

Scikit-learn

Sommaire

- Introduction
- Concepts fondamentaux
- Prétraitement des données
- Apprentissage supervisé - Régression & Classification
- Validation et optimisation

Introduction

Qu'est-ce que scikit-learn ?

- Scikit-learn est la bibliothèque Python de référence pour le machine learning. Elle offre une interface simple et cohérente pour appliquer des algorithmes d'apprentissage automatique.
- **Points clés :**
 - Open-source et gratuite
 - API uniforme et intuitive
 - Documentation exhaustive
 - Intégration parfaite avec Numpy, pandas et matplotlib.

Qu'est-ce que scikit-learn ?

- **Installation :**

```
pip install scikit-learn
```

- **Dépendances :** Numpy, SciPy, joblib

- **Philosophie de scikit-learn :**

- **Simplicité :** API cohérente et facile à utiliser
- **Efficacité :** Implémentation optimisées en C/Cython
- **Accessibilité :** Documentation claire avec de nombreux exemples.

Concepts fondamentaux

API uniforme

Tous les algorithmes de scikit-learn suivent la même interface :

- **Estimateurs** : objets qui apprennent à partir des données
 - `fit(X, y)` : Entraîne le modèle
 - `predict(X)` : Faire des prédictions.
 - `score(X, y)` : évaluer les performances.
- **Transformateurs** : objets qui transforment les données.
 - `fit(X)` : Apprendre les paramètres de transformation
 - `transform(X)` : appliquer la transformation.
 - `fit_transform(X)` : fit + transform en une seule étape.

Types d'apprentissage

- **Apprentissage supervisé :**
 - On dispose de données étiquetées (X, y)
 - **Objectif** : Apprendre une fonction f telle que $y = f(X)$
 - **Deux catégories:**
 - **Régression** : Prédire une valeur continue (prix, température...)
 - **Classification** : Prédire une catégorie (spam/non-spam, maladie...)

Types d'apprentissage

- **Apprentissage non supervisé :**
 - Données non étiquetées (Seulement X)
 - **Objectif :** découvrir des structures cachées
 - Applications :
 - **Clustering:** Regrouper des observations similaires
 - **Réduction de dimensionnalité :** Visualiser ou compresser les données.
 - **Détection d'anomalies:** identifier des observations anormales.

Format des données

- **Structure attendue :**

- **x**: Matrice 2D de shape (n_samples, n_features)
 - Chaque ligne = Une observation
 - Chaque colonne = une caractéristique
- **y**: Vecteur 1D de shape (n_samples)
 - Une valeur par observation

- **Conventions :**

- Données numériques (pas de texte brut)
- Pas de valeurs manquantes (NaN à traiter avant)

Workflow typique d'un projet ML

1. **Charger et explorer** les données
2. **Préparer** les données (nettoyage, encodage, normalisation)
3. **Diviser** en ensemble train/test
4. **Entraîner** le modèle
5. **Évaluer** les performances
6. **Optimiser** les hyperparamètres
7. **Déployer** le modèle

Prétraitement des données

Pourquoi prétraiter ?

Les algorithmes ML nécessitent des données propres et formatées :

- Valeurs numérique uniquement
- Pas de valeurs manquantes
- Features sur des échelles comparables (pour certains algos)
- Encodage approprié des variables catégorielles.

Gestion des valeurs manquantes

- Stratégies disponibles :
- **Mean** : Remplacer par la moyenne (Variables continues)
- **Median** : Remplacer par la médiane (robuste aux outliers)
- **Most frequent**: remplacer par la mode (variables catégorielles)
- **Constant**: Remplacer par une valeur fixe (ex: 0, "unknown")

Outil : SimpleImputer

Encodage des variables catégorielles

Problème : Les algorithmes ML ne comprennent que les nombres.

Solutions :

- **LabelEncoder** :
 - Transforme les catégories en nombre (0,1,2,...)
 - /!\ Introduit un ordre artificiel
 - à utiliser principalement pour la variable cible en classification.

Encodage des variables catégorielles

- **OneHotEncoder :**

- Crée une colonne binaire par catégorie
- Pas d'ordre artificiel
- Recommandé pour les features catégorielles
- **Exemple** : couleur [rouge, bleu, vert] => [1,0,0], [0,1,0], [0,0,1]

- **OrdinalEncoder:**

- Comme LabelEncoder mais pour plusieurs colonnes
- à utiliser quand il existe un ordre naturel (ex: petit/moyen/grand)

Normalisation et standardisation

- **Pourquoi** ? Certains algorithmes sont sensibles à l'échelle des features
- **StandardScaler** :
 - Transforme pour avoir moyenne=0 et écart-type=1
 - Formule : $z = (x - \mu) / \sigma$
 - **Usage** : cas général, la plupart des algorithmes

Normalisation et standardisation

MinMaxScaler :

- Met à l'échelle entre 0 et 1 (ou autre intervalle)
- Formule : $x_{\text{scaled}} = (x - \min) / (\max - \min)$
- **Usage** : quand les features sont déjà bornées, réseaux de neurones

RobustScaler :

- Utilise la médiane et les quartiles
- Résistant aux outliers
- **Usage** : présence de valeurs aberrantes importantes

Normalisation et standardisation

MaxAbsScaler :

- Divise par la valeur absolue maximale
- Préserve les zéros (important pour les matrices creuses)

/!\ **Important** : Toujours faire `fit()` sur train et `transform()` sur test (jamais `fit_transform()` sur test).

