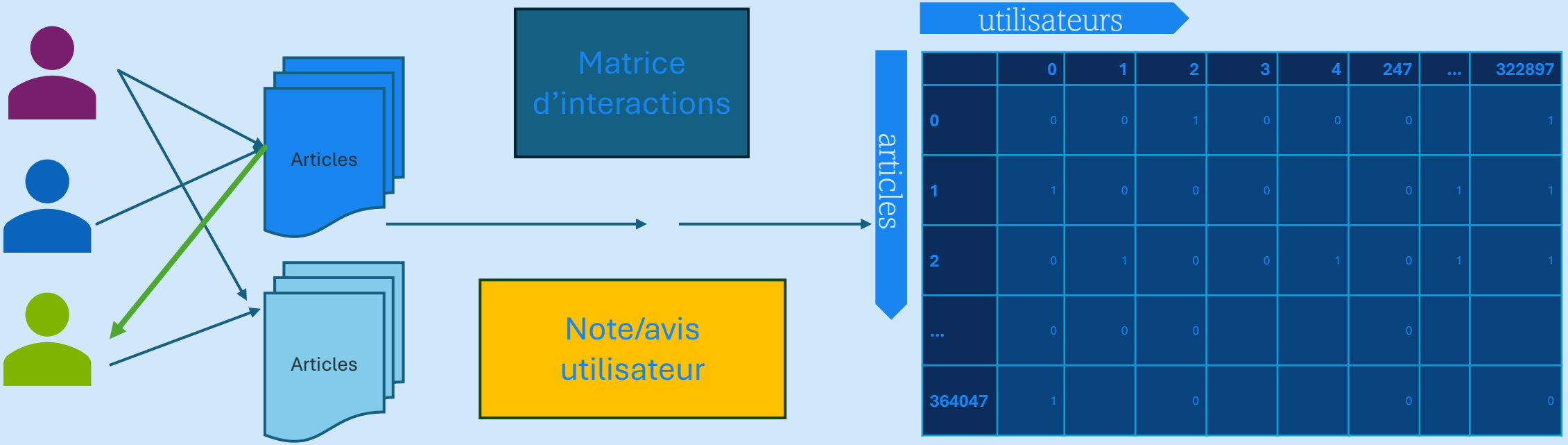
Nous nous appuierons sur le **profil de l'utilisateur** pour effectuer la recommandation



Modèles développés

Filtrage collaboratif

Le filtrage collaboratif repose sur l'idée que les utilisateurs ayant des goûts similaires dans le passé continueront d'avoir des goûts similaires à l'avenir. Il s'appuie sur les **interactions utilisateur-élément** pour découvrir des modèles dans les préférences des utilisateurs et faire des recommandations.



	0	1	2	3	4	247	...	322897
0	0	0	1	0	0	0		1
1	1	0	0	0		0	1	1
2	0	1	0	0	1	0	1	1
...	0	0	0			0		
364047	1		0			0		0

Modèles développés

Filtrage collaboratif

Rating implicite

A défaut d'avoir directement les avis des utilisateurs, nous avons établis une notation basée sur la répétition des interactions d'un utilisateur sur un même article:

