Dossier Technique - Projet Machine Learning Vin

Table des matières

1. Présentation du projet

Ce projet vise à développer une application interactive en Python avec Streamlit qui présente un pipeline complet de Machine Learning appliqué à un dataset de vins (vin.csv). L'application permet :

- L'exploration des données
- Le prétraitement avancé (gestion des valeurs manquantes par différentes stratégies, suppression de colonnes)
- La modélisation avec différents algorithmes ML (Logistic Regression, Random Forest, SVM) et Deep Learning (Keras)

- L’évaluation détaillée des modèles via métriques, visualisations, et comparaison rapide avec LazyPredict
- L’optimisation automatique des hyperparamètres avec GridSearchCV
- La prédiction interactive sur de nouvelles données, avec gestion automatique des valeurs manquantes

2. Architecture et organisation du code

```
├── data/
│   └── vin.csv              # Jeu de données CSV (à fournir manuellement)
├── models/                  # Modèles ML sauvegardés (joblib / Keras)
│   ├── logistic_regression.joblib
│   ├── random_forest.joblib
│   ├── label_encoder.joblib
│   ├── scaler.pkl
│   ├── encoder.pkl
│   ├── model_dl.h5
│   └── feature_names.txt
├── models_dl/
│   ├── scaler.pkl
│   ├── encoder.pkl
│   ├── model_dl.h5
├── pages/
│   ├── exploration.py
│   ├── pretraitement.py
│   ├── training.py
│   ├── evaluation.py
│   ├── deep_learning.py
│   └── app.py
├── setup.py
├── run.py
├── requirements.txt
├── README.md
└── .gitignore
```

8. 📦 Dépendances techniques

3. Dépendances techniques

Le projet repose sur un ensemble de bibliothèques Python bien établies, assurant à la fois **robustesse**, **performance**, et **facilité d'utilisation**. La version exacte des dépendances est figée dans requirements.txt afin de garantir la **reproductibilité** de l'environnement.

Interface utilisateur

- streamlit==1.31.1
  → Permet de créer une interface web interactive et réactive sans développement front-end.

Manipulation et visualisation de données

- numpy==1.23.5
  → Traitement des tableaux multidimensionnels, calculs vectorisés rapides
- pandas==2.2.1
  → Manipulation de données tabulaires avec DataFrame, nettoyage et transformation
- seaborn==0.13.2
  → Visualisations statistiques avancées (boxplots, heatmaps, etc.)
- matplotlib==3.8.4
  → Visualisation de données personnalisée et compatible avec seaborn

Machine Learning

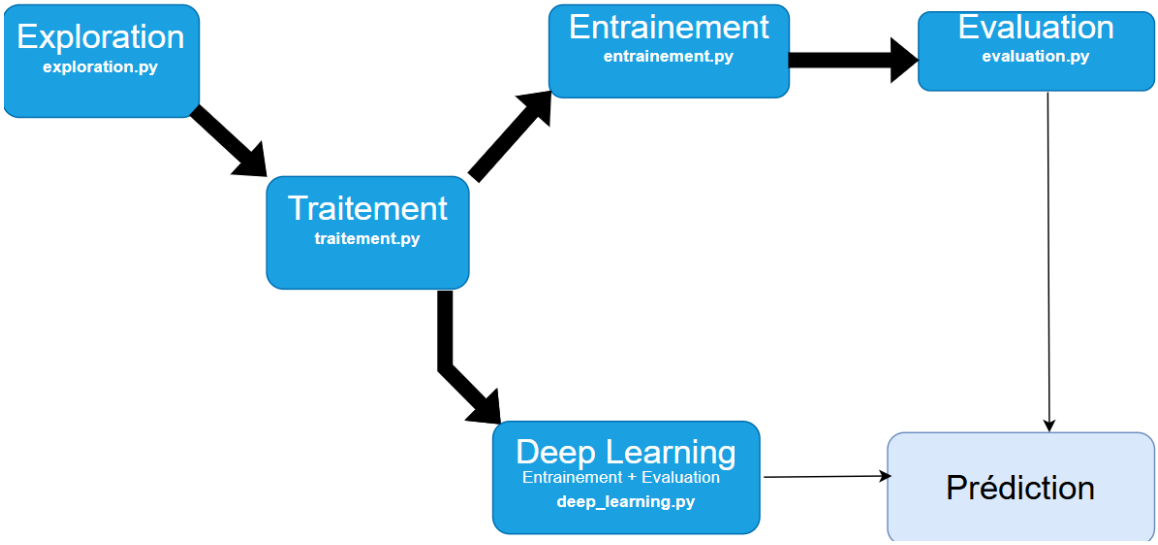
- scikit-learn==1.2.2
  → Modèles supervisés classiques (Logistic Regression, Random Forest, SVM)
  → Pipeline d'entraînement, encodage, standardisation, GridSearchCV
- lazypredict==0.2.12
  → Comparaison automatisée d'un grand nombre de modèles sans configuration manuelle

