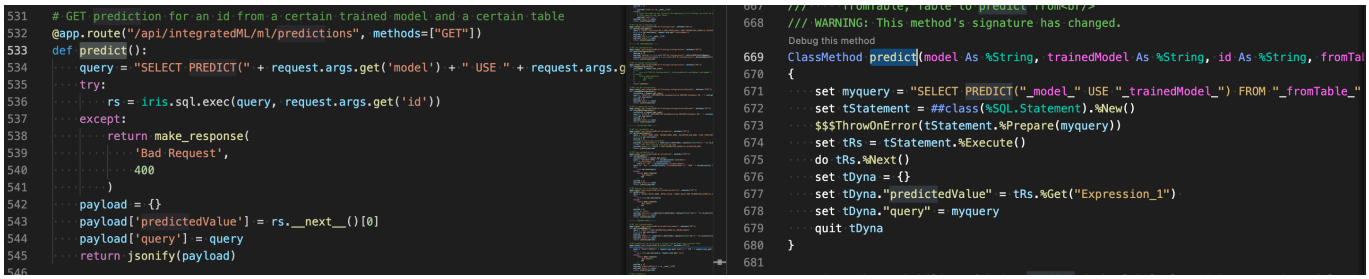
Annonce
Guillaume Rongier · Mars 16, 2022

Open Exchange

Démonstration complète d'IntegratedML et de Embedded Python

1. Démonstration d'IntegratedML

Ce dépôt est une démonstration d'IntegratedML et d'Embedded Python.



2. Construction de la démo

Pour construire la démo, il suffit d'exécuter la commande :

```
docker compose up
```

2.1. Architecture

Deux conteneurs seront construits : un avec IRIS et un avec un serveur nginx.

L'image IRIS utilisée contient Python embarqué. Après la construction, le conteneur exécutera un serveur wsgi avec l'API Flask.

Nous utilisons le paquet communautaire csvgen pour importer le jeu de données titanic dans iris. Pour le jeu de données noshow, nous utilisons une autre méthode personnalisée (la classmethod Load() de la classe Util.Loader). Pour que le conteneur ait accès aux fichiers csv, nous lions le répertoire local iris/ au répertoire /opt/irisapp/ dans le conteneur.

2.2. Construction du conteneur nginx

Afin de construire notre conteneur nginx, docker utilise une construction en plusieurs étapes. Tout d'abord, il crée un conteneur avec node. Il installe ensuite npm et copie tous nos fichiers dans ce conteneur. Il construit le projet avec la commande ng build, et le fichier de sortie est copié dans un nouveau conteneur qui ne contient que nginx.

Grâce à cette manœuvre, nous obtenons un conteneur très léger qui ne contient pas toutes les bibliothèques et outils nécessaires à la construction de la page Web.

Vous pouvez vérifier les détails de ce multi-build dans le fichier angular/Dockerfile. Nous avons également configuré les paramètres de notre serveur nginx grâce au fichier angular/nginx.default.conf.

3. Exécution de la démo

Il suffit d'aller à l'adresse : http://localhost:8080/ et c'est tout ! Profitez-en !

4. Backend Python

Le back-end est réalisé avec Python Flask. Nous utilisons Python embarqué afin d'appeler les classes iris et d'exécuter des requêtes depuis Python.

4.1. Python embarqué

4.1.1. Configuration du conteneur

Dans le dockerfile, nous devons d'abord expliciter deux variables d'environnement que Python embarqué utilisera :

```
ENV IRISUSERNAME "SuperUser"
ENV IRISPASSWORD $IRIS_PASSWORD
```

Avec $IRISPASSWORD configuré comme ceci dans le fichier docker-compose :

```
iris:
  build:
    args:
      - IRIS_PASSWORD=${IRIS_PASSWORD:-SYS}
```

(Le mot de passe transféré est celui configuré sur votre machine locale ou -s'il n'est pas configuré- sera par défaut « SYS »)

4.1.2. Utiliser Python embarqué

Afin d'utiliser Python embarqué, nous utilisons irispython comme intercepteur de python, et faisons :

```
import iris
```

Tout au début du fichier.