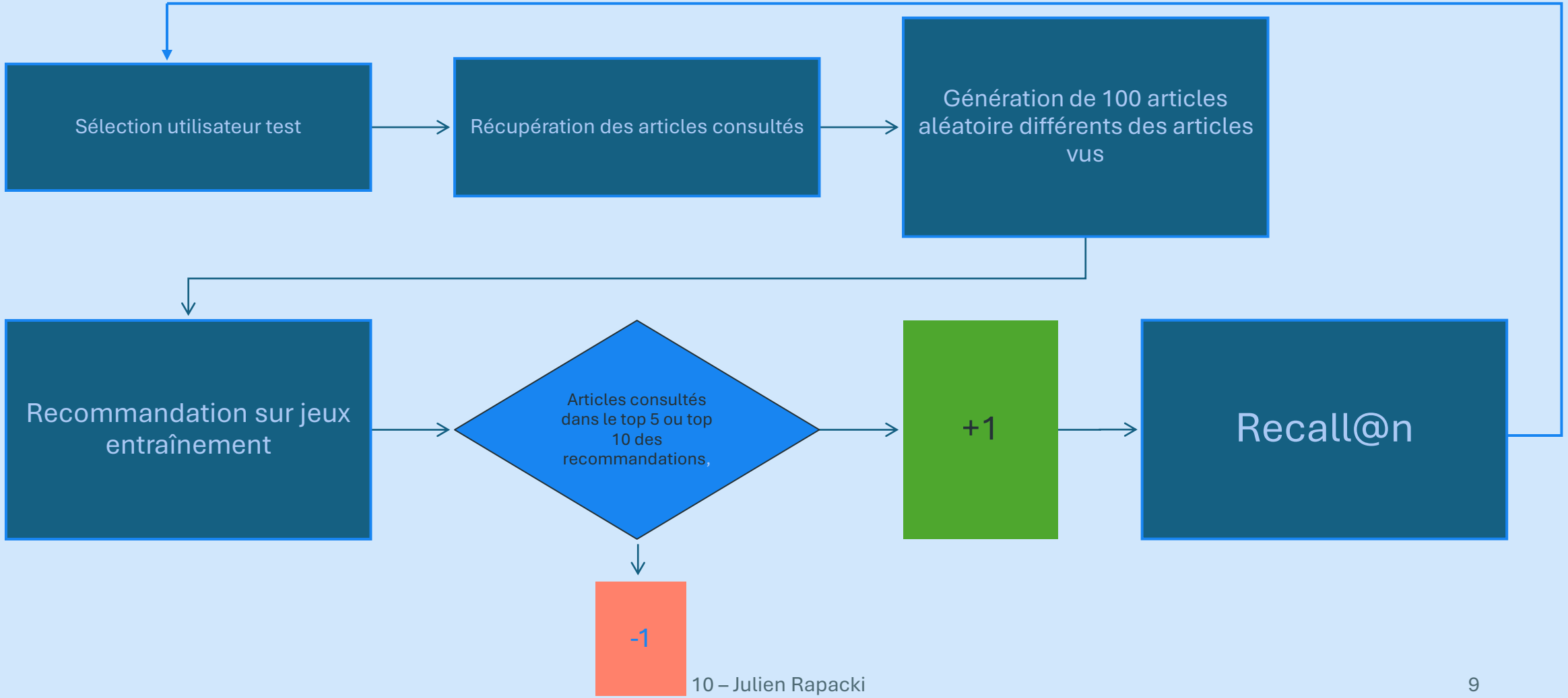
user_id	click_article_id	count_clicks_by_articles_by_user
16280	68851	33
	50864	1
	62764	3
	64329	1
	38823	30
2520	237807	17
33937	225378	16
	96173	16
188046	69463	13
10188	73431	13
26751	105941	13
2520	158046	13
14073	166581	13
2546	146230	13
14073	199198	12

user_id	article_id	rating
16280	68851	0.112628

user_id	article_id	rating
16280	62764	0.010239

Modèles développés

Principe d'évaluation (parmi d'autres)



Modèles développés

Filtrage collaboratif

	model	Precision@k	MAP@k	nDCG@k	train_time
0	AlternatingLeastSquares	0.09990	0.06261	0.09064	77.11195
1	BayesianPersonalizedRanking	0.12497	0.08582	0.12047	171.08927
2	LogisticMatrixFactorization	0.03090	0.01096	0.01981	88.18852

Algo	MAP	nDCG@k	Precision@k	Recall@k
ALS	0.004732	0.044239	0.048462	0.017796
BiVAE	0.146126	0.475077	0.411771	0.219145
BPR	0.132478	0.441997	0.388229	0.212522
FastAI	0.025503	0.147866	0.130329	0.053824
LightGCN	0.088526	0.419846	0.379626	0.144336
NCF	0.107720	0.396118	0.347296	0.180775
SAR	0.110591	0.382461	0.330753	0.176385
SVD	0.012873	0.095930	0.091198	0.032783

https://github.com/recommenders-team/recommenders

Modèles développés

Filtrage collaboratif

Pas besoin de connaissances préalables sur le contenu : Le filtrage collaboratif peut fonctionner **sans informations explicites sur le contenu** des éléments, ce qui le rend applicable à une large gamme de domaines.

