Pipeline MLOps Complet
Classification Automatique de Texte

GitHub Actions • Docker • MLflow • CML • FastAPI • Angular

Réalisé par :
Akram BENHAMMOU & Oussama KHOUYA

MLOPS & DEVOPS

15/12/2025

Sommaire

Contexte du Projet

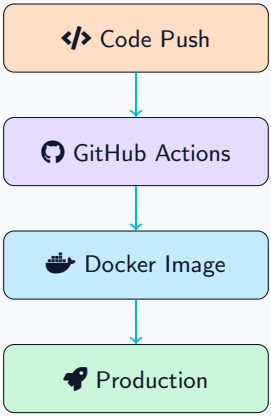
Problématique

Comment automatiser le cycle de vie complet d'un modèle de Machine Learning, de l'entraînement au déploiement en production ?

Cas d'usage : Classification automatique d'articles de journaux

Objectifs du projet

- Pipeline **CI/CD** automatisé
- Tracking d'expériences **MLflow**
- Conteneurisation **Docker**
- Déploiement **Staging** → **Production**



Les 7 Catégories de Classification

Informatique | Sport | Science

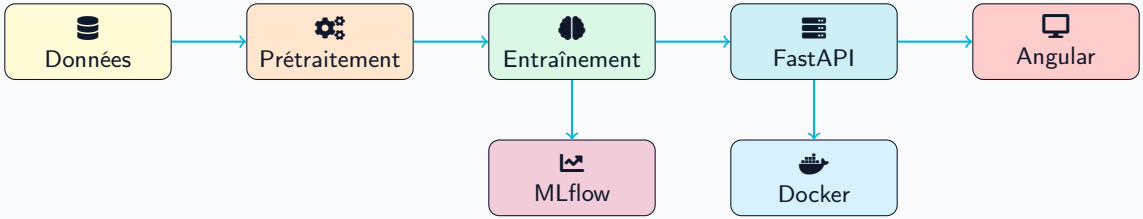
Politique | Religion | Automobile

Commerce

Dataset : 20 Newsgroups

- **18,846 articles** répartis en 20 classes originales
- Regroupement en **7 catégories** pour simplification
- Split : 80% entraînement / 20% test

Architecture Globale du Système



Data Pipeline
NLTK, pandas
TF-IDF

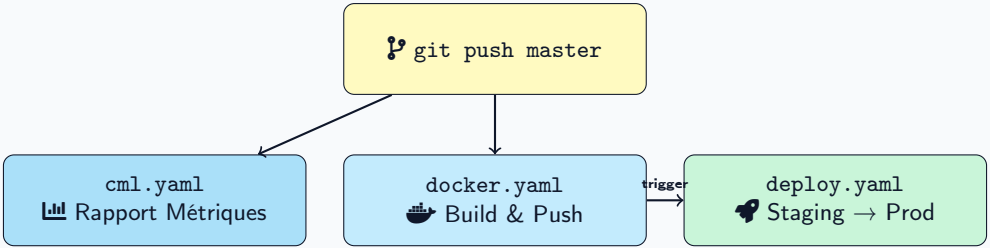
ML Training
scikit-learn
Random Forest

MLOps
MLflow, CML
Tracking

Deployment
Docker, GHCR
FastAPI

GitHub Actions - Pipeline CI/CD

Automatisation complète avec 3 workflows



1. CML Report

- Entraînement modèle
- Génération métriques
- Commentaire PR auto

2. Docker Build

- Tests pytest
- Build image
- Push vers GHCR

3. Deploy Pipeline

- Deploy Staging
- Tests d'intégration
- Deploy Production

Workflow CML - Continuous Machine Learning

cml.yaml - Étapes clés

```yaml
name: CML Report
on:
  push:
    branches: [ master ]

jobs:
  train-and-report:
    steps:
      - uses: iterative/setup-cml@v2

      - name: Preprocess Data
        run: python src/preprocess.py

      - name: Train Model
        run: python src/train.py

      - name: Write CML Report
        run: |
          cml publish reports/...
          cml comment create report.md
```

Fonctionnalités CML

Rapport Automatique

- Setup Python 3.9
- Prétraitement données
- Entraînement modèle
- Génération métriques JSON
- Matrice de confusion
- Commentaire GitHub auto