Nous serons alors en mesure d'exécuter des méthodes telles que :

Comme vous pouvez le voir, pour OBTENIR un passager avec un ID, nous exécutons simplement une requête et utilisons son jeu de résultats.

Nous pouvons également utiliser directement les objets IRIS :

Ici, nous utilisons une requête SQL pour obtenir tous les ID du tableau, puis nous récupérons chaque passager du tableau avec la méthode %OpenId() de la classe Titanic.Table.Passenger (notez que % étant un caractère illégal en Python, nous utilisons _à la place).

Grâce à Flask, nous implémentons toutes nos routes et méthodes de cette façon.

4.1.3. Comparaison côte à côte

Sur cette capture d'écran, vous avez une comparaison côte à côte entre une implémentation **Flask** et une implémentation **ObjectScript**.

Comme vous pouvez le voir, il y a beaucoup de similitudes.

4.2. Démarrage du serveur

Pour lancer le serveur, nous utilisons gunicorn avec irispython.

Dans le fichier docker-compose, nous ajoutons la ligne suivante :

```
iris:
  command: -a "sh /opt/irisapp/flask_server_start.sh"
```

Cela lancera, après le démarrage du conteneur (grâce à l'indicateur -a), le script suivant :

```
#!/bin/bash

cd ${FLASK_PATH}

${PYTHON_PATH} /usr/irissys/bin/gunicorn --bind "0.0.0.0:8080" wsgi:app -w 4 2>&1

exit 1
```

Avec les variables d'environnement définies dans le dockerfile comme suit :

```
ENV PYTHON_PATH=/usr/irissys/bin/irispython
```

```
ENV FLASK_PATH=/opt/irisapp/python/flask
```

Nous aurons alors accès au back-end Flask via le port local 4040, puisque nous y avons lié le port 8080 du conteneur.

5. IntegratedML

5.1. Explorer les deux ensembles de données

Pour les deux ensembles de données, vous aurez accès à un CRUD complet, vous permettant de modifier à volonté les tableaux enregistrés.

Afin de passer d'un ensemble de données à l'autre, vous pouvez appuyer sur le bouton en haut à droite.

5.2. Managing models

5.2.1. Création d ' un modèle

Une fois que vous avez découvert les données, vous pouvez créer un modèle prédisant la valeur que vous souhaitez.

En cliquant dans le menu de navigation latéral Gestionnaire de modèles, dans la liste des modèles, vous aurez accès à la page suivante (ici dans le cas de l'ensemble de données NoShow) :

Vous pouvez choisir la valeur que vous voulez prédire, le nom de votre modèle, et avec quelles variables vous voulez prédire.

Dans le menu latéral, vous pouvez activer Voir les requêtes SQL ? pour voir comment les modèles sont gérés dans IRIS.

Après avoir créé un modèle, vous devriez voir ceci :

Comme vous pouvez le voir, la création d'un modèle ne prend qu'une seule requête SQL. Les informations que vous avez sont toutes les informations que vous pouvez récupérer d'IRIS.

Dans la colonne actions, vous pouvez supprimer un modèle ou le purger. La purge d'un modèle supprimera tous ses cycles de formation (et leurs cycles de validation), à l'exception du dernier.

5.2.2. Entraînement d'u modèle

Dans l'onglet suivant, vous pourrez entraîner vos modèles.

Vous avez le choix entre 3 fournisseurs. AutoML d'InterSystems, H2O, une solution open-source, et DataRobot, dont vous pouvez avoir un essai gratuit de 14 jours si vous vous enregistrez sur leur site.