Découverte de relations complexes : Le filtrage collaboratif peut découvrir des relations complexes et non linéaires entre les utilisateurs et les éléments, ce qui permet de faire des recommandations **plus précises** et plus **surprenantes**.

Problème de **démarrage à froid** : Le filtrage collaboratif a du mal à faire des recommandations pour les nouveaux utilisateurs ou les nouveaux éléments, car il n'y a pas suffisamment de données d'interaction disponibles.

Sparsité des données : La matrice utilisateur-élément est souvent très clairsemée, ce qui peut affecter la précision des recommandations.

Scalabilité : Le filtrage collaboratif peut être **difficile à mettre en œuvre** à grande échelle, car il nécessite de calculer les similarités entre tous les utilisateurs ou tous les éléments.

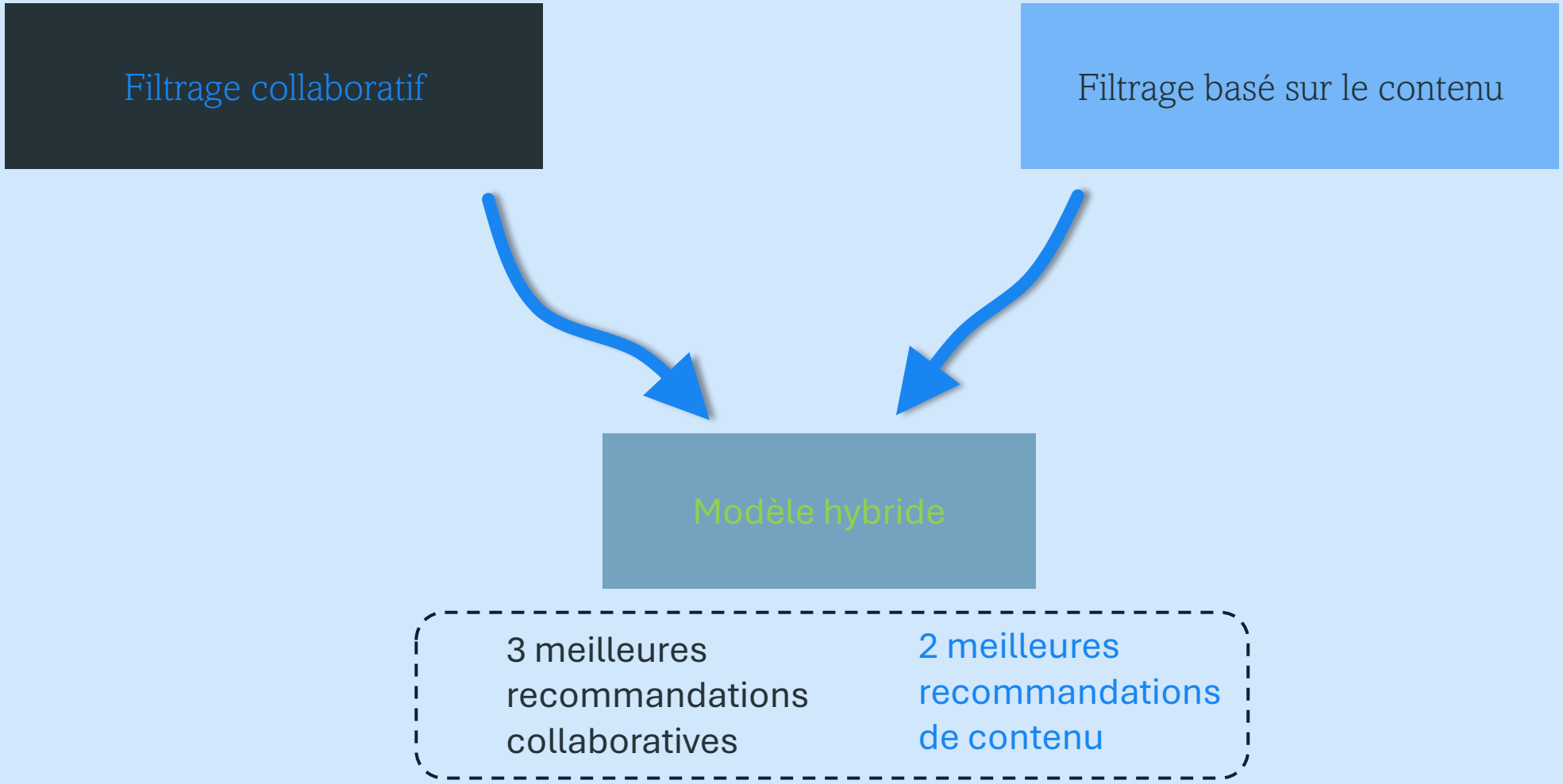
Filtrage basé sur le contenu

Pas de problème de démarrage à froid : Le filtrage basé sur le contenu peut faire des **recommandations pour les nouveaux éléments**, car il ne dépend pas des données d'interaction des utilisateurs.
Interprétabilité : Les recommandations basées sur le contenu sont **faciles à interpréter**, car elles sont basées sur les caractéristiques des éléments.

Dépendance des caractéristiques : Le filtrage basé sur le contenu **dépend fortement de la qualité et de la disponibilité des données de caractéristiques**

Manque de surprise : Le filtrage basé sur le contenu peut avoir tendance à recommander des éléments très similaires à ceux que l'utilisateur a déjà aimés, ce qui peut limiter la découverte de nouveaux contenus