Outil : iterative/cml

Permet de publier des rapports ML directement dans les Pull Requests GitHub

Workflow Docker - Build & Push

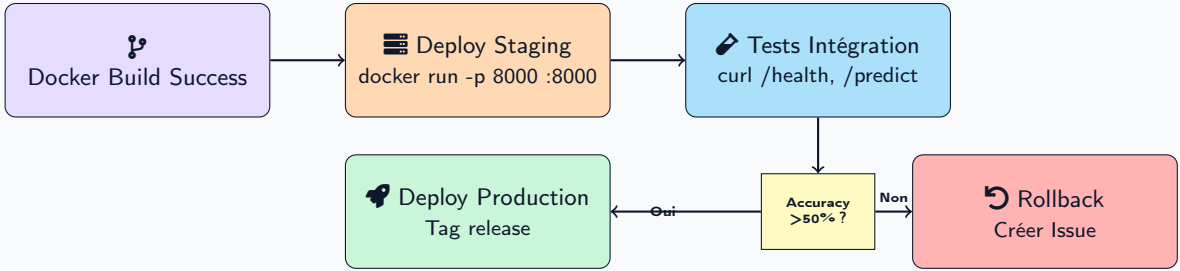
docker.yaml - Pipeline

```yaml
name: Docker Build & Push
env:
  REGISTRY: ghcr.io
  IMAGE_NAME: ${{ github.repository }}

jobs:
  build-and-push:
    steps:
      - name: Run tests
        run: pytest tests/ -v

      - name: Log in to GHCR
        uses: docker/login-action@v3

      - name: Build and push
        uses: docker/build-push-action@v5
        with:
          push: true
          tags: |
            type=sha
            type=raw,value=latest
```

Dockerfile Optimisé

```dockerfile
FROM python:3.9-slim

WORKDIR /app

# Cache des dependances
COPY requirements.txt .
RUN pip install --no-cache-dir \
    -r requirements.txt

# Ressources NLTK
RUN python -c "import nltk; \
    nltk.download('stopwords'); \
    nltk.download('wordnet')"

COPY src/ ./src/
COPY models/ ./models/

EXPOSE 8000

CMD ["uvicorn", "src.app:app", \
     "--host", "0.0.0.0", "--port", "8000"]
```

Workflow Deploy - Staging → Production



Tests d'intégration Staging

- `GET /health` → model_loaded : true
- `POST /predict` → Test prédiction
- Seuil accuracy minimum : 50%

Rollback Automatique

- Restauration version précédente
- Création Issue GitHub automatique
- Notification équipe

Tests Automatisés - pytest

Suite de tests complète exécutée dans le CI/CD

test_preprocess.py

Tests du prétraitement

- clean_text()
- process_text()
- Gestion stopwords
- Lemmatisation

test_train.py

Tests d'entraînement

- Chargement données
- TF-IDF vectorizer
- RandomForest fit
- Sauvegarde modèle

test_api.py

Tests API FastAPI

- GET /health
- POST /predict
- POST /upload
- Codes erreur

```
# Execution dans le workflow docker.yaml
- name: Run tests
  run: pytest tests/ -v --tb=short
```

MLflow - Tracking d'Expériences

Intégration MLflow

```python
import mlflow

mlflow.set_tracking_uri("file:./mlruns")
mlflow.set_experiment("Text_Classification")

with mlflow.start_run():
    # Hyperparametres
    mlflow.log_param("max_features", 5000)
    mlflow.log_param("n_estimators", 100)

    # Metriques
    mlflow.log_metric("accuracy", accuracy)
    mlflow.log_metric("f1_score", f1)

    # Artefacts
    mlflow.sklearn.log_model(model, "model")
```

Éléments trackés

Paramètres

- `max_features` : 5000
- `n_estimators` : 100

Métriques

- Accuracy, Precision
- Recall, F1-Score

Artefacts

- Modèle
- TF-IDF Vectorizer
- Matrice de confusion

Rapport GitHub Automatique

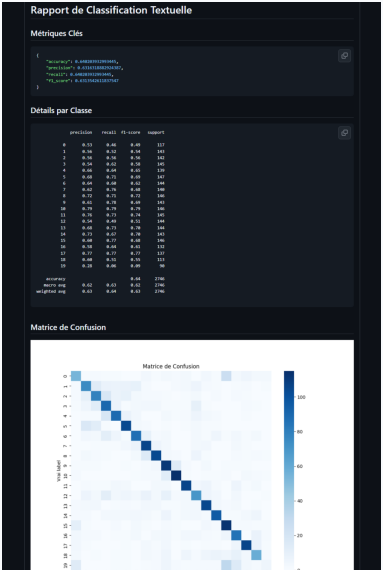
Contenu du rapport

1. Metriques JSON (accuracy, f1)
2. Rapport par classe (txt)
3. Matrice confusion (image)

Avantages CML

- Visibilite dans PR
- Historique performances

Exemple de rapport



FastAPI - Backend API REST

Endpoints de l'API

Méthode	Endpoint
GET	/health
POST	/predict
POST	/upload
GET	/docs

Fonctionnalités

- Support PDF, DOCX, TXT
- Scores de confiance
- Health check
- Swagger auto (OpenAPI)
- CORS configuré

