trainedml

Framework pédagogique et modulaire de Machine Learning en Python

Yéro Diamanka

Master SSD - Statistique et Sciences des Données
Université de Montpellier

Encadrant : **Bilel Bensaid**

Projet HAX712X - Software Development for Data Science

26 décembre 2025

github.com/diamankayero/trainedml

Plan de la présentation

Plan de la présentation

Contexte du projet

Contexte académique
Projet réalisé dans le cadre du cours HAX712X (Software Development for Data Science) du Master SSD à l'Université de Montpellier.

Motivation du choix
Au-delà des projets de visualisation de données typiques, création d'un outil réutilisable pour :

- Faciliter l'apprentissage pratique du Machine Learning
- Appliquer les bonnes pratiques de développement logiciel
- Créer un package Python complet et professionnel

Problématique
Comment construire un framework ML qui soit à la fois :

- Pédagogique (code clair, bien documenté)
- Modulaire et extensible (architecture robuste)
- Accessible (interfaces CLI, Web et API)

Objectifs du projet

Objectifs principaux

1. **Développement logiciel** : Créer un package Python professionnel
   - Architecture orientée objet (classes, héritage, abstraction)
   - Tests unitaires avec pytest
   - Documentation avec Sphinx
   - Intégration continue (CI/CD)
2. **Machine Learning** : Implémenter des algorithmes de classification
   - KNN, Régression Logistique, Random Forest
   - Système de benchmark automatique
3. **Visualisation** : Interface interactive
   - Application web Streamlit
   - Graphiques interactifs (Plotly, Matplotlib)

Public cible : Étudiants en data science, enseignants

Plan de la présentation

Architecture globale

Structure du projet

- **src/trainedml/** : Package principal (sans doc)
  - `models/` : Algorithmes ML (base.py, knn.py, logistic.py, random_forest.py)
  - `data/` : Gestion des datasets (loaders.py)
  - `viz/` : Visualisations (visualization.py)
  - `utils/` : Utilitaires (factory pattern)
  - `benchmark.py` : Comparaison de modèles
  - `cli.py` : Interface en ligne de commande
- **trainedml_webapp/** : Application Streamlit
- **tests/** : Tests unitaires (couverture $> 80\%$)
- **doc/** : Documentation Sphinx
- **slides/** : Présentation Beamer (cette présentation)

Conformité HAX712X : Respect de toutes les exigences du cahier des charges

Hiérarchie des modèles : Pattern Template Method

Programmation orientée objet

```python
"""
Définit l'interface commune à tous les modèles supervisés.
"""
from abc import ABC, abstractmethod

class BaseModel(ABC):
    """
    Classe abstraite pour les modèles de classification.
    Toutes les classes de modèles doivent hériter de
    cette classe et implémenter ses méthodes.
    """
    def __init__(self):
        self.model = None  # L'objet du modèle sous-
        jacent (scikit-learn, etc.)

    @abstractmethod
    def fit(self, X, y):
        """Entraîne le modèle sur les données X et la
        cible y."""
        pass

    @abstractmethod
    def predict(self, X):
        pass

    @abstractmethod
    def evaluate(self, X, y):
        pass
```

Avantages : Abstraction, réutilisabilité, extensibilité

Modèles disponibles

Trois algorithmes de classification

1. **K-Nearest Neighbors (KNN)**
   - Algorithme basé sur la proximité
   - Paramètres : nombre de voisins k, métrique de distance
2. **Régression Logistique**
   - Modèle linéaire probabiliste
   - Paramètres : régularisation, solveur
3. **Random Forest**
   - Ensemble d'arbres de décision
   - Paramètres : nombre d'arbres, profondeur max