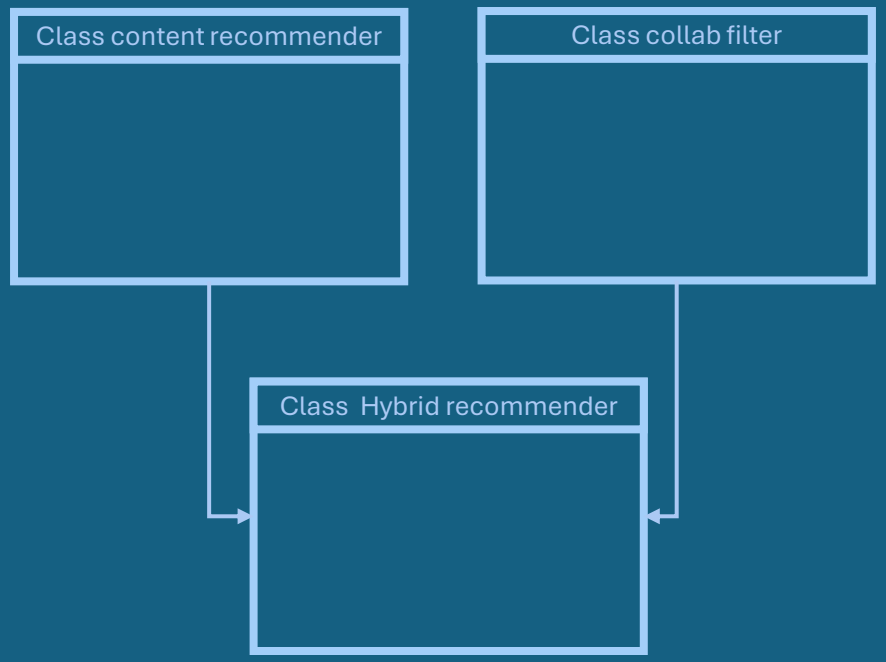
Modèles développés



Architecture retenue

Conception modèle

Utilisation d'un notebook colab pour l'élaboration du modèle

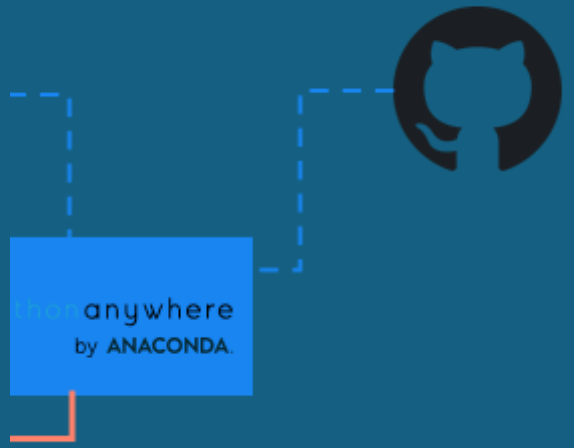



```python
class ContentBasedRecommender:

    MODEL_NAME = 'Content-Based'

    def __init__(self, article_df, interaction_df, article_embeddings):
        self.article_df = article_df
        self.interaction_df = interaction_df
        self.article_embeddings = article_embeddings
        self.item_ids = article_df['article_id'].tolist()

    def get_model_name(self):
        return self.MODEL_NAME

    def _get_similar_items_to_user_profile(self,person_id, topn=5000000):
        # Listing des artciles vus par l'utilisateur
        user_items = self.interaction_df[self.interaction_df.index == person_id]['click_article_id'].to_list()
        if not user_items:
            return []
        # Création d'un profil utilisateur avec les embeddings des articles consultés
        user_profile = np.mean([self.article_embeddings[self.item_ids.index(item)] for item in user_items if item in self.item_ids], axis=0)


