

# Projet Kedro

## Introduction:

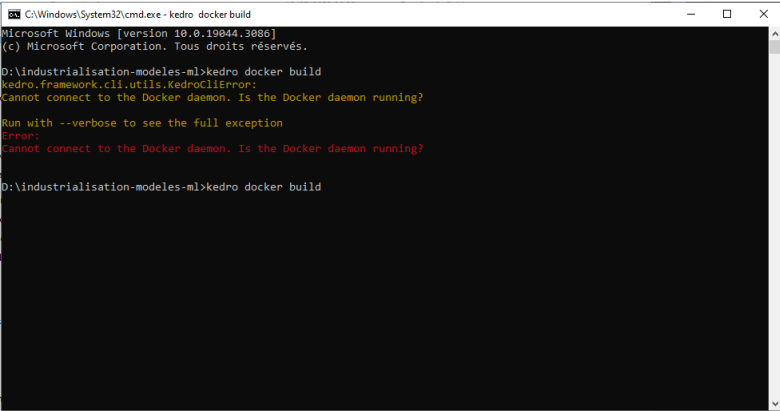
Contributeurs : Julien Florent, Pierre Chaumont, Erwan dusang

github : [https://github.com/Athroniaeth/industrialisation\\_modeles\\_ml](https://github.com/Athroniaeth/industrialisation_modeles_ml)

Présentez le projet, y compris son objectif, son public cible, les problèmes qu'il résout et pourquoi il est important.

Ceci est un projet d'école de groupe avec comme contributeur Pierre Chaumont, Julien Florent, Erwan dusang. L'objectif est la création d'un projet via Kedro pour la mise en place de l'industrialisation d'un modèle de machine learning. Le thème est l'analyse d'écoute auditive avant et après examens et de pouvoir faire des prédictions à partir de ces données.

**ATTENTION, CE PROJET N'À PAS ÉTÉ TERMINÉ À 100% POUR LA PARTIE FLASK, CELUI CI PEUT DONC CONTENIR DES BUGS. L'IMAGE DOCKER N'À PAS PU ÊTRE CRÉÉ POUR LES MÊME RAISONS**



```
C:\Windows\System32\cmd.exe - kedro docker build
Microsoft Windows [version 10.0.19044.3886]
(c) Microsoft Corporation. Tous droits réservés.

D:\industrialisation-modeles-ml>kedro docker build
kedro.framework.cli.utils.KedroCliError:
Cannot connect to the Docker daemon. Is the Docker daemon running?

Run with --verbose to see the full exception
Error:
Cannot connect to the Docker daemon. Is the Docker daemon running?

D:\industrialisation-modeles-ml>kedro docker build
```

## Prérequis:

Liste des logiciels et des connaissances nécessaires pour comprendre et utiliser votre projet.

Installer Python 3.10 - 3.9

Installer les librairies python, vous pouvez le faire en allant dans la racine du projet et en faisant "pip install -r requirements.txt"

# Installation:

Expliquez comment installer l'environnement de projet Kedro et toutes les dépendances nécessaires.

Faite la commande :

git clone [https://github.com/Athroniaeth/industrialisation\\_modeles\\_ml.git](https://github.com/Athroniaeth/industrialisation_modeles_ml.git)

en cas d'erreur, veuillez faire

- pip install mlflow kedro kedro-viz scikit-learn tensorflow
- pip install --upgrade "protobuf<=3.20.1" (à faire en cas d'erreur protobuff)

## Configuration du projet:

Montrez comment configurer le projet pour une utilisation locale. Cela peut inclure des étapes pour l'installation d'environnements virtuels, la configuration de variables d'environnement et d'autres tâches de configuration spécifiques au projet.

## Structure du projet :

Détaillez la structure du projet, y compris une description de chaque répertoire et fichier dans le projet, et leur rôle.

