Sklearn, Généralisation & Overfitting

Jour 2 — Après-midi

Julien Rolland

Formation M2 Développement Fullstack

Jour 2

Plan du module

Le Vrai Objectif : Généraliser

Le Piège

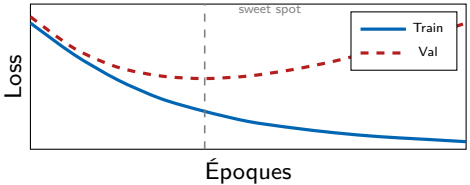
- Minimiser $J(\Theta, X_{\text{train}})$ = problème résolu
- La performance sur le train **ne compte pas**

Ce qu'on veut vraiment

- Données **non-vues** (unseen data)
- Capter la loi sous-jacente
- Ignorer le bruit du dataset

Définition

Généralisation : capacité à performer sur des exemples jamais vus pendant l'entraînement.



Underfitting vs Overfitting

Underfitting

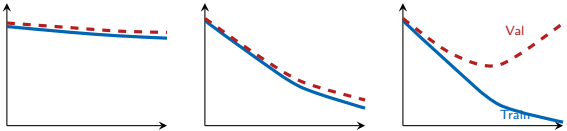
- Modèle trop **simple** — biais élevé
- Train $\approx$ Val $\Rightarrow$ toutes deux élevées

Bon fit

- Complexité adaptée aux données
- Train $\approx$ Val $\Rightarrow$ toutes deux basses

Overfitting

- Modèle trop **complexe** — variance élevée
- Train $\ll$ Val $\Rightarrow$ Val explose

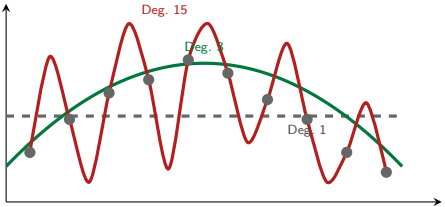


Théorème de Cover

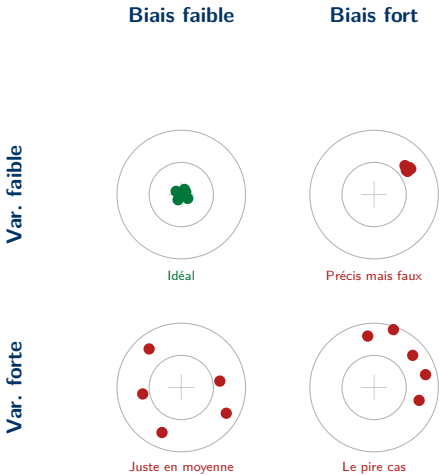
- Dans $\mathbb{R}^D$, $N \leq D$ points sont toujours linéairement séparables
- Un modèle assez complexe peut **mémoriser** n'importe quel dataset
- Accuracy train $= 100\%$ ne signifie rien

Analogie MNIST

- Mémoriser les pixels exacts de chaque «3»
- Incapable de reconnaître un «3» écrit avec un stylo différent
- $\Rightarrow$ 0% de généralisation



Le Compromis Biais-Variance



En Machine Learning

- **Biais** : erreur systématique
  modèle trop simple
- **Variance** : sensibilité aux données
  modèle trop complexe

$$\text{Erreur} = \text{Biais}^2 + \text{Variance} + \varepsilon$$

Leviers

- **Complexité** du modèle
- **Régularisation** (λ)
- **Volume** de données

Le Split Standard

3 ensembles distincts

1. **Train** — ajuster les paramètres Θ
2. **Validation** — choix du modèle et des hyperparamètres
3. **Test** — mesure finale

Règle d'or : Data Leakage

Ne **jamais** entraîner sur le test set.
Ne **jamais** utiliser le test pour choisir les HP.
Le test set doit rester **invisible**.

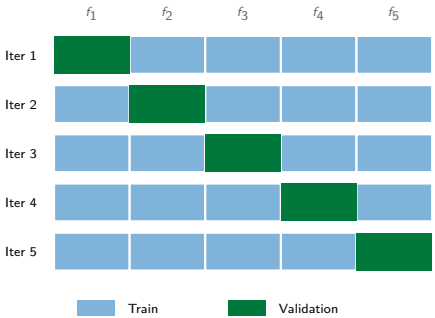
K-Fold Cross-Validation

Le Problème

- Peu de données $\Rightarrow$ split instable
- Un seul val set $\Rightarrow$ variance élevée de l'estimation

Solution : K-Fold

- Découper le train en K morceaux (*folds*)
- K rotations : chaque fold sert de validation
- Score final = moyenne sur K runs
- Utilisation maximale des données disponibles



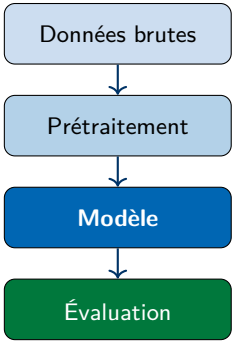
$$s = \frac{1}{K} \sum_{k=1}^{K} s_k$$

La bibliothèque ML de référence

- Open-source, basée sur NumPy / SciPy
- Standard de l'industrie pour le ML classique
- Interface cohérente pour tous les algorithmes

Ce qu'elle couvre

- **Prétraitement** : scaling, encodage, imputation
- **Modèles** : régression, classification, clustering
- **Validation** : cross-validation, grid search
- **Métriques** : accuracy, F1, AUC, ...



L'API Estimator

Interface universelle

- `.fit(X, y)` — entraînement
- `.predict(X)` — inférence
- `.score(X, y)` — évaluation
- `.transform(X)` — transformation

Modèles clés

- `LogisticRegression`
- `LinearRegression`
- `StandardScaler`
- `train_test_split`

```python
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
X_tr = scaler.fit_transform(X_train)
X_te = scaler.transform(X_test)

clf = LogisticRegression()
clf.fit(X_tr, y_train)
print(clf.score(X_te, y_test))
```

Pipeline

Principe

- Enchaîner Scaler → Modèle
- `.fit()` appliqué séquentiellement
- Compatible avec la cross-validation

Sans Pipeline

Scaler fitté sur train+val
⇒ **Data Leakage !**

```python
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression

pipe = Pipeline([
    ('scaler', StandardScaler()),
    ('clf',    LogisticRegression()),
])

pipe.fit(X_train, y_train)
print(pipe.score(X_test, y_test))
```

train_test_split & cross_val_score

train_test_split

- Découpe aléatoire du dataset
- `test_size` : proportion du test set
- `random_state` : reproductibilité
- `stratify` : préserve les proportions de classes

cross_val_score

- K-Fold intégré, compatible Pipeline
- Retourne un score par fold
- `cv` : nombre de folds
- `scoring` : métrique (`'accuracy'`, ...)

```python
from sklearn.model_selection import train_test_split, cross_val_score

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

scores = cross_val_score(pipe, X_train, y_train, cv=5)
print(f"CV: {scores.mean():.3f} +/- {scores.std():.3f}")
```