Méthodes d'agrégation avec Python

Novembre 2024

Cédric Dangeard
cedric.dangeard@orange.com

Cours 3

- Optimisation des Hyperparamètres
- Grid & Random Search
- **TP : Optimisation**
- Les objets Sklearn
- Pipelines
- Faire ces propres estimateurs

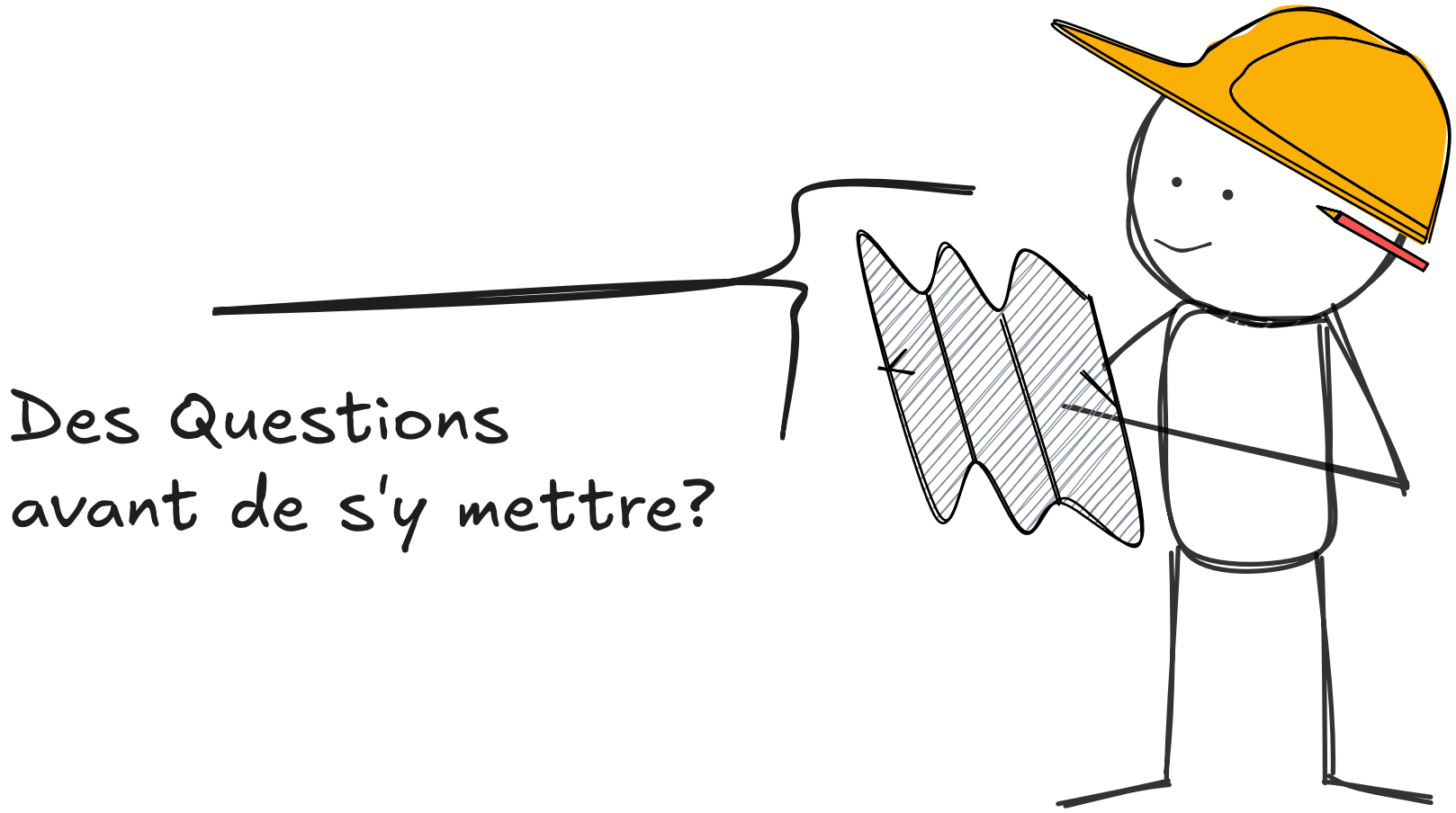
Optimisation des hyperparamètres

- GridSearchCV
  - Prend un modèle et un dictionnaire d'hyperparamètres à tester.
  - Évalue l'ensemble des combinaisons en validation croisée
  - Retourne la combinaison qui maximise la performance.
- RandomSearchCV
  - Equivalent à GridSearchCV sur la plupart des points
  - **n_iter** combinaison d'hyperparamètres :
    - Maîtrise claire du temps d'éxecution
    - Certaines combinaisons ignorées