Tous les modèles héritent de `BaseModel` et implémentent l'interface commune

Exemple : Implémentation KNN

```python
"""
Implémentation du modèle K-Nearest Neighbors pour trainedml.
"""
from .base import BaseModel
from sklearn.neighbors import KNeighborsClassifier

class KNNModel(BaseModel):
    """
    Modèle de classification K-Nearest Neighbors.
    """
    def __init__(self, n_neighbors=5):
        super().__init__()
        self.model = KNeighborsClassifier(n_neighbors=n_neighbors)

    def fit(self, X, y):
        """Entraîne le modèle KNN."""
        self.model.fit(X, y)

    def predict(self, X):
        """Prédit la classe pour de nouvelles données."""
        return self.model.predict(X)

    def evaluate(self, X, y):
        """Retourne la précision du modèle sur les données de test."""
        return self.model.score(X, y)
```

Plan de la présentation

Utilisation du CLI

Interface en ligne de commande avec argparse

```
# Afficher l'aide complète
python -m trainedml.cli --help

# Benchmark automatique sur Iris
python -m trainedml.cli --benchmark --dataset iris

# Entraîner un modèle spécifique et un dataset avec un url
python -m trainedml.cli --url https://archive.ics.uci.edu/ml/machine-learning-
databases/wine-quality/winequality-red.csv --target quality --model logistic --show

# Entraîner avec visualisation
python -m trainedml.cli --benchmark --dataset iris --show
```

Fonctionnalités :

- Entraînement simple ou comparaison multiple
- Sauvegarde des résultats (JSON, CSV)
- Génération de rapports automatiques

Application web interactive

Interface graphique avec Streamlit

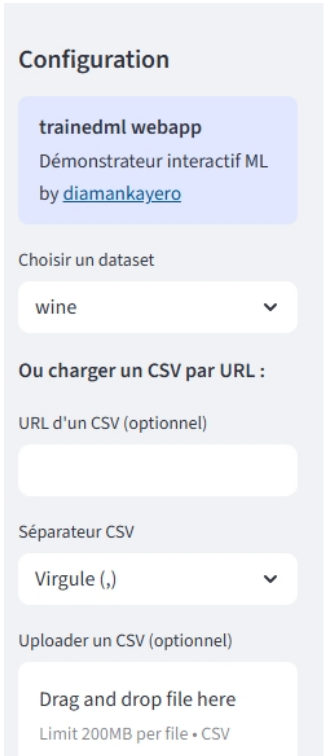
```
# Installation
pip install -e .

# Lancer l'application
streamlit run trainedml_webapp/src/app.py
```

Fonctionnalités de l'application :

1. **Chargement** : Datasets intégrés (Iris, Wine, Breast Cancer) ou CSV
2. **Exploration** : Statistiques descriptives, distributions, corrélations
3. **Entraînement** : Configuration des modèles et hyperparamètres
4. **Évaluation** : Métriques (accuracy, precision, recall, F1)
5. **Visualisation** : Matrices de confusion, courbes ROC
6. **Prédiction** : Interface pour tester sur nouvelles données

Captures d'écran de l'application

Interface Streamlit



Widgets interactifs : Sliders, selectbox, file uploader, boutons

Utilisation programmatique

API Python simple et intuitive

```python
from trainedml import Trainer
from trainedml.data import load_dataset

# Charger les données
X_train, X_test, y_train, y_test = load_dataset("iris")

# Créer et entraîner un modèle
trainer = Trainer(model="random_forest", n_estimators=100)
trainer.fit(X_train, y_train)

# Évaluer les performances
results = trainer.evaluate(X_test, y_test)
print(f"Accuracy: {results['accuracy']:.3f}")

# Faire des prédictions
predictions = trainer.predict(X_test)

# Sauvegarder le modèle
trainer.save("my_model.pkl")
```

Intégration facile dans notebooks Jupyter ou scripts

Plan de la présentation

Comparaison automatique de modèles

Module benchmark.py
Fonctionnalité clé pour comparer les performances :

- **Entraînement automatique** : Tous les modèles disponibles
- **Validation croisée** : 5-fold CV pour robustesse
- **Métriques multiples** :
  - Accuracy, Precision, Recall, F1-Score
  - Temps d'entraînement et de prédiction
  - Utilisation mémoire
- **Visualisations** :
  - Tableau comparatif
  - Graphiques de performances
  - Matrices de confusion côte à côte

