Challenge #1

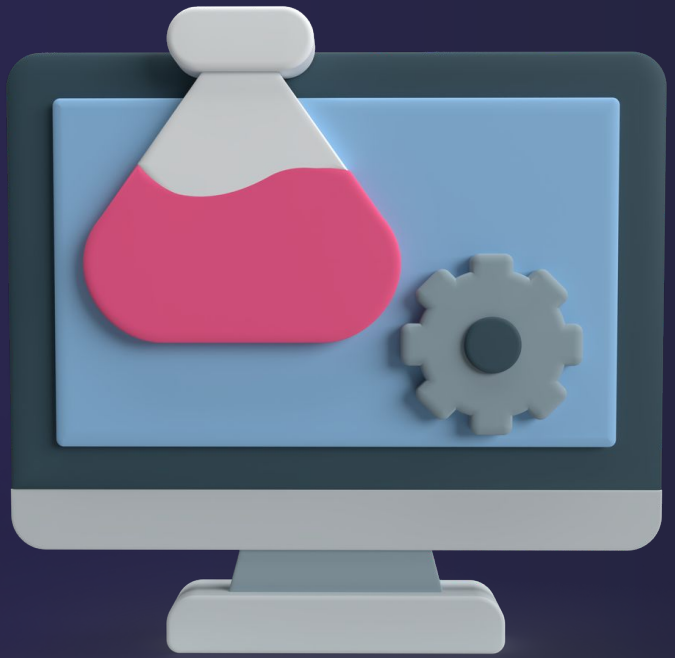
Login Biométrique

par Reconnaissance Faciale.

PARIS ynov CAMPUS

Travaux Pratiques

- **Face** ID
- **Docker** & Docker-compose



Etapes.

Commandes utiles :
$> docker compose build
$> docker compose up

- Création d'un fichier app.py
- Création d'un fichier Dockerfile
- Création d'un fichier docker-compose.yml

Fichier “app.py”

```
import face_recognition

known_image = face_recognition.load_image_file('file1.jpg')
unknown_image = face_recognition.load_image_file('file2.jpg')

biden_encoding = face_recognition.face_encodings(known_image)[0]
unknown_encoding = face_recognition.face_encodings(unknown_image)[0]

results = face_recognition.compare_faces([biden_encoding],
unknown_encoding)

print(str(results))
```

Création d’un fichier

Dockerfile

```
FROM python:3.9-slim


ENV APP_HOME /app
WORKDIR $APP_HOME


# Install production dependencies.
RUN apt-get clean && apt-get -y update && apt-get install -y
build-essential libopenblas-dev liblapack-dev libopenblas-dev
liblapack-dev
RUN pip install cmake==3.25.2
RUN pip install dlib==19.9

COPY requirements.txt .
RUN pip install -r requirements.txt

COPY . .

CMD ["python3", "app.py"]
```

Création d’un fichier

docker-compose.yml

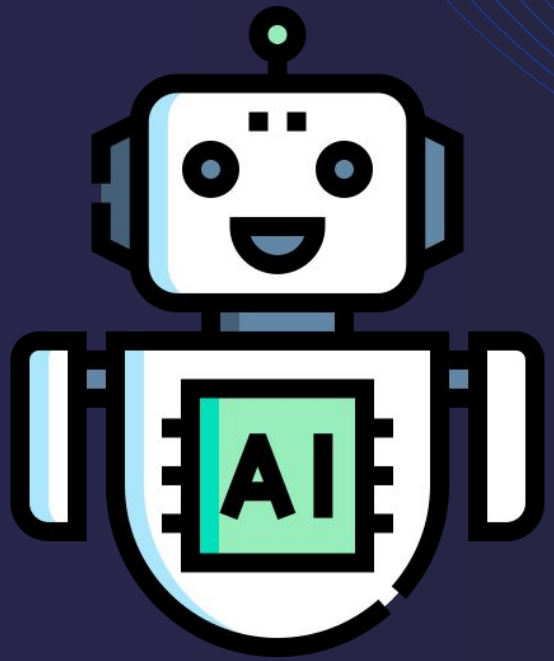
```yaml
version: '3'

services:
   web:
     build: .
     command: python app.py
     ports:
       - "80:80"
     restart: always
```

Challenge #2

Moteur de Recherche Semantique

Avec TF-IDF ou BM25-RANK.



PARIS ynov CAMPUS

TF-IDF

Mesure utilisée en **text mining, recherche d'information**, et **traitement** du **langage naturel (NLP)** pour **évaluer l'importance d'un mot dans un document** par rapport à un **corpus**.

$$TF - IDF(t) = TF(t) \times IDF(t)$$

Méthode utiliser pour vectoriser des documents avant training

Corrélation Pearson

Une mesure statistique qui **indique** la **force et la direction d'une relation linéaire** entre **deux variables numériques.**

$$r = \frac{\sum(X_i - \bar{X})(Y_i - \bar{Y})}{\sqrt{\sum(X_i - \bar{X})^2} \times \sqrt{\sum(Y_i - \bar{Y})^2}}$$

r est le **coefficient de corrélation de Pearson**, qui varie entre :

- **+1** (corrélation positive parfaite)
- **0** (aucune corrélation)
- **-1** (corrélation négative parfaite)

