Cours Machine Learning

Annexes
Aide-mémoire, Glossaire, Ressources

Contenu :

— Aide-mémoire : Formules essentielles
— Glossaire : Terminologie ML
— Ressources : Bibliothèques, datasets, liens
— FAQ : Pièges courants et solutions

Cours ML - Sandbox-ML
Version 1.0 - 2026

Table des matières

1 Aide-Mémoire : Formules Essentielles

1.1 Algèbre Linéaire

Produit Matriciel

$$\mathbf{C} = \mathbf{AB} \quad \text{où } C_{ij} = \sum_k A_{ik} B_{kj}$$

$$\text{Dimensions : } (m \times n) \cdot (n \times p) = (m \times p)$$

Valeurs Propres

$$\mathbf{Av} = \lambda \mathbf{v} \quad \Leftrightarrow \quad \det(\mathbf{A} - \lambda \mathbf{I}) = 0$$

SVD (Singular Value Decomposition)

$$\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T$$

où $\mathbf{U}, \mathbf{V}$ orthogonales, $\mathbf{\Sigma}$ diagonale (valeurs singulières)

1.2 Probabilités

Théorème de Bayes

$$P(A|B) = \frac{P(B|A) \cdot P(A)}{P(B)}$$

Loi Normale

$$\mathcal{N}(\mu, \sigma^2) : \quad f(x) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)$$

1.3 Régression

Régression Linéaire

$$\text{Modèle : } \hat{y} = \mathbf{w}^T \mathbf{x} + b$$

$$\text{Solution : } \mathbf{w}^* = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{y}$$

$$\text{MSE : } \frac{1}{m} \sum_{i=1}^{m} (y_i - \hat{y}_i)^2$$

Ridge (L2)

$$\mathbf{w}^* = (\mathbf{X}^T \mathbf{X} + \lambda \mathbf{I})^{-1} \mathbf{X}^T \mathbf{y}$$

Lasso (L1)

$$\min_{\mathbf{w}} \|\mathbf{X}\mathbf{w} - \mathbf{y}\|_2^2 + \lambda\|\mathbf{w}\|_1$$

1.4 Classification

Logistic Regression

$$\sigma(z) = \frac{1}{1+e^{-z}}$$
$$P(y=1|\mathbf{x}) = \sigma(\mathbf{w}^T\mathbf{x} + b)$$
$$\text{Loss} : -\frac{1}{m}\sum_{i=1}^{m}[y_i \log \hat{y}_i + (1-y_i)\log(1-\hat{y}_i)]$$

Softmax

$$P(y=k|\mathbf{x}) = \frac{e^{z_k}}{\sum_{j=1}^{K} e^{z_j}}$$

1.5 Métriques

Classification

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{Total}}$$
$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}$$
$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$
$$\text{F1-score} = 2 \cdot \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

1.6 Clustering

K-Means

$$\min_{\mu_1,\ldots,\mu_K} \sum_{i=1}^{n} \min_{k=1,\ldots,K} \|\mathbf{x}_i - \mu_k\|^2$$

1.7 Réseaux de Neurones

Forward Pass

$$\mathbf{z}^{[l]} = \mathbf{W}^{[l]}\mathbf{a}^{[l-1]} + \mathbf{b}^{[l]}$$
$$\mathbf{a}^{[l]} = \sigma^{[l]}(\mathbf{z}^{[l]})$$

Backpropagation

$$\delta^{[l]} = \frac{\partial L}{\partial \mathbf{z}^{[l]}}$$
$$\frac{\partial L}{\partial \mathbf{W}^{[l]}} = \delta^{[l]} (\mathbf{a}^{[l-1]})^T$$
$$\frac{\partial L}{\partial \mathbf{b}^{[l]}} = \delta^{[l]}$$

Optimiseurs

$$\text{SGD} : \theta \leftarrow \theta - \alpha \nabla L$$
$$\text{Momentum} : \mathbf{v} = \beta \mathbf{v} + (1-\beta)\nabla L, \quad \theta \leftarrow \theta - \alpha \mathbf{v}$$
$$\text{Adam} : \text{Combine momentum + RMSprop}$$

1.8 CNN

Convolution 2D

$$\text{Output}[i,j] = \sum_m \sum_n \text{Input}[i+m, j+n] \cdot \text{Kernel}[m,n]$$
$$\text{Taille sortie} : \left\lfloor \frac{H + 2p - k}{s} \right\rfloor + 1$$

