Note Technique

Frames Dataset

Méthodologie pour le pilotage de la performance

Projet ChatBot de FlyMe

Parcours Ingénieur IA chez Openclassrooms
Projet 10

Nicolas Blanchon

I. Introduction

A. Présentation du projet

Fly Me est une agence qui propose des voyages clé en main pour les particuliers ou les professionnels.
Fly Me a lancé un projet ambitieux de développement d'un **chatbot** pour **aider les utilisateurs à choisir une offre de voyage**.

La première étape de ce projet est de **construire un MVP** qui aidera les employés de Fly Me à réserver facilement un billet d'avion pour leurs vacances.

La V1 du Chatbot devra identifier dans la demande de l'utilisateur les cinq éléments suivants:

1. Ville de départ
2. Ville de destination
3. Date aller souhaitée du vol
4. Date retour souhaitée du vol
5. Budget maximum pour le prix total des billets

Si un des éléments est manquant, le chatbot devra pouvoir poser les questions pertinentes (en anglais) à l'utilisateur pour comprendre complètement sa demande. Lorsque le chatbot pense avoir compris tous les éléments de la demande de l'utilisateur, il doit pouvoir reformuler la demande de l'utilisateur et lui demander de valider sa compréhension.

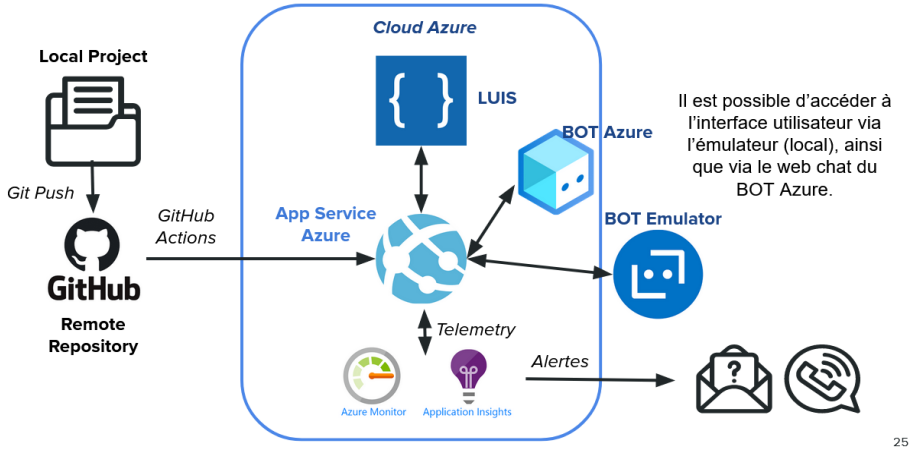
B. Pilotage de la performance

Ce document a pour objectif de clarifier la méthodologie pour le suivi de la performance du modèle et sa mise à jour en production.

Pour ce faire, nous séparons ce document en 3 axes:

- Evaluation de la performance
- Pilotage du service en production
- Mise à jour du modèle en production

C. Stack technique



D. Préparation du jeu de donnée

Afin de mesurer la performance du modèle LUIS, nous avons séparé notre jeu de donnée en 2 fichiers JSON pour:

1. Entrainement du modèle LUIS
2. Validation des performance du modèle LUIS

II. Méthodologie pour le pilotage de la performance

Dans le cadre du projet, l'utilisation des services LUIS requiert une attention particulière dans sa gestion de la performance (= compréhension du besoin utilisateur).

A ce titre, il est primordial de définir les bonnes pratiques nécessaires à son bon fonctionnement, tel que la mise à jour régulière du modèle d'IA conversationnelle qui applique une intelligence NLP personnalisée via LUIS.

A. Evaluation de la performance

L'évaluation de la performance du modèle peut se faire via 2 options:

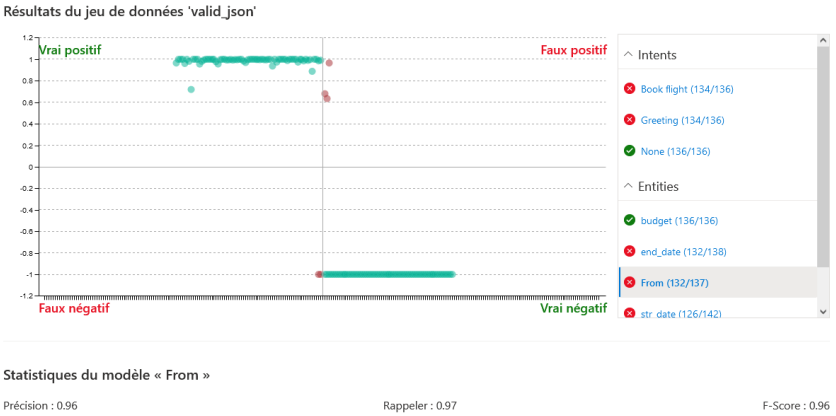
- Via l'interface LUIS: https://www.luis.ai/
- Via le script de test luis_testing.py

Nous obtenons dans les 2 cas le mesures suivantes:

1. Précision des entités et intentions
2. Recall des entités et intentions
3. F-Score des entités et intentions

Nous privilégions l'utilisation de test par lots (batch testing), car ils permettent de mesurer la performance sur un jeu de validation important.