Deep Learning

- tensorflow==2.12.0
  → Framework de Deep Learning utilisé avec Keras (modèles denses, Dropout, EarlyStopping)
  → Optimisation avec Adam, gestion des callbacks, sauvegarde en .h5

4. Description des modules

Pipeline générale du projet :



Chaque module du projet correspond à une étape clé du pipeline Machine Learning. Ils sont organisés de manière modulaire et sont appelés dynamiquement dans l'application Streamlit.

exploration.py

- Affichage du DataFrame et statistiques descriptives
- Visualisations interactives
- Sélection dynamique des colonnes numériques

traitement.py

- Nettoyage des données
- Imputation configurable (moyenne, médiane, mode)
- Suppression de colonnes

entrainement.py

- Prétraitement des données
- Entraînement ML (Logistic Regression, Random Forest, SVM)
- Entraînement DL avec Keras
- Sauvegarde des artefacts

evaluation.py

- Chargement des modèles et encodeurs
- Interface prédiction manuelle/CSV
- Imputation automatique
- Prédiction avec décodage des labels

deep_learning.py

- Entraînement DL avec Keras
- GridSearchCV, LazyPredict

- Construction et entraînement du modèle Keras
- Sauvegarde/chargement du modèle, scaler, encodeur

app.py

- Point d'entrée Streamlit
- Navigation fluide via sidebar

5. Scripts utilitaires

setup.py :
- Installe automatiquement les dépendances Python
- Prépare l'environnement virtuel

run.py :
- Script principal pour démarrer l'application Streamlit

Usage :
- python setup.py
- python run.py
- ou streamlit run run.py

6. Fonctionnalités implémentées

- Chargement dynamique du dataset
- Exploration interactive avec graphiques et statistiques
- Prétraitement avancé : gestion fine des valeurs manquantes
- Suppression de colonnes personnalisable
- Modélisation ML complète avec split, fit, predict
- Comparaison automatique de modèles avec LazyPredict
- Optimisation d'hyperparamètres avec GridSearchCV
- Deep Learning avec Keras (Dropout, EarlyStopping)
- Sauvegarde/chargement de tous les artefacts
- Prédiction interactive, gestion automatique des valeurs manquantes
- Interface utilisateur fluide avec sidebar

7. Conception et maintenance du projet

L'architecture du projet a été pensée pour assurer à la fois clarté, évolutivité et facilité de maintenance. Chaque étape du pipeline (exploration, nettoyage, modélisation, évaluation…) est isolée dans un module distinct, rendant le code plus lisible et facilement modifiable sans impact sur les autres composants.

L'utilisation de Streamlit permet de naviguer intuitivement entre les fonctionnalités via une interface fluide. Le dossier pages/ organise les scripts comme une suite d'étapes logiques, ce qui facilite leur exécution dans un ordre cohérent. Tous les artefacts produits (modèles, encodeurs, scalers, features) sont systématiquement enregistrés dans des dossiers dédiés, ce qui assure leur réutilisabilité immédiate lors des prédictions ou des ajustements futurs.