        # Calcul de la similarité entre le profil utilisateur est tous les articles disponibles
        similarities = cosine_similarity([user_profile], self.article_embeddings)[0]

        # Articles similaires au profil utilisateur en excluant la liste des articles déjà vus
        similar_indices = similarities.argsort()[::-1]
        similar_items = [(self.item_ids[i], similarities[i]) for i in similar_indices if self.item_ids[i] not in user_items]

        return similar_items[:topn]
```

Architecture retenue

Stockage / mise à disposition des sources

Le modèle est alimenté par une Github release qui stocke les fichiers suivants

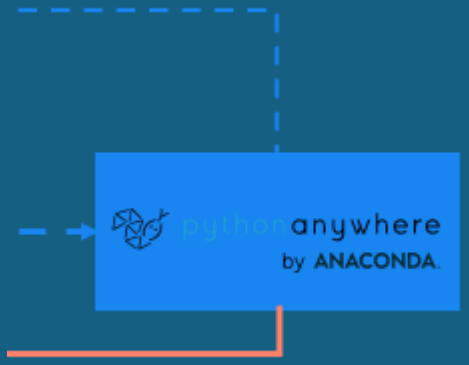


- df_articles.pkl
- df_clicks.pkl
- Articles_embeddings.pickle → Matrice des caractéristiques articles
- Csr_user_item.pkl → Matrice des interactions utilisateurs-articles
- collab_model_weights.pkl → Poids du modèle collaboratif entraîné

Architecture retenue

API

Le modèle est déployé sur Python anywhere d'où il pourra générer les recommandations appelée par la fonction Azure



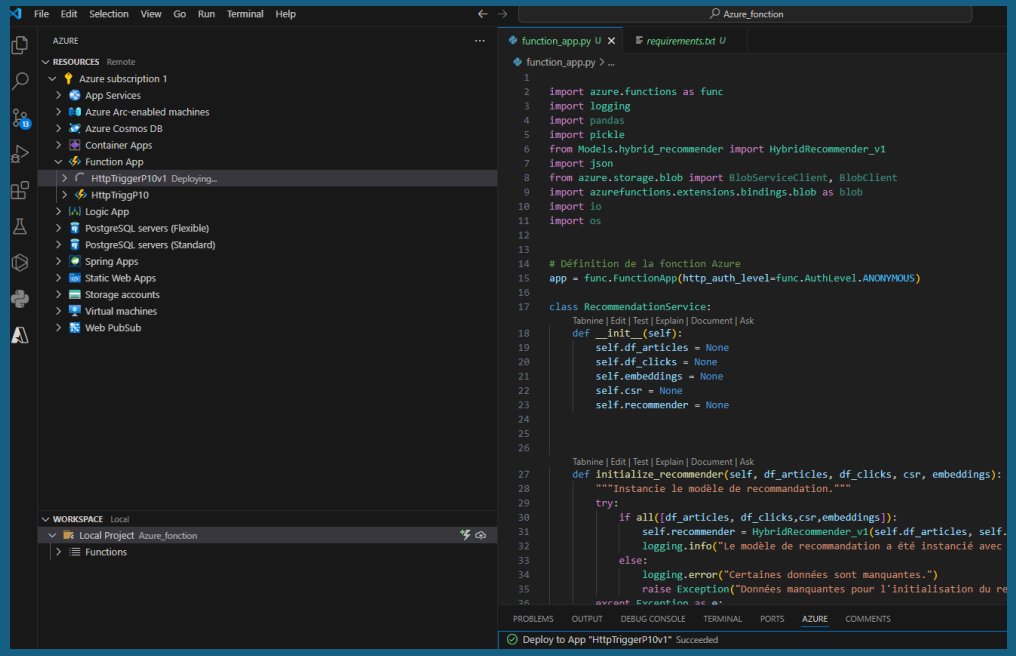
https://julienrapacki.pythonanywhere.com/