Voici ci-dessous un extrait du résultat obtenu via l'interface LUIS (Entités From):



Nous observons ici la matrice de confusion associée à la détection de l'entité "From", ainsi que les scores de Précision, Rappel et F-Score.

Nous obtenons les résultats identiques via l'utilisation du script luis_testing.py, qui utilisera également le fichier valid.json pour le test par lot.

Note: l'appel à la méthode *client.examples.batch* du SDK azure nécessite de découper la taille de chaque batch en moins de 100 utterances pour utiliser l'API LUIS (cf: Article API ici). Cette action est intégrée au script, et donc transparente pour l'opérationnel en charge du test de performance.

A la fin du processus de test via le script, nous obtenons:

- Dans le dossier results au même niveau que le script python, un fichier de résultat avec tout le détail: luis_batch_testing_results.json
- Dans le terminal de commande, un résumé des performances (voir capture d'écran ci-dessous):

```
Json loading process completed

--> Intent Overall results:
{'modelName': 'Book flight', 'modelType': 'Intent Classifier', 'precision': 0.99, 'recall': 1.0, 'fScore': 1.0}
{'modelName': 'Greeting', 'modelType': 'Intent Classifier', 'precision': 1.0, 'recall': 0.5, 'fScore': 0.67}
{'modelName': 'None', 'modelType': 'Intent Classifier', 'precision': 'NaN', 'recall': 'NaN', 'fScore': 'NaN'}
--> Entities Overall results:
{'modelName': 'budget', 'modelType': 'Entity Extractor', 'precision': 1.0, 'recall': 1.0, 'fScore': 1.0}
{'modelName': 'end_date', 'modelType': 'Entity Extractor', 'precision': 0.85, 'recall': 0.97, 'fScore': 0.91}
{'modelName': 'From', 'modelType': 'Entity Extractor', 'precision': 0.97, 'recall': 0.96, 'fScore': 0.97}
{'modelName': 'str_date', 'modelType': 'Entity Extractor', 'precision': 0.73, 'recall': 0.86, 'fScore': 0.79}
{'modelName': 'To', 'modelType': 'Entity Extractor', 'precision': 0.94, 'recall': 0.98, 'fScore': 0.96}
```

A titre informatif, nous avons également traité l'information contenu dans le fichier de résultat au travers de notre notebook EDA.ipynb (voir fin du notebook)

```
    intent_detection_efficiency = 100 - performance_luis_model_u
    print(f"Le taux d'intention correctement detectée par le mod
165]

··  Le taux d'intention correctement detectée est de = 99.26 %
```

B. Pilotage du service en production

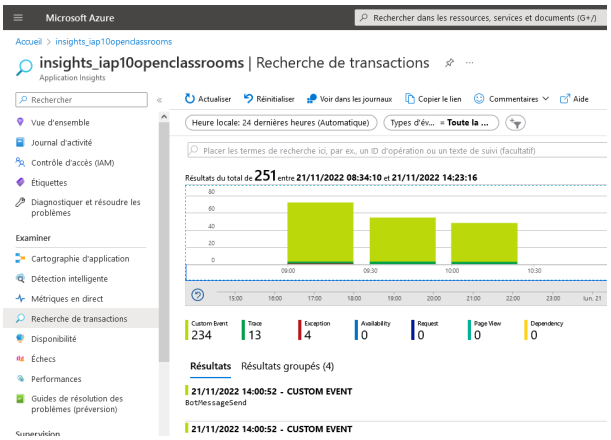
Nous analysons l'utilisation du service en production, au travers des services Microsoft d'Application insights (Azure Monitor).