1.9 RNN/LSTM

RNN

$$\mathbf{h}_t = \tanh(\mathbf{W}_{hh}\mathbf{h}_{t-1} + \mathbf{W}_{xh}\mathbf{x}_t + \mathbf{b})$$

LSTM (simplifié)

$$\mathbf{f}_t = \sigma(\mathbf{W}_f[\mathbf{h}_{t-1}, \mathbf{x}_t]) \quad \text{(forget)}$$
$$\mathbf{i}_t = \sigma(\mathbf{W}_i[\mathbf{h}_{t-1}, \mathbf{x}_t]) \quad \text{(input)}$$
$$\mathbf{c}_t = \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \tilde{\mathbf{c}}_t$$

Attention

$$\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d_k}}\right)\mathbf{V}$$

1.10 Reinforcement Learning

Q-Learning

$$Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma \max_{a'} Q(s',a') - Q(s,a)]$$

2 Glossaire

Terme	Définition
Accuracy	Proportion de prédictions correctes
Activation Function	Fonction non-linéaire appliquée aux neurones (ReLU, sigmoid, etc.)
Adam	Optimiseur adaptatif combinant momentum et RMSprop
Backpropagation	Algorithme pour calculer gradients dans réseaux de neurones
Batch	Sous-ensemble de données utilisé pour une mise à jour
Batch Normalization	Normalisation des activations par batch
Bias	Biais (intercept) ou biais statistique (erreur systématique)
Bias-Variance Tradeoff	Compromis entre underfitting et overfitting
BPTT	Backpropagation Through Time (pour RNN)
CNN	Convolutional Neural Network (réseau convolutif)
Confusion Matrix	Tableau TP/TN/FP/FN pour classification
Cross-Entropy	Fonction de perte pour classification
Cross-Validation	Validation croisée (K-Fold)
Data Augmentation	Augmentation artificielle du dataset
Data Leakage	Fuite d'info du test vers train (erreur grave)
Dropout	Désactivation aléatoire de neurones (régularisation)
Early Stopping	Arrêt si validation loss n'améliore plus
Embedding	Représentation vectorielle dense (mots, etc.)
Ensemble	Combinaison de plusieurs modèles
Epoch	Une passe complète sur le dataset d'entraînement
F1-Score	Moyenne harmonique de Precision et Recall
Feature Engineering	Création/transformation de features
Fine-Tuning	Ajustement d'un modèle pré-entraîné
Gradient Descent	Optimisation itérative par descente de gradient
GRU	Gated Recurrent Unit (variante LSTM)
Hyperparameter	Paramètre fixé avant entraînement (LR, etc.)
L1/L2 Regularization	Pénalisation sur poids (Lasso/Ridge)
Learning Rate	Taille du pas de gradient descent
LSTM	Long Short-Term Memory (RNN avec mémoire)
MLP	Multi-Layer Perceptron (réseau fully-connected)
MSE	Mean Squared Error
Overfitting	Modèle trop complexe, bon sur train, mauvais sur test
PCA	Principal Component Analysis (réduction dim)
Pooling	Sous-échantillonnage (Max/Average)
Precision	TP / (TP + FP)

Terme	Définition
Recall	TP / (TP + FN)
Regularization	Techniques contre overfitting (L1, L2, dropout)
ReLU	Rectified Linear Unit : $\max(0, x)$
RNN	Recurrent Neural Network (séquences)
SGD	Stochastic Gradient Descent
Softmax	Fonction de sortie pour multi-classe
Transfer Learning	Réutilisation modèle pré-entraîné
Transformer	Architecture attention pour NLP
Underfitting	Modèle trop simple
Vanishing Gradient	Gradients $\to 0$ dans réseaux profonds
Validation Set	Données pour tuner hyperparamètres

3 Ressources

3.1 Bibliothèques Python

3.1.1 ML Classique

- **scikit-learn** : ML complet (classification, régression, clustering)
- **XGBoost** : Gradient Boosting haute performance
- **LightGBM** : Gradient Boosting rapide (Microsoft)
- **CatBoost** : Gradient Boosting pour catégorielles (Yandex)

3.1.2 Deep Learning

- **PyTorch** : Framework DL flexible (recherche)
- **TensorFlow/Keras** : Framework DL production
- **FastAI** : Haut niveau sur PyTorch

