Cours Machine Learning

Chapitre 11

Best Practices en Machine Learning

Objectifs :

— Maîtriser le workflow ML de bout en bout
— Éviter les pièges courants
— Optimiser performances et reproductibilité
— Déployer des modèles en production

Cours ML - Sandbox-ML
Version 1.0 - 2026

Table des matières

1 Workflow ML **2**
1.1 Pipeline standard . . . 2
1.2 Checklist avant entraînement . . . 2

2 Traitement des Données **2**
2.1 Nettoyage . . . 2
2.2 Feature Scaling . . . 3
2.3 Feature Engineering . . . 3

3 Sélection de Modèle **3**
3.1 Guide de choix . . . 3
3.2 Baseline Models . . . 4

4 Validation et Métriques **4**
4.1 Cross-Validation . . . 4
4.2 Choix de métrique . . . 4

5 Hyperparameter Tuning **4**
5.1 Stratégies . . . 4

6 Overfitting et Régularisation **5**
6.1 Diagnostic . . . 5
6.2 Solutions overfitting . . . 5

7 Déploiement **5**
7.1 Sauvegarder le modèle . . . 5
7.2 API REST avec FastAPI . . . 6
7.3 Monitoring . . . 6

8 Reproductibilité **6**
8.1 Random seeds . . . 6
8.2 Version control . . . 7

9 Checklist Production **7**

10 Pièges courants **7**
10.1 Data Leakage . . . 7
10.2 Optimisation sur test set . . . 7
10.3 Sélection de features sur tout le dataset . . . 7

11 Ressources **8**
11.1 Outils . . . 8
11.2 Lectures . . . 8

12 Résumé **8**
12.1 Règles d'or . . . 8

12.2 Workflow résumé 8

13 Notebooks Pratiques **8**

1 Workflow ML

1.1 Pipeline standard

1. **Définir le problème** : Classification, régression, clustering ?
2. **Collecter les données** : Qualité > Quantité
3. **EDA (Exploratory Data Analysis)** : Visualisations, statistiques
4. **Preprocessing** : Nettoyage, features engineering
5. **Train/Val/Test Split** : 60/20/20 ou 70/15/15
6. **Baseline Model** : Modèle simple de référence
7. **Itération** : Modèles plus complexes, hyperparameter tuning
8. **Évaluation finale** : Test set (une seule fois !)
9. **Déploiement** : API, monitoring

1.2 Checklist avant entraînement

- ☐ Données équilibrées (ou stratifiées) ?
- ☐ Features normalisées/standardisées ?
- ☐ Train/val/test splits fixés avec random seed ?
- ☐ Baseline défini ?
- ☐ Métrique d'évaluation choisie ?

2 Traitement des Données

2.1 Nettoyage

Bonne Pratique

Valeurs manquantes :
- Numériques : Imputation médiane/moyenne, ou indicateur manquant
- Catégorielles : Mode ou catégorie "Unknown"
- Supprimer si $< 5\%$ de données manquantes

Listing 1 – Gestion des NaN

```
import pandas as pd
from sklearn.impute import SimpleImputer

# Imputation numérique
imputer_num = SimpleImputer(strategy='median')
df[num_cols] = imputer_num.fit_transform(df[num_cols])

# Imputation catégorielle
imputer_cat = SimpleImputer(strategy='most_frequent')
df[cat_cols] = imputer_cat.fit_transform(df[cat_cols])
```

Table 1 – Méthodes de scaling

Méthode	Formule	Quand
StandardScaler	$(x - \mu)/\sigma$	Gaussian-like, SVM, NN
MinMaxScaler	$(x - \min)/(\max - \min)$	Features bornées [0,1]
RobustScaler	$(x - \text{median})/\text{IQR}$	Présence outliers