Pour ce faire, nous transmettons un ensemble d'informations depuis notre code python hebergé sous Service App Azure, afin de:

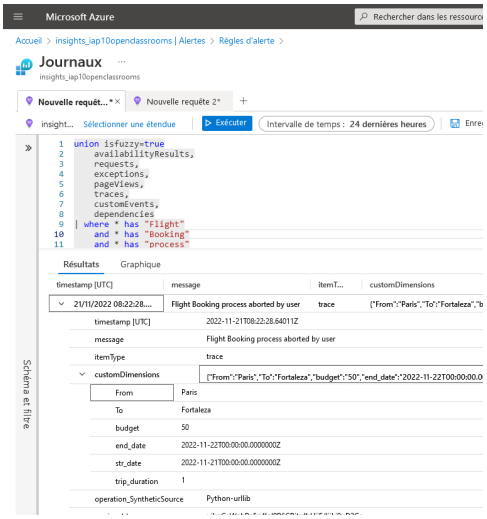
- Comprendre et mesure le fonctionnement de l'application (Nombre de requete, Temps de réponse, Disponibilité).
- Passez en revue les données d'exécution de l'application (Décision utilisateur, Incident, transactions utilisateur/BOT).

Dans notre cas, Application Insights fournit différents types de données pour la télémétrie, tels que les traces et le évenements, à partir desquels nous pouvons créer des alertes. Voici ci-dessous quelques exemples de pages qui nous permettent de piloter la performance via Azure Application Insights:

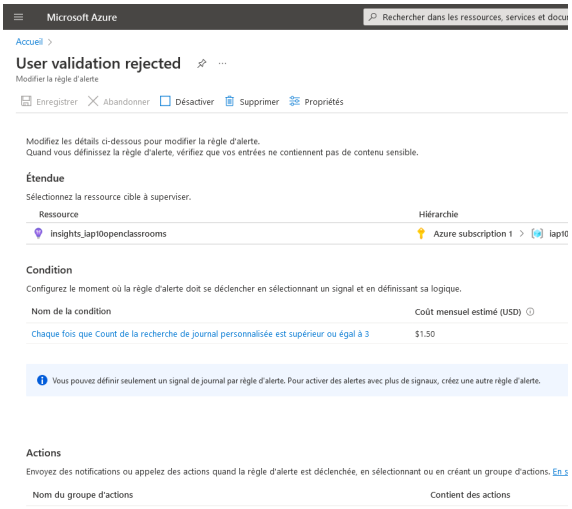
➔ Recherche de Transaction: Elément de télémétrie pour chaque demande

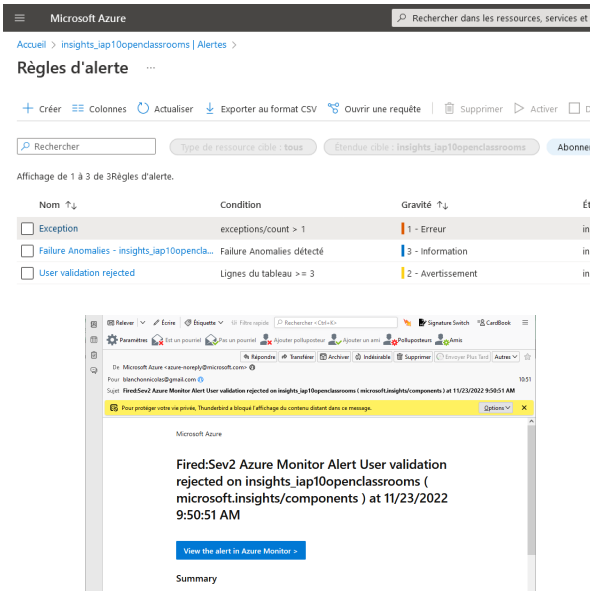


➔ Journaux: Information liées aux transactions, accessible sous form de Tables



➔ Alertes: Configuration d'alertes sur évenements critiques (Déclenchement de Mail)





C. Mise à jour du modèle en production

Il est primordial d'assurer un service de réservation de vol évolutif, permettant de prendre en considération de possible nouvelles intentions (Reservation d'hotel, de voiture, …) et entités (Nombre d'enfant, d'adultes, …).
Nous préconisons donc de mettre à jour le modèle LUIS de façon régulière.

A ce titre, nous pouvons fournir les recommandations ci-dessous:

1. Collecte des échanges utilisateurs, pour l'enrichissement futur de notre jeu de donnée d'entrainement
2. Labellisation du texte, avec identification des intentions et entités
3. Cloner le modèle, avec incrémentation de la version (voir script luis_cloning.py)
4. Ré-entrainement du modèle, à partir du jeu de donnée enrichi
5. Test du modèle, et comparaison avec les résultat de la version précédente
6. Publication si résultats positifs
7. Monitoring en production du taux de conversion utilisateur (=satisfaction)

Ces étapes devront être integrés dans les activités opérationnelles de l'équipe, et pourront être partiellement automatisées par l'ajout d'une base de donnée pour la collecte des dialogues et par le lancement des scripts de clonage/test sur une fréquence determinée.