3.1.3 NLP

- **Transformers (HuggingFace)** : BERT, GPT, T5, etc.
- **spaCy** : NLP industriel
- **NLTK** : NLP pédagogique

3.1.4 Data Science

- **NumPy** : Calcul numérique
- **Pandas** : Manipulation données tabulaires
- **Matplotlib/Seaborn** : Visualisation
- **Plotly** : Visualisation interactive

3.1.5 Outils

- **Jupyter** : Notebooks interactifs
- **MLflow** : Tracking expériences
- **Optuna** : Hyperparameter tuning
- **Weights & Biases** : Expérience tracking
- **DVC** : Version control données

3.2 Datasets

3.2.1 Débutant

— **MNIST** : Chiffres manuscrits (10 classes, 60K images)
— **Iris** : Classification florale (150 samples, 4 features)
— **Titanic** : Survie passagers (Kaggle)
— **Boston Housing** : Prédiction prix immobilier

3.2.2 Intermédiaire

— **CIFAR-10/100** : Images couleur (10/100 classes)
— **IMDB Reviews** : Sentiment analysis
— **Fashion-MNIST** : Vêtements (alternative MNIST)

3.2.3 Avancé

— **ImageNet** : 1.2M images, 1000 classes
— **COCO** : Détection objets, segmentation
— **SQuAD** : Question answering
— **Kaggle Competitions** : Datasets réels variés

3.3 Environnements RL

— **OpenAI Gym** : Environnements RL standards
— **Stable Baselines3** : Implémentations RL
— **PettingZoo** : Multi-agent RL

3.4 Cours en Ligne

— **Fast.ai** : Deep Learning pratique
— **CS231n (Stanford)** : CNN pour vision
— **CS224n (Stanford)** : NLP avec Deep Learning
— **Coursera - Andrew Ng** : ML et DL
— **DeepLearning.AI** : Spécialisations DL

3.5 Livres

— Géron - *Hands-On Machine Learning (3rd ed, 2023)*
— Goodfellow et al. - *Deep Learning*
— Bishop - *Pattern Recognition and Machine Learning*
— Hastie et al. - *The Elements of Statistical Learning*
— Sutton & Barto - *Reinforcement Learning*
— Chollet - *Deep Learning with Python (2nd ed)*

4 FAQ : Pièges Courants

4.1 Données

Q : Dois-je normaliser mes données ?

A : Oui, pour la plupart des algorithmes (SVM, réseaux de neurones, K-Means). Non pour les arbres de décision. Utiliser StandardScaler ou MinMaxScaler selon le contexte.

Q : Comment gérer les valeurs manquantes ?

A : (1) Supprimer si $< 5\%$, (2) Imputer médiane/moyenne pour numériques, (3) Mode pour catégorielles, (4) Créer indicateur "missing".

Q : Qu'est-ce que le data leakage ?

A : Utiliser des informations du test set pendant l'entraînement. Erreurs courantes :
- Scaler fit sur toutes les données
- Features du futur dans séries temporelles
- Duplicatas entre train et test

4.2 Modélisation

Q : Mon modèle a 99% d'accuracy mais ne marche pas bien ?

A : Classes probablement déséquilibrées. Utiliser F1-score, Precision/Recall ou AUC-ROC au lieu d'accuracy.

Q : Overfitting vs Underfitting ?

A :
- **Overfitting** : Train error faible, val error élevé → Régularisation, plus de données
- **Underfitting** : Train et val errors élevés → Modèle plus complexe

Q : Combien de données faut-il ?

A : Règle empirique :
- ML classique : $10\times$ le nombre de features minimum
- Deep Learning : 1000+ par classe minimum
- Transfer Learning : 100+ par classe peut suffire

4.3 Entraînement

Q : Mon loss est NaN ?

A : (1) Learning rate trop élevé, (2) Explosion gradients → gradient clipping, (3) Données non normalisées.

Q : Mon modèle n'apprend pas (loss stagne) ?

A : (1) Learning rate trop faible, (2) Mauvaise initialisation, (3) Dead ReLU (tous neurones inactifs).

Q : Quelle taille de batch ?

A : 32-128 généralement. Plus grand = plus rapide mais moins de généralisation. Plus petit = meilleure généralisation mais plus lent.

4.4 Évaluation

Q : Puis-je utiliser le test set plusieurs fois ?

A : **NON !** Le test set ne doit être utilisé qu'**une seule fois** pour l'évaluation finale. Utiliser validation set pour tuning.

