PROJET MLE

Damien Le Dirach – Août 2021

Présentation

I. Introduction
II. Modélisation/Notebook
III. API
IV. Test de l'API
V. Docker
VI. Kubernetes
VII. Conclusion

I - Introduction

1) **Problématique**
   - Mettre en production un modèle d'analyse de sentiment
   - Les modèles ne sont pas ré-entraînés en production
   - Développer une API qui permet:
     - ✓ D'interroger les modèles
     - ✓ D'accéder aux performances des modèles
     - ✓ D'authentifier les utilisateurs

2) Présentation du jeu de données
3) Environnement de développement

I - Introduction

1) Problématique

2) Présentation du jeu de données

- Kaggle
- Le jeu de données inclut 42,000 avis sur 3 parcs Disneyland (Paris, Californie et Hong Kong)
- Colonnes:
  - ✓ Review_ID: unique id given to each review
  - ✓ **Rating**: ranging from 1 (unsatisfied) to 5 (satisfied) => **target**
  - ✓ Year_Month: when the reviewer visited the theme park
  - ✓ Reviewer_Location: country of origin of visitor
  - ✓ **Review_Text**: comments made by visitor => **feature**
  - ✓ Disneyland_Branch: location of Disneyland Park

3) Environnement de développement

I - Introduction

1) Problématique
2) Présentation du jeu de données

3) Environnement de développement

- Expérience Linux avec les mini projets
- Transposition dans un environnement Windows avec l'IDE **Visual Studio Code**:
  - ✓ intelliSense (code complétion..)
  - ✓ Débogage (pas à pas, points d'arrêt..)
  - ✓ Intégration de Docker (avec Docker Desktop)
  - ✓ Intégration du Cloud azure
  - ✓ Nombreuses extensions (snippets…)

II – Modélisation / Notebook

1) **Notebook Fourni**
   - Données partitionnées en jeu d'entraînement / test
   - **CountVectorizer**
     - Entraîné/appliqué sur jeu d'entraînement
     - À appliquer sur les jeux de test (i.e. colonne « Review_Text »)
     - Représentation du texte sous forme de vecteur
   - 4 **Modèles**
     - LogisticRegression() appliquée sur l'ensemble des données d'entraînement
     - RandomForestClassifier(n_estimators=20, max_depth=5) appliqué respectivement sur les parcs HK/Californie/Paris

2) Adaptation du notebook à la problématique

II – Modélisation / Notebook

1) Notebook Fourni

2) Adaptation du notebook à la problématique

- 2 fichiers pickles générés pour chaque modèle au sein de l'API:
  - **CountVectorizer**
  - **Modèles pré-entraînés**
- 2 autres fichiers globaux:
  - NLTKWordTokenizer.pkl
  - stopwords.pkl

III – API

1) Endpoints

"/" : retourne {score : 1} si l'API fonctionne

"/get_username" : retourne le username actuellement utilisé [authentification*]

"/get_performance" : retourne le score du modèle

"/text_to_sentiment/" : retourne la prédiction (score de 1 à 5) associé au texte saisi [authentification*]

2) Classe ModelFromFiles

- **Charger un modèle** (parmi les 4 existants) à partir des fichiers pré-enregistrés « count_vectorizer{i}.pkl » et « model{i}.pkl »

  ⇨ `def _load_from_pickles_files(self)`

- **Pré-processer un texte** afin de le rendre exploitable par le modèle

  ⇨ `def preprocess(self, text, pkl_stopwords, pkl_tokenizer)`

- **Effectuer une prédiction** à partir d'un texte

  ⇨ `def predict(self, text, pkl_stopwords, pkl_tokenizer)`

3) Utilisation d'un Type Enum

```
class EnumModel(IntEnum):
    AllBranch = 1
    HK = 2
    California = 3
    Paris = 4
```

IV – Test de l'API

1) Test manuel [FastAPI - Swagger UI (disneyreviews.azurewebsites.net)]

2) Description du script de test 

3) Tests effectués

```
#test du endpoint « / » => [code HTTP attendu = 200, score attendu = 1]

    '####': (200, '1')

#test => bad username, 401 attendu
    'text_to_sentiment#{sentence}#1#alice1#wonderland'.format(sentence=lst_senten
ces[0]): (401, '')
```

V – Docker

1) Docker de l'API 

2) Docker du script de test 

3) Docker Compose 


```
==============================
API test
==============================
Request done at "/text_to_sentiment"
| sentence=Visited 21 5 2014. This park is a joke, three main rides were closed in one park...
| model_index=1
| username="alice"
| password="wonderland"
=> Test(expected vs actual) / HTTP Status: 200 vs 200 / Score: [1] vs [1] ==> success
```

VI – Kubernetes

1) Deployment 
2) Service 
3) Ingress 

VII – Conclusion

- https://disneyreviews.azurewebsites.net/docs#/

- Perspectives :
  - Workflow GitHub Actions
  - Conteneur avec BDD MySQL pour username/password
  - Déploiement Kubernetes Azure
  - Logging

- Questions/Remarques?