Business & Decision

Formation

Machine Learning avec Python : Cours 2

08/11/2023

IAE NANTES
ÉCONOMIE & MANAGEMENT

VOTRE FORMATEUR

Cédric DANGEARD

Consultant Data

cedric.dangeard@businessdecision.com

Business & Decision

MACHINE LEARNING

LES DIFFÉRENTS ALGOS

PRÉSENTATION DE SCIKIT LEARN

SCIKIT LEARN

- Le package Machine Learning de python
  - Libre
  - Orienté objet
  - Autres Bibliothèques ML reprennent la syntaxe

OBJECT - ESTIMATOR

- Méthodes à implémenter
  - fit(X, y)

    Entrainer le modèle.
  - predict(X)

    Prédire sur de Nouvelles données.
  - predict_proba(X)

    Prédire les probabilités pour les modèles de classification.
  - transform(data)

    Transformer les données
  - score(X)

    Évaluer les performances du modèle

ENCODAGE DES VARIABLES QUALITATIVES

LABEL ENCODING

- Fonctionnement
  - ordonner les modalités
  - remplacer les modalités par leur rang
- Exemple
  - variable : couleur
  - modalités : rouge, vert, bleu
  - rang : rouge = 1, vert = 2, bleu = 3
  - transformation : rouge = 1, vert = 2, bleu = 3
- Avantages
  - ne rajoute pas de variables
- Inconvénients
  - ne fonctionne pas avec les arbres de décision
  - ne fonctionne pas avec les modèles linéaires
  - ne fonctionne pas avec les modèles qui utilisent la distance euclidienne

ONE HOT ENCODING

- Fonctionnement
  - créer une variable par modalité
  - remplacer les modalités par 0 ou 1
- Exemple
  - variable : couleur
  - modalités : rouge, vert, bleu
  - transformation : rouge = 1, vert = 0, bleu = 0
  - transformation : rouge = 0, vert = 1, bleu = 0
  - transformation : rouge = 0, vert = 0, bleu = 1
- Avantages
  - fonctionne avec les arbres de décision
  - fonctionne avec les modèles linéaires
  - fonctionne avec les modèles qui utilisent la distance euclidienne
- Inconvénients
  - rajoute des variables

ENCODAGE DES VARIABLES QUALITATIVES

TARGET ENCODING

- Fonctionnement
  - remplacer les modalités par la moyenne de la variable cible
- Exemple
  - variable : couleur
  - modalités : rouge, vert, bleu
  - transformation : rouge = 0.5, vert = 0.3, bleu = 0.2
- Avantages
  - ne rajoute pas de variables
  - fonctionne avec les arbres de décision
  - fonctionne avec les modèles linéaires
  - fonctionne avec les modèles qui utilisent la distance euclidienne
- Inconvénients
  - peut créer du sur-apprentissage

PYTHON

- Pour les données nominales

```
from sklearn.preprocessing import LabelEncoder

le = LabelEncoder()
le.fit_transform(review)
```

```
from pandas import get_dummies

get_dummies(
    review,
    drop_first=True,
    )
```

- Pour les données ordinales

```
from sklearn.preprocessing import OrdinalEncoder

oe = OrdinalEncoder(categories=[['Mauvais', 'Moyen', 'Bon']])
oe.fit_transform(X = review)
```