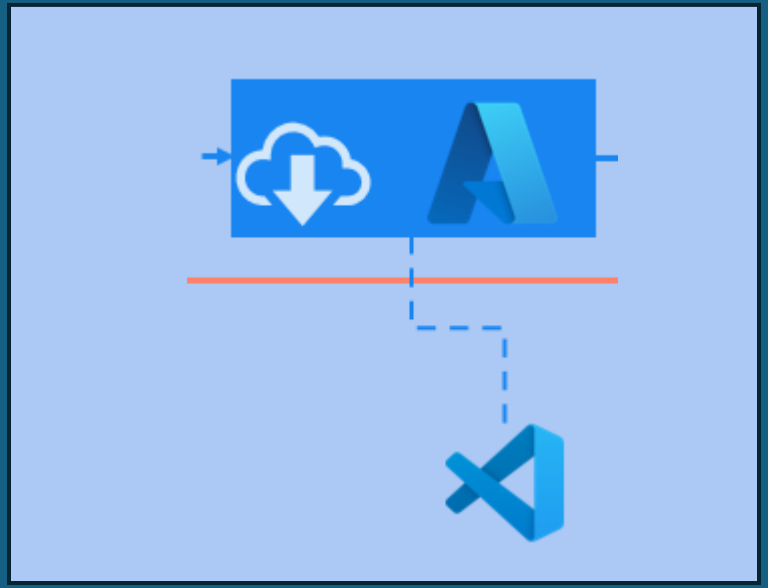
- App.py
- collaborative_filtering_recommender.py
- content_based_recommender.py
- hybrid_recommender.py