Q : Cross-validation ou simple train/val/test ?

A :
- CV : Petites données (< 10K), évaluation robuste
- Train/Val/Test : Grandes données, DL

4.5 Production

Q : Comment déployer mon modèle ?

A : (1) Sauvegarder avec joblib/pickle, (2) API REST (FastAPI/Flask), (3) Containeriser (Docker), (4) Monitoring.

Q : Mon modèle se dégrade en production ?

A : Distribution shift (data drift). Solution : Monitoring, réentraînement périodique, détection d'anomalies.

5 Commandes Python Essentielles

5.1 scikit-learn Workflow

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report

# Split
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42, stratify=y
)

# Scale
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)  # Pas de fit!

# Train
model = RandomForestClassifier(n_estimators=100, random_state=42)
model.fit(X_train_scaled, y_train)

# Evaluate
y_pred = model.predict(X_test_scaled)
print(classification_report(y_test, y_pred))
```

5.2 PyTorch Workflow

```
import torch
import torch.nn as nn
import torch.optim as optim

# Model
model = nn.Sequential(
    nn.Linear(input_dim, 128),
    nn.ReLU(),
```

```
    nn.Dropout(0.2),
    nn.Linear(128, output_dim)
)

# Loss et optimizer
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)

# Training loop
for epoch in range(num_epochs):
    model.train()
    for X_batch, y_batch in train_loader:
        optimizer.zero_grad()
        outputs = model(X_batch)
        loss = criterion(outputs, y_batch)
        loss.backward()
        optimizer.step()
```

6 Annexe D : Cheat Sheets par Framework

6.1 scikit-learn : Workflow Complet

```
# Pipeline scikit-learn classique
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report

# 1. Split des donn es
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42, stratify=y
)

# 2. Pipeline avec preprocessing
pipeline = Pipeline([
    ('scaler', StandardScaler()),
    ('clf', RandomForestClassifier(random_state=42))
])

# 3. Grid Search pour hyperparameters
param_grid = {
    'clf__n_estimators': [100, 200, 300],
    'clf__max_depth': [10, 20, None],
    'clf__min_samples_split': [2, 5, 10]
}

grid = GridSearchCV(pipeline, param_grid, cv=5,
                    scoring='f1_weighted', n_jobs=-1)
grid.fit(X_train, y_train)

# 4. Evaluation
print(f"Best␣params:␣{grid.best_params_}")
print(f"Best␣CV␣score:␣{grid.best_score_:.3f}")
```

```
y_pred = grid.predict(X_test)
print(classification_report(y_test, y_pred))
```

6.2 PyTorch : Training Loop Standard

```
# Training loop PyTorch typique
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import DataLoader, TensorDataset

# 1. Pr paration des donn es
train_dataset = TensorDataset(
    torch.tensor(X_train, dtype=torch.float32),
    torch.tensor(y_train, dtype=torch.long)
)
train_loader = DataLoader(train_dataset, batch_size=32, shuffle=True)

# 2. D finition du mod le
model = nn.Sequential(
    nn.Linear(input_dim, 128),
    nn.ReLU(),
    nn.Dropout(0.3),
    nn.Linear(128, 64),
    nn.ReLU(),
    nn.Linear(64, num_classes)
)

# 3. Loss et optimizer
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)
scheduler = optim.lr_scheduler.ReduceLROnPlateau(optimizer, patience=5)

# 4. Training loop
for epoch in range(num_epochs):
    model.train()
    total_loss = 0
    for batch_X, batch_y in train_loader:
        optimizer.zero_grad()
        outputs = model(batch_X)
        loss = criterion(outputs, batch_y)
        loss.backward()
        optimizer.step()
        total_loss += loss.item()

    # Validation
    model.eval()
    with torch.no_grad():
        val_outputs = model(X_val)
        val_loss = criterion(val_outputs, y_val)

    scheduler.step(val_loss)
    print(f"Epoch {epoch+1}/{num_epochs} | "
          f"Train Loss: {total_loss/len(train_loader):.4f} | "
          f"Val Loss: {val_loss:.4f}")
```

6.3 Hugging Face Transformers : Fine-tuning

```
# Fine-tuning BERT avec Hugging Face
from transformers import (BertTokenizer, BertForSequenceClassification,
                          Trainer, TrainingArguments)

# 1. Tokenizer et mod le
tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')
model = BertForSequenceClassification.from_pretrained(
    'bert-base-uncased', num_labels=2
)