ARBRE DE DÉCISION - CART

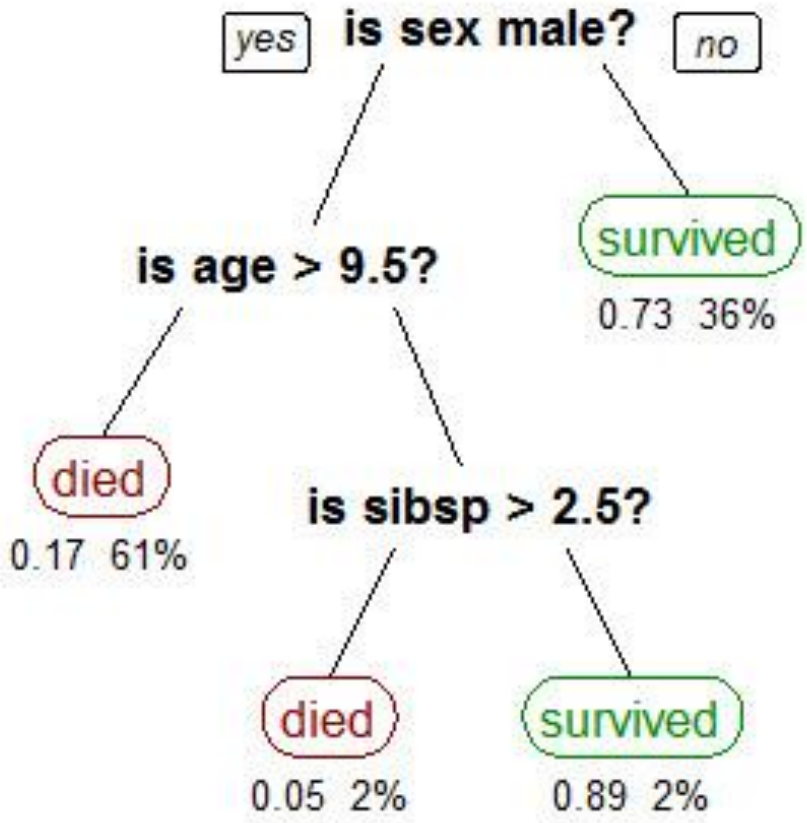
CART est un arbre binaire. Pour chaque nœud, il teste toutes les divisions possibles et garde celle qui donne la meilleure homogénéité du découpage.

Avantages :

- Accepte les variables qualitatives et quantitatives
- Pas besoin de vérifier des hypothèses de normalité et de variance
- Facilement interprétable
- Il répond aux problèmes de classification et régression

Inconvénients :

- Modèle un peu simpliste sur des problèmes de modélisation complexe



ARBRE DE DÉCISION – IMPLÉMENTATION SKLEARN

- criterion :
- splitter :
- max_depth
- min_samples_split
- min_samples_leaf :
- max_features :

```python
from sklearn.tree import DecisionTreeClassifier

tree = DecisionTreeClassifier(
    criterion='entropy',
    max_depth=None,
    min_samples_split=2,
    min_samples_leaf=1,
    max_features="log2",
    random_state=0
)
tree.fit(X, y)
```

ARBRE DE DÉCISION – IMPLÉMENTATION SKLEARN

- criterion : Mesure de la qualité d'un split.
  - Gini, Entropy
- splitter : Choix du split.
  - Best, random (selon la distribution)
- max_depth : Profondeur maximale de l'arbre.
- min_samples_split : Nombre minimum d'individus pour créer un nœud.
- min_samples_leaf : Nombre minimum d'individus pour créer une feuille.
- max_features : Nombre maximum de variables à tester pour créer un nœud.

```python
from sklearn.tree import DecisionTreeClassifier

tree = DecisionTreeClassifier(
    criterion='entropy',
    max_depth=None,
    min_samples_split=2,
    min_samples_leaf=1,
    max_features="log2",
    random_state=0
)
tree.fit(X, y)
```

BAGGING

Le Bagging (***b***ootstrap ***agg***regat***ing***) est une méthode qui consiste à sous-échantillonner les données d'apprentissage aléatoirement avec remise et de créer un modèle sur chacun des échantillons.