Les pipelines se trouvent dans le dossier src/industrialisation-modeles-ml/pipeline

Il y'a 3 pipelines dans ce projet :

- Preprocess) : Cette pipeline sert à traiter les données
  - Nettoyage des données : elle va supprimer les lignes si celle-ci contient une colonne avec une valeur vide.
  - Normalisation des données : elle va "normaliser" les données, c'est-à-dire remettre toutes les données sur la même échelle entre 0-1 avec un MinMaxScaler  $(x - \min) / (\max - \min)$ . Le maximum et minimum seront enregistré dans le fichier de configuration
  - Split des données, les données seront séparées entre entrée et sortie, et séparées sur un ratio 80/20 entre la donnée d'entraînement et la donnée de test.
- Entraînement : Cette pipeline sert à la création du modèle et de son entraînement. L'architecture du modèle est expliquée plus bas dans la partie "modèle". L'entraînement lui récupère les données d'entraînement et de test et fera l'entraînement tout en l'enregistrant dans MLFlow

**Pour des raisons pratiques et de bug, les dossier de test se trouveront dans les dossier des pipelines**

## Guide d'utilisation :

**Le projet n'a pas pu être fini à 100% pour la partie Flask**

Pour lancer l'entraînement d'un modèle dans le cas où vous n'en n'auriez pas, vous pouvez faire "kedro run", le projet enregistrera le dernier modèle créé.

Pour lancer l'application Flask, aller dans la racine du projet et faire "

.

## Modèle :

Détaillez le modèle choisi pour l'entraînement ainsi que la prédiction. Expliquez le choix de vos paramètres et de vos couches de neurone.

Les modifications apportées au modèle de base sont :

- Changement de la loss de "binary cross entropy" à "mse"
- Suppression des couches de Max Pooling qui en temps normal permet de réduire la taille des images en perdant le moins d'informations mais qui dans ce cas détruisait les informations importantes au vu de l'entrée très petite du modèle (7 entrées)
- Modifier des couches de convolution en augmentant la taille du kernel (de 2 à 4-5). Cela permet d'avoir des "images" de caractéristiques prenant en compte l'ensemble de la donnée plutôt que par petit bout. Augmentation du nombre de filtre et remplacement des couches de paddings par l'argument "padding='same' "
- Remplacement de la fonction d'activation de la dernière couche en "sigmoid" car "softmax" est une fonction d'activation pour de la classification. Car celle-ci prend en entrée les valeurs brutes (2, 1, 0, 0...) et donne un % de sortie (66%, 33%, 0%, 0%)
- D'après nos tests, mettre units à 192 (nous avons testé 126, 192, 224, 256) est la plus performante,
- D'après nos tests, mettre un taux d'apprentissage de  $1e-3$  est la plus performante (nous avons testé  $1e-2$ ,  $1e-3$ ,  $1e-4$ )
- D'après nos tests, Enlever le dropout et le kernel regularizer a eu aussi de meilleure performance, le dropout (qui désactive x % des neurones de la couche de manière aléatoire, qui est censé aider à ne pas surentraîner le modèle) ne doit pas bien marcher car celui-ci est probablement plus adapté aux gros réseaux de neurones avec une entrée de modèle plus grande avec plus de données et un apprentissage plus long.

```

# Création du modèle d'entraînement
def create_model(input_shape, units=192, activation='relu', l2_value=0.01, dropout_rate=None, learning_rate=1e-3):

    # Définition de la couche d'entrée
    inputs = layers.Input(shape=input_shape)

    # Définition des couches de convolution
    # padding='same' rajoute un "pixel" à droite et à gauche pour avoir la même taille de sortie
    x = layers.Conv1D(filters=64, kernel_size=5, padding='same', activation=activation)(inputs)
    x = layers.Conv1D(filters=128, kernel_size=4, padding='same', activation=activation)(x)

    # Aplatissement des données
    x = layers.Flatten()(x)

    # Définition des couches entièrement connectées
    x = layers.Dense(units, activation='relu')(x)

    if dropout_rate is not None:
        x = layers.Dropout(dropout_rate)(x)

    x = layers.Dense(input_shape[0], activation='sigmoid')(x)

    #Création du modèle
    model = tf.keras.Model(inputs=inputs, outputs=x)
    model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=learning_rate),
                  loss=[tf.keras.losses.MeanSquaredError()], metrics=[tf.keras.metrics.CategoricalAccuracy()])

    return model

```

## Test :

Pour les tests, en cas d'erreur si il n'y a pas d'erreur il vous faudra utiliser la commande "kedro run" pour lancer un entraînement de modèle, le projet gardera en mémoire le dernier modèle entraîné pour les tests et le fast api

Pour lancer les tests, ouvrez une invite de commande, aller dans la racine du projet, et faite "pytest"