Langage naturel 1/3.

```
#### Recherche de l'utilisateur
query = input("Saisir votre recherche")

# Textes à comparer
texts = [
query,
"Le traitement du langage naturel est fascinant.",
"Le traitement des langues est une branche de l'intelligence artificielle.",
"L'analyse de texte est utilisée pour la traduction automatique."
]
```

Langage naturel 2/3.

pip install scikit-learn

```
from sklearn.feature_extraction.text import TfidfVectorizer

# Vectorisation TF-IDF
vect = TfidfVectorizer()
tfidf_mat = vect.fit_transform(texts).toarray()

query_tf_idf = tfidf_mat[0]
corpus = tfidf_mat[1:]
```

Langage naturel 3/3.

pip install scipy

```
from scipy.stats import pearsonr

# Corellation de pearson
for id, document_tf_idf in enumerate(corpus):
   pearson_corr, _ = pearsonr(query_tf_idf, document_tf_idf)
   if pearson_corr > 0.20:
       result = {"ID": id, "document": texts[id+1], "similarity": pearson_corr}
       print(str(result))
```

Challenge #3

Analyse de sentiment des

From scratch, Bert ou avec comprehend.

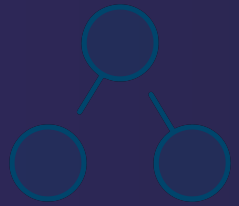
PARIS ynov CAMPUS

Bon à savoir...

XGBOOST

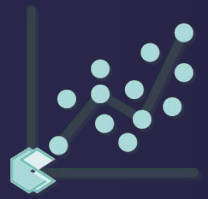
XGBoost (**eXtreme Gradient Boosting**) est une implémentation open source optimisée et parallélisée du Gradient Boosting, créée par Tianqi Chen, Doctorant à l'Université de Washington.

Apprentissage supervisée.

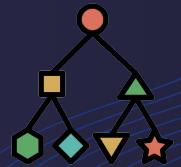


Utilise les arbres de décision (comme Random Forest).

ranking



Régression



Classification

Etapes.

- Importer les bibliothèques
- Définir les paramètres
- Création de prédicteurs et de variables cibles
- Division des données d’entraînement et de tests
- Initialisation du modèle d'apprentissage automatique XGBoost
- Validation croisée dans le jeu de données Train
- Importance des fonctionnalités
- Rapport de prédiction

Importation des bibliothèques

Après installations des différents packages, l'étape suivante consiste à importer les bibliothèques requises pour le mini projet.

```
import warnings
warnings.simplefilter('ignore')
import os
import pandas, unidecode, json
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.model_selection import train_test_split,
RandomizedSearchCV
from sklearn.preprocessing import OneHotEncoder, LabelEncoder
from sklearn.metrics import accuracy_score
from xgboost import XGBClassifier
```

Définir les paramètres.

Nous définissons la liste des actions avec lesquelles nous travaillerons : chemin du jeu de données, séparateur, vectoriseur, score, modèle.

```
class Main:


    def __init__(self):
        # Init
            dataset_path, separator = [os.path.join('model',
        "dataset.txt"), "    "]
            # Load and Train model
            self.dataset = pandas.read_csv(dataset_path,
        names=['sentence', 'label'], sep=separator)
            self.vectorizer = None
            self.score = None
            self.model = None
            self.train()
```

Division des données d'entraînement et de test.

Division de toutes les données du jeu de données en données de test et d'entraînement.

```
    def train(self):

        # Separate dataset and expected output
            sentences = self.dataset['sentence'].values
            y = self.dataset['label'].values

            # Split datasets
            sentences_train, sentences_test, y_train, y_test =
        train_test_split(sentences, y, test_size=0.25,
        random_state=1000)

            # Verctorization of training and testing data
            self.vectorizer = CountVectorizer()
            self.vectorizer.fit(sentences_train)
            X_train = self.vectorizer.transform(sentences_train)
            X_test  = self.vectorizer.transform(sentences_test)

            # Init model and fit it
            self.model = XGBClassifier(max_depth=2,
        n_estimators=30)
            self.model.fit(X_train, y_train)
```

Évaluer les prédictions.

L'exactitude est définie comme le pourcentage de prédictions correctes pour les données de test. Elle peut être facilement calculée en divisant le nombre de prédictions correctes par le nombre total de prédictions.

```
        # Show xboost parameters
        print(self.model)

        # make predictions for test data
        y_pred = self.model.predict(X_test)
        predictions = [round(value) for value in y_pred]

        # evaluate predictions
        self.score = accuracy_score(y_test, predictions)
        print("Accuracy: %.2f%%" % (self.score * 100.0))
```

Prediction

```
# predictions
        result =
self.vectorizer.transform([unidecode.unidecode(json_text)])
        result = self.model.predict(result)

        if str(result[0]) == "0":
            sentiment = "NEGATIVE"

        elif str(result[0]) == "1":
            sentiment = "POSITIVE"
```

Usage

```
if __name__ == "__main__":
    main = Main()
    print(main.predict("Depuis ce matin votre application ne marche pas, je n'arrive pas à déverrouiller ma voiture."))
    print(main.predict("j'ai adore la prestation"))
```

SOUTENANCES

À partir de 16h00