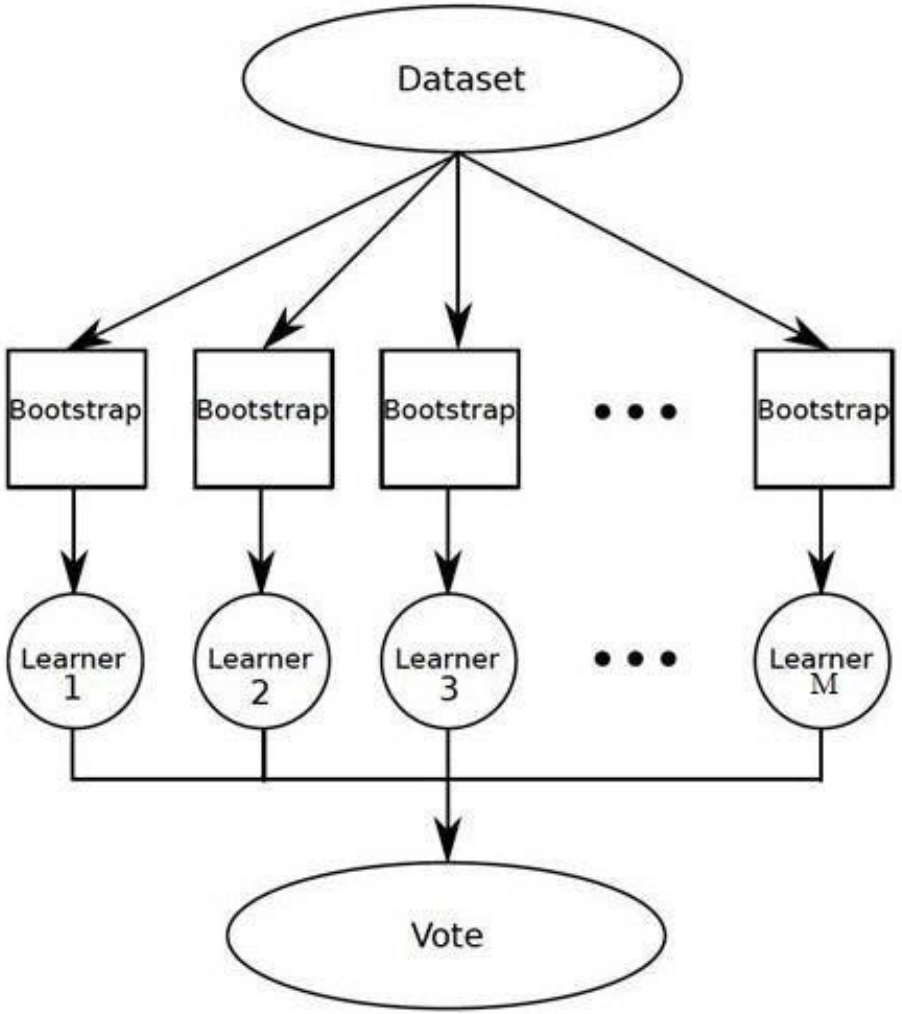
Lors d'une régression, les prédictions des modèles sont moyennées. Lors d'une classification, la prédiction finale est la modalité majoritairement prédite.

Avantages :

- Applicable à tout type de modèle
- Il répond aux problèmes de classification et régression

Inconvénients :

- Difficilement interprétable
- Attention au sur-apprentissage



BAGGING – IMPLÉMENTATION SKLEARN

- estimator :
- n_estimators :
- max_samples :
- max_features :
- bootstrap :
- bootstrap_features:
- oob_score :
- n_jobs :

```python
from sklearn.ensemble import BaggingClassifier
from sklearn.neighbors import KNeighborsClassifier

bagging = BaggingClassifier(
    base_estimator=KNeighborsClassifier(),
    n_estimators=10,
    max_samples=0.5,
    max_features=0.5,
    bootstrap=True,
    bootstrap_features=False,
    oob_score=False,
    n_jobs=-1,
)

bagging.fit(X, y)
```