Étapes des test effectués :

preprocess/

test\_nodes.py

Ce test a pour but de donner à la node de traitement des données, un dataframe avec une colonne vide pour voir si la node enlève bien les lignes où il y a au moins une valeur Null (None). On vérifie aussi que la normalisation s'est bien faite en ayant une valeur à 0 et une valeur à 1.

test\_quality.py

Ce test a pour but de vérifier le nombre de colonnes requises.

entrainement/

test\_model.py

Ce test a pour but de tester notre modèle et de s'assurer que la précisions du modèle est correcte via le calcul du %erreurs (root-mean-square), Le pourcentage d'erreur root-mean-square (RMSE) est une mesure d'écart entre les valeurs prédites et réelles dans un modèle. Il est calculé en prenant la

racine carrée de la moyenne des carrés des différences, et il représente l'erreur relative par rapport à la plage des valeurs réelles.

## Guide de contribution :

Si vous souhaitez contribuer à notre projet Kedro vous le pouvez. Toutes les contributions sont les bienvenues, qu'il s'agisse de corrections de bugs, d'améliorations de fonctionnalités ou de mises à jour de la documentation.

### Étapes pour contribuer

**Forker le dépôt :** Commencez par créer un fork de notre dépôt sur GitHub.

Cloner le fork : Clonez le fork sur votre machine locale pour pouvoir apporter des modifications.

**Configurer l'environnement de développement :** Installez Kedro et toutes les autres dépendances nécessaires. Reportez-vous à la section "Installation" de la documentation pour obtenir des instructions détaillées.

**Créer une branche :** Pour chaque ensemble de modifications (par exemple, une correction de bug ou une nouvelle fonctionnalité), créez une nouvelle branche dans votre fork. Cela permet de séparer clairement chaque ensemble de modifications et rend le processus d'examen plus facile pour nous.

**Apporter des modifications :** Apportez les modifications nécessaires dans votre branche. Assurez-vous de suivre les conventions de codage que nous avons définies pour ce projet.

**Exécuter les tests :** Avant de soumettre vos modifications, assurez-vous d'exécuter tous les tests pour vérifier que votre code n'a pas introduit de bugs.

**Soumettre un pull request :** Une fois que vous avez terminé vos modifications et que tous les tests passent, soumettez un pull request à notre dépôt principal. Assurez-vous de décrire clairement les modifications que vous avez apportées dans le message de votre pull request.

### Normes de codage

Nous suivons les conventions de codage de PEP 8 pour Python. Assurez-vous de suivre ces conventions lorsque vous écrivez du code pour ce projet.

**Tests :** Les tests sont essentiels pour maintenir la qualité de notre code. Chaque fois que vous soumettez une pull request, assurez-vous d'inclure des tests unitaires pour tout nouveau code que vous avez écrit. Consultez la section "Test" de la documentation pour voir comment nous organisons nos tests.

Documentation

**Documentation** : La documentation est tout aussi importante que le code lui-même. Lorsque vous ajoutez de nouvelles fonctionnalités ou faites des modifications significatives, assurez-vous de mettre à jour la documentation en conséquence.

## Dépannage :

Une section FAQ ou de dépannage peut être utile pour répondre aux questions courantes ou aux problèmes connus avec votre projet.

Si vous avez une erreur "protobuf" :

`pip install --upgrade "protobuf<=3.20.1"` (à faire en cas d'erreur protobuf)

Si vous avez une erreur du chargement du modèle via MLFlow :

Dans :

`../blob/main/src/industrialisation-modeles-ml/pipelines/entrainement/test_model.py`

Vous pouvez trouver un exemple du fichier sur ce [lien](#)

Sur la ligne 20, il faudra entrer l'ID de votre modèle en dur, celui que vous avez copié précédemment.

## Changements et historique :

Incluez un fichier "CHANGELOG" décrivant les changements importants dans chaque version du projet.

## Contacts et soutien :

Fournissez des détails sur la façon dont les utilisateurs peuvent obtenir de l'aide ou poser des questions sur votre projet.

Vous pouvez contacter les principaux contributeurs du projet :

- Pierre Chaumont (propriétaire du github)

[pierre.chaumont@hotmail.fr](mailto:pierre.chaumont@hotmail.fr)

- Erwan Duchampt:

\*\*\*\*\*

- Julien Florent :

\*\*\*\*\*