Pipelines

Concept : Enchaîner plusieurs étapes de prétraitement et modélisation dans un seul objet.

Avantages :

- Code plus propre et maintenable
- Évite les fuites de données (data leakage)
- Facilite la validation croisée
- Simplifie le déploiement

Composants :

- **Pipeline** : chaîne séquentielle d'étapes
- **ColumnTransformer** : applique différentes transformations

Séparation Train/Test

Pourquoi séparer les données ?

La séparation des données en ensembles d'entraînement (train) et de test est **essentielle** pour :

- **Évaluer la capacité de généralisation** : Un modèle doit bien performer sur des données qu'il n'a jamais vues
- **Détecter le surapprentissage (overfitting)** : Si le modèle performe bien sur train mais mal sur test, il a "appris par cœur" les données d'entraînement
- **Obtenir une estimation réaliste des performances** : L'évaluation sur les données d'entraînement donne toujours des résultats trop optimistes

Pourquoi séparer les données ?

Principe fondamental : Ne jamais toucher aux données de test pendant l'entraînement ! Elles servent uniquement à l'évaluation finale.

Comment faire la séparation ?

```
# Séparation basique (80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(
    X, y,
    test_size=0.2,          # 20% pour le test
    random_state=42         # Pour la reproductibilité
)
```

Paramètres importants

```
X_train, X_test, y_train, y_test = train_test_split(  
    X, y,  
    test_size=0.2,           # Proportion du test (0.2 = 20%)  
    random_state=42,         # Graine aléatoire pour reproductibilité  
    stratify=y,              # Préserve la classification  
    shuffle=True             # Mélange avant de séparer (par défaut)  
)
```

Bonnes pratiques :

- Utiliser `stratify=y` en classification pour conserver les proportions de classes
- Définir `random_state` pour avoir des résultats reproductibles

Les modèles

Decision Tree

Principe : Crée un arbre de questions successives pour segmenter les données.

Avantages :

- Facile à comprendre et visualiser
- Ne nécessite pas de normalisation
- Gère les relations non-linéaires
- Gère les variables catégorielles et numériques

Inconvénients :

- Tendance au surapprentissage
- Instable (petites variations de données changent l'arbre)

Random Forest

Principe : Ensemble de nombreux arbres de décision indépendants qui votent pour la prédiction finale.

Avantages :

- Très performant dans la plupart des cas
- Réduit le surapprentissage par rapport à un seul arbre
- Robuste au bruit et outliers
- Importance des variables disponible
- Nécessite peu de préparation des données

Random Forest

Inconvénients :

- Moins interprétable qu'un arbre seul
- Plus lent à entraîner et prédire
- Peut être gourmand en mémoire

Logistic Regression (Régression Logistique)

Principe : Modèle linéaire qui estime la probabilité d'appartenance à une classe via une fonction sigmoïde.

Avantages :

- Simple et rapide
- Interprétable (coefficients = importance des variables)
- Probabilités calibrées
- Fonctionne bien avec des données linéairement séparables
- Peu de paramètres à tuner

Logistic Regression (Régression Logistique)

Inconvénients :

- Suppose une relation linéaire
- Sensible aux outliers
- Nécessite la normalisation des features
- Moins performant sur données complexes

Gradient Boosting (XGBoost, LightGBM)

Principe : Construit séquentiellement des arbres pour corriger les erreurs des précédents.

Avantages :

- Très performant (souvent gagnant de compétitions)
- Gère bien les données manquantes
- Robuste au surapprentissage (si bien paramétré)
- Importance des variables

Gradient Boosting (XGBoost, LightGBM)

Inconvénients :

- Complexe à paramétrer
- Temps d'entraînement long
- Moins interprétable
- Risque d'overfitting si mal configuré

Conseils de Choix de Modèle

1. **Logistic Regression** (classification) ou **Linear Regression** (régression) : baseline simple
2. **Random Forest** : bon compromis performance/simplicité
3. Comparer avec un **Gradient Boosting** si besoin de maximiser la performance

Si peu de données : Logistic Regression, SVM

Si beaucoup de données : Random Forest, Gradient Boosting

Si besoin d'interprétabilité : Logistic Regression, Decision Tree

Conseils de Choix de Modèle

Si données linéaires : Logistic/Linear Regression

Si données non-linéaires : Random Forest, SVM (kernel rbf),
Gradient Boosting

Merci pour votre attention

Des questions ?