BAGGING – IMPLÉMENTATION SKLEARN

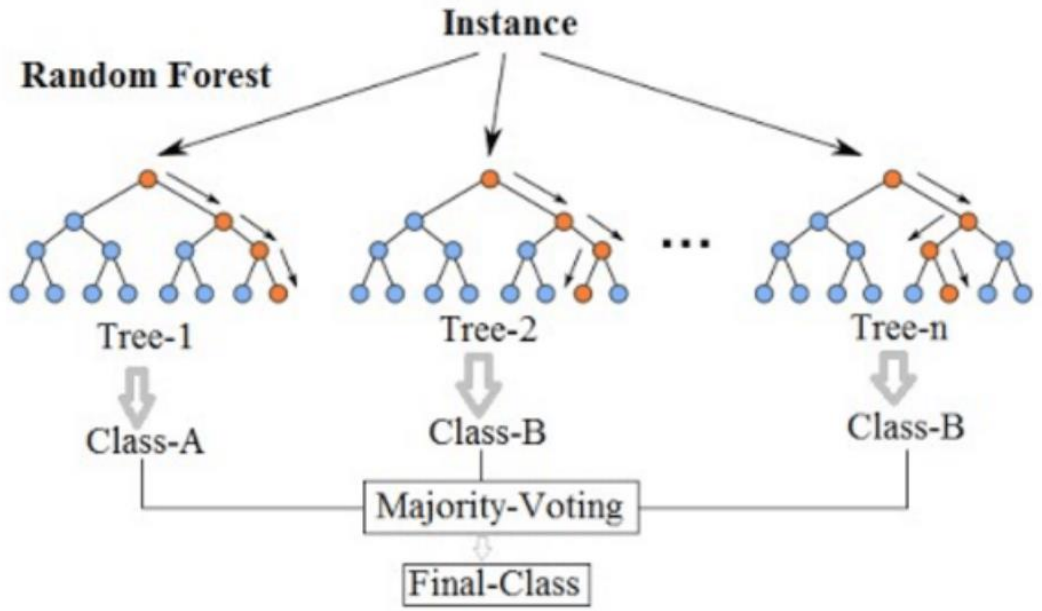
- estimator : Algorithme à agréger
- n_estimators : Nombre d'estimateur à agréger
- max_samples : Nombre de données maximum par estimateur
- max_features : Proportion de variables à utiliser
- bootstrap : Tirage d'individus avec ou sans remise
- bootstrap_features : Tirage de variables avec ou sans remise
- oob_score : Evaluation de l'erreur en out of the bag (si bootstrap = True)
- n_jobs : Pour paralléliser

```python
from sklearn.ensemble import BaggingClassifier
from sklearn.neighbors import KNeighborsClassifier

bagging = BaggingClassifier(
    base_estimator=KNeighborsClassifier(),
    n_estimators=10,
    max_samples=0.5,
    max_features=0.5,
    bootstrap=True,
    bootstrap_features=False,
    oob_score=False,
    n_jobs=-1,
)

bagging.fit(X, y)
```

RANDOM FOREST

Le Random Forest est un modèle ensembliste basé sur la construction de multiples arbres de décision (CART).

Chaque arbre est construit à partir d'un échantillon aléatoire avec remise des individus et des variables. Lors d'une régression, les prédictions des arbres sont moyennées. Lors d'une classification, la prédiction finale est la modalité majoritairement prédite.



Avantages :