2.2 Feature Scaling

À Éviter

Erreur Data Leakage : Toujours fit le scaler sur le train set uniquement !

```
# MAUVAIS
scaler.fit(X)  # Contient test set !
X_train_scaled = scaler.transform(X_train)

# BON
scaler.fit(X_train)
X_train_scaled = scaler.transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

2.3 Feature Engineering

- **Encoding catégorielles**
  - One-Hot : < 10 catégories
  - Label Encoding : Arbres de décision
  - Target Encoding : High cardinality
- **Interactions** : $x_1 \times x_2$, x^2
- **Features temporelles** : Jour semaine, mois, heure
- **Aggregations** : Mean, std, count par groupe

3 Sélection de Modèle

3.1 Guide de choix

Table 2 – Quel algorithme choisir ?

Contexte	Algorithme recommandé
Petites données (< 1K)	Logistic Regression, SVM
Données tabulaires moyennes	Random Forest, XGBoost
Images	CNN (ResNet, EfficientNet)
Texte	Transformers (BERT, GPT)
Séries temporelles	LSTM, Transformers
Interprétabilité requise	Arbres de décision, Régression linéaire
Temps réel critique	Modèles légers (LR, Small NN)

3.2 Baseline Models

Toujours commencer par un modèle simple :

— **Classification** : Logistic Regression, Dummy Classifier

— **Régression** : Linear Regression, Mean Predictor

Listing 2 – Baseline simple

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score

# Baseline
baseline = LogisticRegression(max_iter=1000)
baseline.fit(X_train, y_train)
y_pred = baseline.predict(X_test)
print(f"Baseline Accuracy: {accuracy_score(y_test, y_pred):.3f}")
```

4 Validation et Métriques

4.1 Cross-Validation

Bonne Pratique

Utiliser **StratifiedKFold** pour classification déséquilibrée.

```
from sklearn.model_selection import cross_val_score,
    StratifiedKFold

cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
scores = cross_val_score(model, X_train, y_train, cv=cv, scoring='
    f1')
print(f"CV F1: {scores.mean():.3f} +/- {scores.std():.3f}")
```

4.2 Choix de métrique

Table 3 – Métriques selon le contexte

Contexte	Métrique
Classes équilibrées	Accuracy
Classes déséquilibrées	F1-score, AUC-ROC
Faux négatifs coûteux	Recall
Faux positifs coûteux	Precision
Régression	MSE, MAE, R^2

5 Hyperparameter Tuning

5.1 Stratégies

1. **Grid Search** : Exhaustif, lent
2. **Random Search** : Plus efficace (Bergstra & Bengio 2012)

3. **Bayesian Optimization** : Optuna, Hyperopt

Listing 3 – Random Search

```python
from sklearn.model_selection import RandomizedSearchCV
from scipy.stats import uniform, randint

param_dist = {
    'max_depth': randint(3, 20),
    'n_estimators': randint(50, 500),
    'learning_rate': uniform(0.01, 0.3)
}

search = RandomizedSearchCV(
    model, param_dist, n_iter=50, cv=3,
    scoring='f1', random_state=42, n_jobs=-1
)

search.fit(X_train, y_train)
print(f"Best params: {search.best_params_}")
print(f"Best CV score: {search.best_score_:.3f}")
```

6 Overfitting et Régularisation

6.1 Diagnostic

— **Underfitting** : Train error ET val error élevés Modèle trop simple
— **Overfitting** : Train error faible, val error élevé Modèle trop complexe
— **Good fit** : Train error ≈ val error, tous deux faibles

6.2 Solutions overfitting

Bonne Pratique

Arsenal anti-overfitting :

1. Plus de données (data augmentation)
2. Régularisation L1/L2
3. Dropout (NN)
4. Early stopping
5. Réduire complexité modèle
6. Ensembling

7 Déploiement

7.1 Sauvegarder le modèle

Listing 4 – Persistence avec joblib

```python
import joblib

# Sauvegarder
joblib.dump(model, 'model.pkl')
joblib.dump(scaler, 'scaler.pkl')

# Charger
model = joblib.load('model.pkl')
scaler = joblib.load('scaler.pkl')
```

7.2 API REST avec FastAPI

Listing 5 – API simple

```python
from fastapi import FastAPI
import joblib
import numpy as np

app = FastAPI()
model = joblib.load('model.pkl')

@app.post("/predict")
def predict(features: list[float]):
    X = np.array(features).reshape(1, -1)
    prediction = model.predict(X)[0]
    proba = model.predict_proba(X)[0].tolist()
    return {"prediction": int(prediction), "probabilities": proba}

# Lancer : uvicorn main:app --reload
```