Objectif : Identifier le meilleur modèle pour un problème donné

Exemple de benchmark

```python
from .evaluation import Evaluator
import time

class Benchmark:
    def __init__(self, models):

        self.models = models

    def run(self, X_train, y_train, X_test, y_test):
        results = {}
        for name, model in self.models.items():
            # Mesure du temps d'entraînement
            start_fit = time.time()
            model.fit(X_train, y_train)
            fit_time = time.time() - start_fit

            # Mesure du temps de prédiction
            start_pred = time.time()
            y_pred = model.predict(X_test)
            predict_time = time.time() - start_pred

            scores = Evaluator.evaluate_all(y_test, y_pred)
            results[name] = {
                'scores': scores,
                'fit_time': fit_time,
                'predict_time': predict_time
            }
        return results
```

Sortie : DataFrame pandas + graphiques + rapport JSON

Outils de visualisation

Module visualization.py
Fonctions de visualisation pour l'analyse exploratoire :

- **Distributions** : Histogrammes, boxplots, violin plots
- **Relations** : Scatter plots, heatmap de corrélation
- **Performances** :
  - Matrices de confusion
  - Courbes ROC et AUC
  - Courbes d'apprentissage (learning curves)
  - Feature importance (pour Random Forest)
- **Interactivité** : Support Plotly pour graphiques interactifs

Architecture : Classes abstraites pour supporter différents backends (Matplotlib, Plotly, Seaborn)

Plan de la présentation

Tests unitaires et intégration continue

Tests avec pytest (couverture $>$ 80%)

```python
# tests/test_models/test_knn.py
import unittest
from trainedml.data.loader import DataLoader
from trainedml.models.knn import KNNModel
from sklearn.model_selection import train_test_split

class TestKNNModel(unittest.TestCase):
    def setUp(self):
        # Chargement du dataset Iris depuis une URL publique
        X, y = DataLoader().load_dataset(name="iris")
        self.X_train, self.X_test, self.y_train, self.y_test =
        train_test_split(X, y, test_size=0.3, random_state=42)

    def test_fit_predict(self):
        #Teste l'entraînement et la prédiction du modèle KNN sur Iris.
        model = KNNModel(n_neighbors=3)
        model.fit(self.X_train, self.y_train)
        preds = model.predict(self.X_test)
        self.assertEqual(len(preds), len(self.y_test))

    def test_evaluate(self):
        model = KNNModel(n_neighbors=3)
        model.fit(self.X_train, self.y_train)
        score = model.evaluate(self.X_test, self.y_test)
        self.assertTrue(0.0 <= score <= 1.0)

if __name__ == '__main__':
    unittest.main()
```

CI/CD : GitHub Actions pour tests automatiques à chaque commit

Qualité du code et bonnes pratiques

Respect des standards Python

- **PEP 8** : Style de code conforme
- **Type hints** : Annotations de types pour clarté
- **Docstrings** : Google style pour toutes les fonctions/classes
- **Linting** : Vérification avec pylint, flake8
- **Formatting** : Code formaté avec black

Gestion de version

- **Git** : Branches multiples (main, dev, feature/*)
- **.gitignore** : Exclusion fichiers inutiles (`__pycache__`, `.pytest_cache`, etc.)
- **Commits** : Messages clairs et descriptifs
- **Merge** : Pull requests avec revue de code

Documentation complète

Documentation multi-niveaux

1. **README.md principal**
   - Description du projet
   - Installation (`pip install -e .`)
   - Exemples d'utilisation
   - Structure du projet
   - Licence (MIT)
2. **Documentation API (Sphinx)**
   - API complète générée automatiquement
   - Tutoriels et guides
   - Déployée sur GitHub Pages
3. **Docstrings dans le code**
   - Format Google style
   - Descriptions, paramètres, retours, exemples