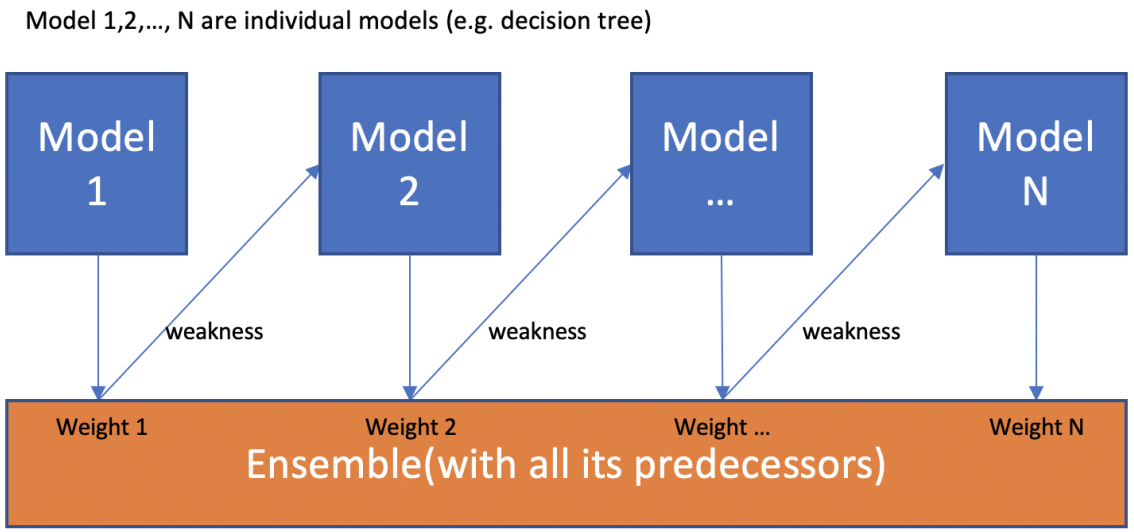
- Accepte les variables qualitatives et quantitatives
- Pas besoin de vérifier des hypothèses de normalité et de variance
- Il répond aux problèmes de classification et régression
- Modèle relativement puissant sur des problèmes de modélisation complexe

Inconvénients :

- Difficilement interprétable
- Attention au sur-apprentissage

```
from sklearn.ensemble import RandomForestClassifier

rf = RandomForestClassifier(
    n_estimators=10,
    criterion='gini'
)

rf.fit(X, y)
```

BOOSTING




```python
from sklearn.ensemble import AdaBoostClassifier
from sklearn.tree import DecisionTreeClassifier

boosting = AdaBoostClassifier(
    base_estimator=DecisionTreeClassifier(max_depth=1),
    n_estimators=10,
)

boosting.fit(X, y)
```

Le Boosting est une méthode qui consiste à sous échantillonner les données d'apprentissage aléatoirement avec remise et de créer un modèle sur chacun des échantillons. Ce qui diffère du Bagging, c'est que les modèles ne sont pas réalisés parallèlement, mais successivement les uns à la suite des autres. Les individus sont pondérés en fonction de l'erreur de prédiction du modèle précédent.

Avantages :

- Applicable à tout type de modèle
- Il répond aux problèmes de classification et régression
- Permet de mieux prédire des individus difficiles à prédire

Inconvénients :

- Difficilement interprétable
- Attention au sur-apprentissage

BOOSTING – POUR ALLER PLUS LOIN…

Gradient Boosting :

- Principe : le gradient boosting est un modèle ensembliste de boosting utilisant la descente de gradient pour optimiser une fonction de perte. Cette descente est utilisée pour le calcul des résidus des individus, lors de la construction du modèle suivant.
- Cette méthode est implémentée dans la librairie *scikit-learn* : *GradientBoostingRegressor* et *GradientBoostingClassifier*.

XGBoost :

- Principe : l'eXtreme Gradient Boosting repose également sur la descente de gradient. Cependant, la méthode utilise une approche plus régularisée dans la fonction de perte afin de contrôler le sur-apprentissage.

```
#pip install xgboost
import xgboost as xgb
xg_reg = xgb.XGBRegressor(objective ='reg:linear', colsample_bytree = 0.3, learning_rate = 0.1,
                max_depth = 5, alpha = 10, n_estimators = 10)
xg_reg.fit(X_train,y_train)
xg_reg.predict(X_test)
```

TP2 – MISE EN PLACE D'UN MODÈLE