Architecture retenue

Fonction déclenchement HTTP

Une fonction http-trigger est déployée sur Azure

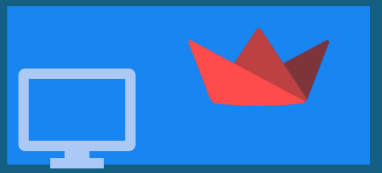




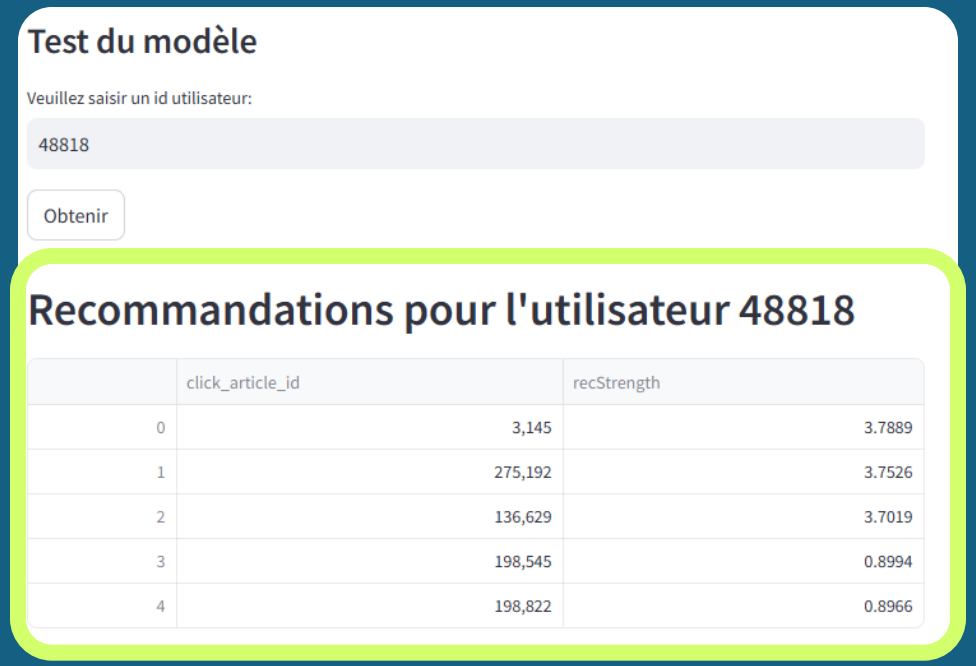
Architecture retenue

Interface de test

La saisie de l'utilisateur se fait via l'interface Streamlit

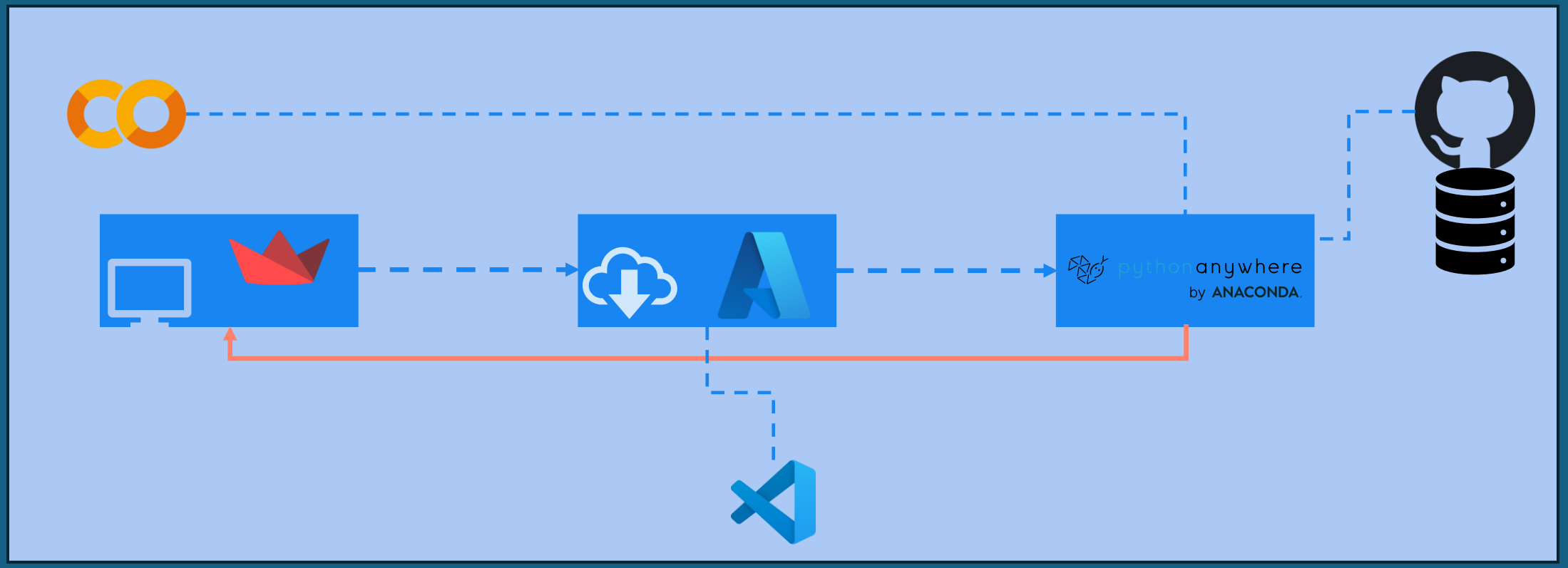


https://p10front-ia-openclassrooms.streamlit.app/



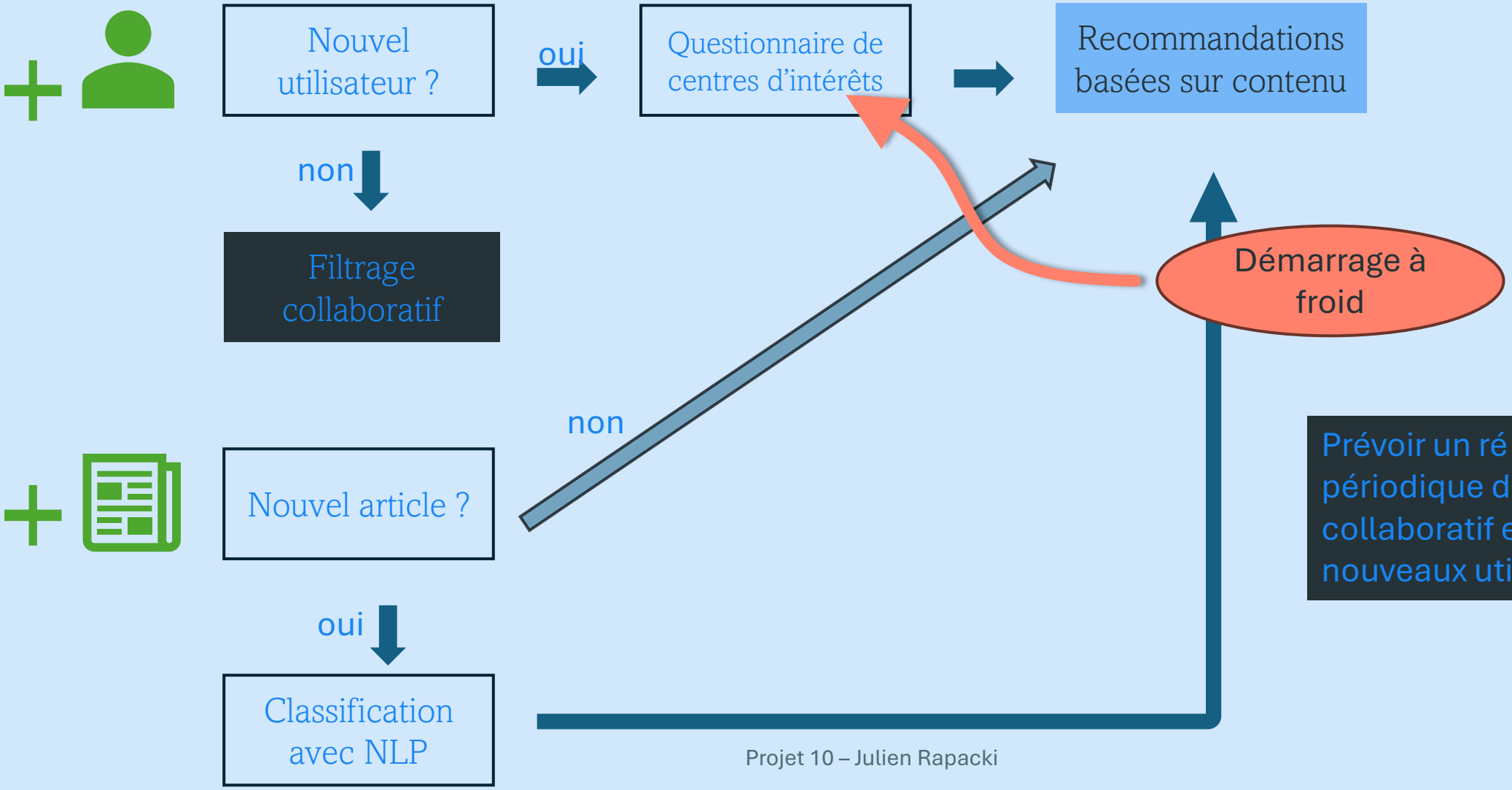
	click_article_id	recStrength
0	3,145	3.7889
1	275,192	3.7526
2	136,629	3.7019
3	198,545	0.8994
4	198,822	0.8966

Architecture retenue



L'analyse de similarité de contenu est faite à la volée

Architectures cibles



Prévoir un ré entraînement périodique du model collaboratif en fonction des nouveaux utilisateurs

Conclusion

Nous avons pu mettre en place et démontrer la viabilité du modèle de recommandation.

Il y a néanmoins quelques points de vigilance vis-à-vis du jeu de données:

-beaucoup de catégories -> plus de 400.

-manque d'information de contexte du fait de l'aspect « brut » des données (pas de noms de catégories)

Le déploiement quant à lui est fonctionnel et évolutif. L'utilisation d'un mode de recommandation hybride pourra répondre au besoin d'ajout d'utilisateurs et articles en évitant notamment le problème de démarrage à froid.