Évaluation temps/mémoire

Profiling et optimisation

```
"""
Module d'évaluation des modèles de classification pour trainedml.
"""
from sklearn.metrics import accuracy_score,
precision_score, recall_score, f1_score

class Evaluator:
    """
    Classe utilitaire pour évaluer les performances d'un
    modèle de classification.
    """
    @staticmethod
    def evaluate_all(y_true, y_pred):

        return {
            'accuracy': accuracy_score(y_true, y_pred),
            'precision': precision_score(y_true, y_pred,
            average='weighted', zero_division=0),
            'recall': recall_score(y_true, y_pred,
            average='weighted', zero_division=0),
            'f1': f1_score(y_true, y_pred,
            average='weighted', zero_division=0)
        }
```

Plan de la présentation

Démonstration pratique

Scénario : Classification du dataset Iris

1. **Chargement et exploration**
   - 150 échantillons, 3 espèces, 4 features
   - Visualisation des distributions
   - Matrice de corrélation
2. **Benchmark des modèles**
   - Comparaison KNN, Logistic, Random Forest
   - Métriques de performance
   - Temps d'exécution
3. **Entraînement du meilleur modèle**
   - Configuration des hyperparamètres
   - Visualisation de la matrice de confusion
4. **Prédictions interactives**
   - Test sur nouvelles observations

Démonstration live de l'application Streamlit

Plan de la présentation

Réalisations

Conformité avec le cahier des charges HAX712X

- ✓ **Programmation objet** : Classes, héritage, abstraction
- ✓ **Tests unitaires** : Pytest, couverture $> 80\%$
- ✓ **Documentation** : Sphinx + docstrings + README
- ✓ **CI/CD** : GitHub Actions
- ✓ **Git** : Branches multiples, .gitignore
- ✓ **Profiling** : Temps/mémoire évalués
- ✓ **Interface interactive** : Application Streamlit

Points bonus obtenus :

- Architecture fortement orientée objet
- Chargement automatique des datasets (pooch)
- Évaluation performances temps/mémoire
- Originalité : framework ML complet

Retour d'expérience

Difficultés rencontrées

- Conception d'une architecture modulaire et extensible
- Gestion des dépendances et compatibilité des versions
- Déploiement de l'application Streamlit
- Tests de tous les cas limites (edge cases)

Compétences acquises

- Développement d'un package Python professionnel
- Maîtrise de Git et GitHub (branches, merges, CI/CD)
- Tests unitaires et intégration continue
- Documentation technique (Sphinx)
- Création d'interfaces utilisateur (CLI, Web)

Impact pédagogique : Outil réutilisable pour l'enseignement du ML

Évolutions futures

Améliorations à court terme

1. **Nouveaux algorithmes**
   - SVM (Support Vector Machines)
   - Naive Bayes
   - Gradient Boosting (XGBoost, LightGBM)
2. **Fonctionnalités avancées**
   - GridSearchCV pour optimisation automatique
   - Feature selection et engineering
   - Cross-validation stratifiée
3. **Extensions**
   - Support de la régression (en plus de la classification)
   - Clustering (K-means, DBSCAN, Hierarchical)
   - Réduction de dimensionnalité (PCA, t-SNE, UMAP)

Perspectives à long terme

Vision future du projet

- **Deep Learning** : Intégration de réseaux de neurones (PyTorch/TensorFlow)
- **AutoML** : Sélection automatique du meilleur modèle
- **Explainability** : SHAP values, LIME pour interpréter les modèles
- **Production** : API REST pour déploiement en production
- **Scalabilité** : Support de gros datasets (Dask, PySpark)

Communauté

- Publication sur PyPI pour installation via `pip install trainedml`
- Contributions open source bienvenues
- Création d'une documentation utilisateur complète
- Tutoriels vidéo et exemples d'utilisation

Merci de votre attention !

Questions ?

Liens et ressources

github.com/diamankayero/trainedml

Documentation, code source, exemples

HAX712X Course Page

Cours Software Development for Data Science

Contributions et feedback bienvenus !