# 2. Tokenization
train_encodings = tokenizer(train_texts, truncation=True,
                            padding=True, max_length=128)
val_encodings = tokenizer(val_texts, truncation=True,
                          padding=True, max_length=128)

# 3. Dataset PyTorch
class CustomDataset(torch.utils.data.Dataset):
    def __init__(self, encodings, labels):
        self.encodings = encodings
        self.labels = labels

    def __getitem__(self, idx):
        item = {key: torch.tensor(val[idx])
                for key, val in self.encodings.items()}
        item['labels'] = torch.tensor(self.labels[idx])
        return item

    def __len__(self):
        return len(self.labels)

train_dataset = CustomDataset(train_encodings, train_labels)
val_dataset = CustomDataset(val_encodings, val_labels)

# 4. Training arguments
training_args = TrainingArguments(
    output_dir='./results',
    num_train_epochs=3,
    per_device_train_batch_size=16,
    per_device_eval_batch_size=16,
    warmup_steps=500,
    weight_decay=0.01,
    logging_dir='./logs',
    logging_steps=100,
    evaluation_strategy="epoch",
    save_strategy="epoch",
    load_best_model_at_end=True
)

# 5. Trainer
trainer = Trainer(
```

```
    model=model,
    args=training_args,
    train_dataset=train_dataset,
    eval_dataset=val_dataset
)