Pour garantir une bonne maintenabilité et reproductibilité de l'environnement :

- Un fichier requirements.txt recense toutes les dépendances nécessaires.
- Le script setup.py automatise l'installation des bibliothèques.
- Le code est conçu pour s'adapter facilement à de nouveaux jeux de données similaires (autres types de vins...).
- Des tests manuels sont effectués pour valider les étapes critiques (imputation, prédiction, entraînement).
- L'architecture modulaire permet d'ajouter facilement de nouveaux algorithmes, visualisations ou options de traitement.

Les performances sont suivies grâce à des logs internes et des visualisations permettant d'identifier rapidement les anomalies ou dérives. À long terme, le projet pourrait intégrer une interface d'administration plus avancée ou une base de suivi versionné des modèles.

8. Modèles d'intelligence artificielle utilisés

Le cœur du projet repose sur deux grandes familles de modèles d'apprentissage supervisé : les algorithmes de **Machine Learning classiques** et les **réseaux de neurones profonds (Deep Learning)**. Chaque approche a été sélectionnée pour sa complémentarité, ses performances et sa capacité à traiter efficacement des données tabulaires.

Modèles de Machine Learning (Scikit-Learn)

- **Régression Logistique** : simple, interprétable, idéale pour une première baseline.
- **Random Forest** : robuste aux valeurs aberrantes, efficace pour détecter les relations non linéaires.
- **SVM (Support Vector Machines)** : performant pour les marges étroites et les petits jeux de données.

Les étapes de traitement incluent :

- Sélection des variables et encodage (OneHot/LabelEncoder)

- Mise à l'échelle avec StandardScaler
- Optimisation par GridSearchCV
- Évaluation automatisée via **LazyPredict**, pour comparer plusieurs modèles en un clic

Tous les modèles et outils (scalers, encodeurs, features) sont sauvegardés dans le dossier models/, assurant leur réutilisation sans retraitement.

Prétraitement :

from sklearn.preprocessing import StandardScaler, LabelEncoder

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y)

Optimisation:

from sklearn.model_selection import GridSearchCV

grid = GridSearchCV(RandomForestClassifier(), param_grid, cv=5)

grid.fit(X_train, y_train)

Modèle de Deep Learning (Keras)

Un réseau de neurones dense (Fully Connected) conçu avec Keras. Il intègre :

- Plusieurs **couches Dense** avec activation ReLU
- **Dropout** pour éviter le surapprentissage
- **EarlyStopping** pour stopper l'entraînement dès stagnation des performances
- Optimisation avec **Adam**
- Perte : categorical_crossentropy ou binary_crossentropy selon le type de classification

Extrait code Keras :

from keras.models import Sequential

from keras.layers import Dense, Dropout

```
model = Sequential([
    Dense(64, activation='relu', input_shape=(X_train.shape[1],)),
    Dropout(0.3),
    Dense(32, activation='relu'),
    Dropout(0.2),
    Dense(1, activation='sigmoid')
])
```

Avant l'entraînement, les données sont :

- Normalisées (StandardScaler)
- Encodées (LabelEncoder)
- Séparées en jeux d'entraînement/test

Les performances du modèle sont évaluées par :

- Précision, rappel, F1-score
- Matrice de confusion
- Courbes ROC et courbes d'apprentissage

Le modèle final est sauvegardé au format .h5 et utilisé pour des prédictions interactives via l'interface Streamlit.

9. Mode d'emploi pour l'utilisateur

1. Cloner le dépôt Git
2. Installer les dépendances avec :
   python setup.py
3. Lancer l'application :
   python run.py
   ou
   streamlit run run.py
4. Utiliser la sidebar pour accéder aux sections :
   - Exploration
   - Prétraitement
   - Modélisation (ML et Deep Learning)
   - Évaluation et Prédiction

5. Suivre les instructions à l'écran pour charger des données, entraîner les modèles, faire des prédictions

10. Ressources et références

- Documentation officielle Streamlit : https://docs.streamlit.io
- Documentation scikit-learn : https://scikit-learn.org
- Keras Documentation : https://keras.io
- Cours LinkedIn Learning : Python pour la Data Science
- LazyPredict GitHub : https://github.com/shankarpandala/lazypredict