Exemple de réponse /predict

```
{
  "text": "Computer graphics and...",
  "prediction_class_id": 4,
  "category_name": "Informatique",
  "confidence_scores": [
    {"name": "Informatique", "value": 0.72},
    {"name": "Science", "value": 0.15},
    {"name": "Commerce", "value": 0.08},
    ...
  ],
  "status": "success"
}
```

Lifespan Pattern

Chargement du modèle au démarrage via `@asynccontextmanager`

Angular 21 - Interface Utilisateur

Application Web Moderne

Fonctionnalités

- Saisie de texte libre
- Import fichiers (PDF, DOCX, txt,md...)
- Barres de classes dynamiques

Technologies

- Angular 21
- Design moderne
- HttpClient pour API

Démonstration - Docker en Local

Lancer l'application conteneurisée

1. Pull & Run

```
docker pull ghcr.io/akrambenhammou-e/\
  classification-texte-pipeline-ci-cd:latest
docker run -d -p 8000:8000 --name app \
  ghcr.io/.../...:latest
```

2. Tester l'API

```
# Health check
curl http://localhost:8000/health

# Prediction
curl -X POST localhost:8000/predict \
  -H "Content-Type: application/json" \
  -d '{"text": "GPU NVIDIA gaming"}'
```

Démonstration Live

Étapes de la démonstration

1. Lancer l'API : `uvicorn src.app:app -reload`
2. Lancer le frontend : `cd frontend && ng serve`
3. Ouvrir `http://localhost:4200`
4. Tester avec un article en anglais
5. Observer les probabilités par catégorie

API Backend
`http://localhost:8000`
Swagger : `/docs`

Frontend Angular
`http://localhost:4200`
Interface utilisateur

MLflow UI
`mlflow ui`
Port 5000

Performances du Modèle

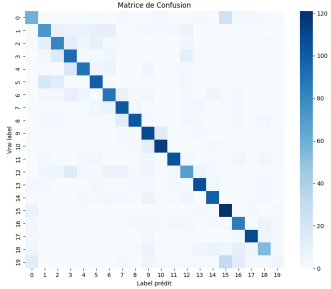
Métriques d'évaluation

Métrique	Score
Accuracy	64%
Precision (weighted)	63%
Recall (weighted)	64%
F1-Score (weighted)	63%

Configuration Modèle

- RandomForest (100 arbres)
- TF-IDF (5000 features)
- Train/Test : 80/20

Matrice de Confusion



Stack Technologique Complète

Catégorie	Outils	Rôle
DevOps	GitHub Actions Docker, GHCR	CI/CD automatisé Conteneurisation
MLOps	MLflow CML	Tracking expériences Rapports ML dans PR
ML	scikit-learn NLTK, pandas	Modèle RandomForest Prétraitement NLP
Backend	FastAPI Uvicorn, Pydantic	API REST Serveur ASGI
Frontend	Angular 21	Interface utilisateur
Tests	pytest, httpx	Tests automatisés

Conclusion & Perspectives

Réalisations

- Pipeline **CI/CD complet**
  - 3 workflows GitHub Actions
  - Tests automatisés (pytest)
  - Rollback automatique
- **MLOps** intégré
  - MLflow tracking
  - CML reporting
- **Déploiement** Docker
  - Image optimisée
  - GitHub Container Registry

Améliorations possibles

- **Modèle**
  - Deep Learning (BERT)
  - Support multilingue
- **Infrastructure**
  - Kubernetes (scaling)
  - Prometheus (monitoring)
- **MLOps avancé**
  - A/B Testing
  - Feature Store
  - Model Registry

Merci de votre attention !

❓ Questions ?

github.com/AkramBENHAMMOU-e/Classification-Texte-Pipeline-CI-CD

Présenté par : Akram BENHAMMOU & Oussama KHOUYA
Encadré par : Pr. Soufiane HAMIDA
Master 2 - AI & Systemes distrubués