# 6. Fine-tuning
trainer.train()
```

7 Annexe E : Guide Hardware et Cloud

7.1 Choix de GPU pour ML/DL

GPU	VRAM	Prix	Use Case	Performance
GTX 1660 Ti	6 GB	$280	Apprentissage	Entrée de gamme
RTX 3060	12 GB	$330	ML léger	Bon rapport qualité/prix
RTX 3080	10 GB	$700	DL moyen	Très bon pour entraînement
RTX 4090	24 GB	$1600	DL lourd	Top performance
A100 (40GB)	40 GB	$10k+	Production	Datacenters

Table 2 – Comparaison GPUs pour Machine Learning (2024)

Recommandations selon budget :

- **Budget < 500$** : RTX 3060 (12GB) - Excellent pour débuter
- **Budget 500-1000$** : RTX 3080/4070 - Bon pour projets sérieux
- **Budget 1000-2000$** : RTX 4090 - Top pour recherche/production
- **Budget illimité** : Multiple A100 ou H100 - Datacenters

VRAM nécessaire selon modèle :

- ResNet-50 : 4-6 GB
- BERT-base : 8-12 GB
- BERT-large : 16-20 GB
- GPT-2 (1.5B) : 20-24 GB
- Fine-tuning LLaMA 7B : 40+ GB (ou quantization)

7.2 Plateformes Cloud pour ML

Plateforme	GPU disponibles	Prix/h	Avantages
Google Colab Pro+	T4, A100	Gratuit-$50/mois	Simple, notebooks
AWS SageMaker	P3, P4	$3-10/h	Complet, scalable
Google Cloud AI	V100, A100, TPU	$2-8/h	TPUs puissants
Azure ML	NC, ND series	$2-10/h	Intégration Microsoft
Lambda Labs	A100, H100	$1-3/h	Moins cher
Paperspace Gradient	RTX 4000-A100	$0.5-3/h	Simple, abordable

Table 3 – Comparaison plateformes Cloud ML (2024)

Recommandations :

- **Débutants** : Google Colab Pro (gratuit tier généreux)
- **Prototyping** : Paperspace Gradient (rapport qualité/prix)
- **Production** : AWS SageMaker ou GCP AI Platform (robustesse)
- **Recherche** : Lambda Labs (GPUs puissants, moins cher)

8 Annexe F : Carrières en Machine Learning

8.1 Parcours Professionnels

1. ML Engineer
- **Rôle** : Développer et déployer des modèles ML en production
- **Compétences** : Python, ML libs, MLOps, CI/CD, cloud
- **Salaire (France)** : 45k-70k€ (junior), 70k-110k€ (senior)
- **Demande** : Très forte (croissance 40%/an)

2. Data Scientist
- **Rôle** : Analyser données, créer modèles prédictifs, insights business
- **Compétences** : Stats, Python/R, ML, visualisation, communication
- **Salaire** : 40k-65k€ (junior), 65k-100k€ (senior)
- **Demande** : Forte mais saturation dans certaines régions

3. Research Scientist (ML/DL)
- **Rôle** : Recherche fondamentale, publications, nouveaux algorithmes
- **Compétences** : PhD, mathématiques avancées, PyTorch, publications
- **Salaire** : 60k-90k€ (postdoc), 90k-150k€+ (senior, FAANG)
- **Demande** : Modérée mais très compétitive

4. MLOps Engineer
- **Rôle** : Infrastructure ML, CI/CD, monitoring, scalabilité
- **Compétences** : DevOps, Kubernetes, Docker, ML frameworks
- **Salaire** : 50k-75k€ (junior), 75k-120k€ (senior)
- **Demande** : En forte croissance (nouveau métier)

8.2 Certifications Recommandées

Certifications Cloud ML :
- **AWS Certified Machine Learning - Specialty** (difficulté : élevée)
- **Google Cloud Professional ML Engineer** (difficulté : élevée)
- **Microsoft Azure AI Engineer Associate** (difficulté : moyenne)

Certifications Académiques :
- **deeplearning.ai Specializations** (Coursera) : 5 spécialisations excellentes
- **fast.ai Practical Deep Learning** : Gratuit, très pratique
- **Stanford CS229 (Machine Learning)** : Théorique et rigoureux

8.3 Roadmap de Progression

Niveau Débutant (0-6 mois) :

1. Maîtriser Python et NumPy/Pandas
2. Comprendre ML supervisé (regression, classification)
3. Implémenter algorithmes from scratch
4. Compléter ce cours ML (chapitres 00-05)

Niveau Intermédiaire (6-18 mois) :

1. Deep Learning (PyTorch ou TensorFlow)
2. Projets Kaggle (top 25%)
3. Compléter chapitres 06-10 de ce cours
4. Contribuer à projets open-source

Niveau Avancé (18+ mois) :

1. Spécialisation (NLP, CV, RL, etc.)
2. Publications ou projets significatifs
3. MLOps et déploiement production
4. Compléter chapitres 11-14 + applications avancées

9 Annexe G : Outils et Librairies Essentiels

9.1 Stack ML Complète

Data Processing :
- **Pandas** : Manipulation de DataFrames
- **Polars** : Alternative ultra-rapide à Pandas
- **Dask** : DataFrames distribués pour big data
- **PySpark** : Processing distribué à grande échelle

Machine Learning :
- **scikit-learn** : ML classique (must-have)
- **XGBoost, LightGBM, CatBoost** : Gradient boosting
- **PyTorch** : Deep Learning (recommandé)
- **TensorFlow/Keras** : Deep Learning (alternative)
- **Hugging Face Transformers** : NLP state-of-the-art

Visualisation :
- **Matplotlib** : Base, flexible
- **Seaborn** : Statistical plots, esthétique
- **Plotly** : Interactif, dashboards
- **Weights & Biases** : Experiment tracking

MLOps :
- **MLflow** : Experiment tracking, model registry
- **DVC** : Version control pour données/modèles
- **Docker** : Conteneurisation
- **Kubernetes** : Orchestration à grande échelle
- **FastAPI** : APIs REST performantes

9.2 Commandes Essentielles

9.2.1 Installation Rapide

```
# ML Stack minimal
pip install numpy pandas scikit-learn matplotlib

# Deep Learning (PyTorch)
pip install torch torchvision torchaudio

# Deep Learning (TensorFlow)
pip install tensorflow

# NLP moderne
pip install transformers datasets tokenizers

# MLOps
pip install mlflow dvc fastapi uvicorn

# Gradient Boosting
```

```
pip install xgboost lightgbm catboost
```

9.2.2 Vérifier GPU PyTorch

```
import torch
print(f"CUDA␣disponible␣:␣{torch.cuda.is_available()}")
print(f"GPU␣:␣{torch.cuda.get_device_name(0)}")
print(f"VRAM␣:␣{torch.cuda.get_device_properties(0).total_memory␣/␣1e9:.2f}␣
    GB")
```

Conclusion

Ce document d'annexes complète le cours complet de Machine Learning. Utilisez-le comme référence rapide pendant vos projets.

Points clés à retenir :

- Toujours commencer simple (baseline)
- Éviter le data leakage à tout prix
- Choisir la bonne métrique selon le problème
- Valider rigoureusement (CV ou train/val/test)
- Test set = une seule utilisation finale
- Documenter et versionner (code, données, modèles)

Bon apprentissage et bons projets ML !