7.3 Monitoring

— Tracker performance en production (drift)
— Logger prédictions et features
— Alertes si dégradation
— Réentraîner périodiquement

8 Reproductibilité

8.1 Random seeds

Listing 6 – Fixer tous les seeds

```python
import random
import numpy as np
import torch

SEED = 42
random.seed(SEED)
np.random.seed(SEED)
```

```
8 torch.manual_seed(SEED)
9 torch.cuda.manual_seed_all(SEED)
```

8.2 Version control

— Code : Git
— Données : DVC (Data Version Control)
— Modèles : MLflow, Weights & Biases
— Environnement : Docker, requirements.txt, conda env

9 Checklist Production

☐ Modèle validé sur test set indépendant
☐ Pipeline complet (preprocessing + model) sauvegardé
☐ Tests unitaires sur preprocessing
☐ Logging configuré
☐ Monitoring des performances
☐ Documentation : features, métriques, limites
☐ Fallback si modèle échoue
☐ Plan de réentraînement

10 Pièges courants

10.1 Data Leakage

À Éviter

Exemples de leakage :
— Scaler fit sur toutes les données (inclut test)
— Features dérivées du target
— Features du futur dans séries temporelles
— Duplicatas entre train et test

10.2 Optimisation sur test set

À Éviter

NE JAMAIS :
— Tuner hyperparamètres sur test set
— Regarder test set avant validation finale
— Réentraîner après avoir vu test performance

Le test set doit rester **inviolé** jusqu'à l'évaluation finale !

10.3 Sélection de features sur tout le dataset

Sélectionner features UNIQUEMENT sur train set, puis appliquer à val/test.

11 Ressources

11.1 Outils

— **scikit-learn** : ML classique
— **PyTorch/TensorFlow** : Deep Learning
— **XGBoost/LightGBM/CatBoost** : Gradient Boosting
— **Optuna** : Hyperparameter tuning
— **MLflow** : Experiment tracking
— **FastAPI** : API deployment
— **Docker** : Containerization

11.2 Lectures

— Géron - *Hands-On Machine Learning*
— Hastie et al. - *The Elements of Statistical Learning*
— Chollet - *Deep Learning with Python*

12 Résumé

12.1 Règles d'or

1. Toujours commencer simple (baseline)
2. Train/val/test splits stricts
3. Pas de data leakage
4. Cross-validation systématique
5. Métriques adaptées au problème
6. Test set : une seule évaluation finale
7. Reproductibilité (seeds, versions)
8. Monitoring en production

12.2 Workflow résumé

$$\boxed{\text{EDA}} \rightarrow \boxed{\text{Preprocessing}} \rightarrow \boxed{\text{Baseline}} \rightarrow \boxed{\text{Tuning}} \rightarrow \boxed{\text{Test}} \rightarrow \boxed{\text{Deploy}}$$

13 Notebooks Pratiques

Ce chapitre est accompagné des notebooks suivants :

— $14_demo_pipeline_complet.ipynb$: $PipelineMLdeboutenbout$

  — Exploration des données (EDA)
  — Feature engineering et preprocessing
  — Sélection de modèle avec validation croisée
  — Pipeline scikit-learn complet

$14_demo_deployment_fastapi.ipynb$: $DploiementavecFastAPI$

`Création d'une API REST`

`Validation avec Pydantic`

`Conteneurisation Docker`

`Tests et documentation automatique`

`14`$demo_monitoring_mlflow.ipynb : Tracking et monitoring avec MLflow$

`Configuration MLflow Tracking Server`

`Logging des hyperparamètres et métriques`

`Model Registry`

`Comparaison de runs et reproductibilité`

Fin du Cours Machine Learning

Félicitations ! Vous maîtrisez maintenant les fondamentaux et techniques avancées du ML.

Continuez à pratiquer sur des projets réels et à vous tenir à jour avec les dernières avancées.