Vous pouvez sélectionner le pourcentage de l'ensemble de données que vous souhaitez utiliser pour entraîner votre modèle. Étant donné que l'entraînement de grands ensembles de données peut prendre beaucoup de temps, il est possible, pour les besoins des démonstrations, de prendre un ensemble de données plus petit.

Ici, nous avons entraîné un modèle en utilisant l'ensemble des données Titanic :

Le bouton dans la colonne actions vous permettra de voir le log. Pour l'AutoML, vous verrez ce que l'algorithme a réellement fait : comment il a préparé les données et comment il a choisi le modèle à utiliser.

L'entraînement d'un modèle ne nécessite qu'une seule requête SQL, comme vous pouvez le voir dans la section

des messages du menu de navigation latéral.

Gardez à l'esprit que dans ces deux onglets, vous ne verrez que les modèles qui concernent l'ensemble de données que vous utilisez réellement.

5.2.3. Validation d'un modèle

Enfin, vous pouvez valider un modèle dans le dernier onglet. En cliquant sur une exécution de validation, une boîte de dialogue s'ouvre avec les métriques associées à la validation. Là encore, vous pouvez choisir un pourcentage de l'ensemble de données à utiliser pour la validation.

Une fois de plus, il suffit d'une seule requête SQL.

5.2.4. Effectuer des prédictions

Dans le menu Faire des prédictions, dernier onglet, vous pouvez faire des prédictions en utilisant vos modèles nouvellement formés.

Il vous suffit de rechercher un passager / patient et de le sélectionner, de choisir l'un des modèles entraînés et d'appuyer sur prédire.

Dans le cas d'un modèle de classification (comme dans cet exemple, pour prédire la survie), la prédiction sera associée à la probabilité d'être dans la classe prédite.

Dans le cas de Mme Fatima Masselmani, le modèle a correctement prédit qu'elle a survécu, avec une probabilité de 73 %. Juste en dessous de cette prédiction, vous pouvez voir les données utilisées par le modèle :

Une fois encore, il faut une requête pour obtenir la prédiction et une pour la probabilité.

6. Utilisation de COS

La démonstration fournit en fait deux API. Nous utilisons l'API Flask avec Python embarqué, mais un service REST dans COS a également été configuré lors de la construction du conteneur.

En appuyant sur le bouton en haut à droite **« Passer à l'API COS »**, vous pourrez utiliser ce service.

Remarquez que rien ne change. Les deux API sont équivalentes et fonctionnent de la même manière.

7. Plus d'explicabilité avec DataRobot

Si vous voulez plus d'explicabilité (plus que ce que le journal peut vous offrir), nous vous suggérons d'utiliser le fournisseur DataRobot.

Pour cela, vous devez vous rendre à l'adresse de votre instance DataRobot, et chercher les outils de développement pour obtenir votre jeton. Lors de l'entraînement de votre modèle, la page Web vous demandera votre jeton.

Une fois l'entraînement commencé, vous pouvez accéder à votre instance DataRobot pour en savoir beaucoup plus sur votre ensemble de données et vos modèles :

Ici, nous pouvons voir que les champs sexe et nom de chaque passager sont les valeurs les plus importantes pour prédire la survie. Nous pouvons également voir que le champ tarif contient des valeurs aberrantes.

Une fois les modèles entraînés, vous pouvez avoir accès à de **nombreux** détails, en voici un aperçu :

8. Conclusion

Grâce à cette démonstration, nous avons pu voir à quel point il était facile de créer, d'entraîner et de valider un modèle ainsi que de prédire des valeurs à l'aide de très peu de requêtes SQL.

Nous avons fait cela en utilisant une API RESTful avec Python Flask, en utilisant Python embarqué, et nous avons fait une comparaison avec une API COS.

Le front-end a été réalisé avec Angular.

9. Remerciements

À Théophile, le stagiaire qui a construit cette belle démo pendant l'été 2021.

#Embedded Python #IntegratedML #Python #InterSystems IRIS
Voir l'application sur InterSystems Open Exchange