Exemple

```
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import RandomForestClassifier
# Definition du modèle et des hyperparamètres
rf = RandomForestClassifier()
param_grid = {
    'n_estimators': [10, 100, 1000],
    'max_features': ['sqrt', 'log2'],
    'max_depth': [None, 10, 100],
}
clf = GridSearchCV(
    estimator=rf,
    param_grid=param_grid
)
# Lancement de la recherche
search = clf.fit(X,y)
# Affichage des meilleurs paramètres
search.best_params_
```

TP : Optimisation

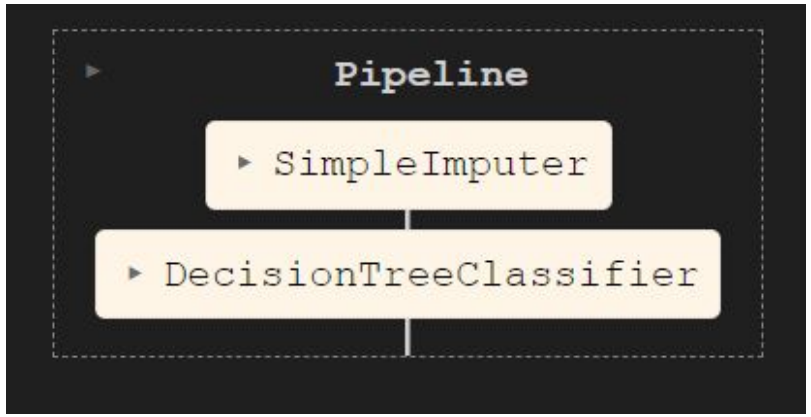


- TransformerMixin
  - Crée une méthode fit_transform() à partir de fit() et *transform()*
- BaseEstimator
  - Crée une methode get_params et set_params utilisé par GridSearchCV

Exemple pour supprimer des colonnes :

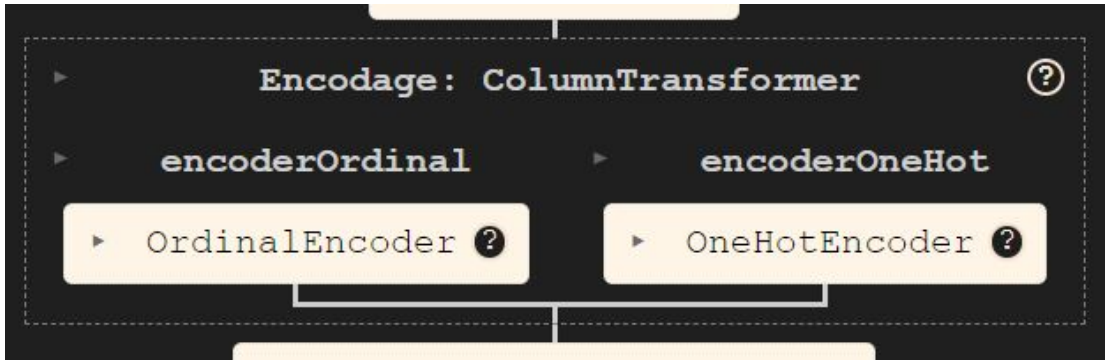
```
from sklearn.base import BaseEstimator, TransformerMixin

class DropColumns(TransformerMixin, BaseEstimator):
    def __init__(self, columnsToDrop=[]) -> None:
        self.columnsToDrop = columnsToDrop

    def fit(self, X, y=None):
        return self

    def transform(self, X, y=None):
        return X.drop(self.columnsToDrop, axis=1)

[methode for methode in dir(DropColumns) if methode[0] != '_']

>>> ['fit',
 'fit_transform',
 'get_metadata_routing',
 'get_params',
 'set_output',
 'set_params',
 'transform']
```

- Pipeline
  - Composé de plusieurs objets Sklearn
  - Permet de composer des fluxs de traitement




```
from sklearn.pipeline import Pipeline
from sklearn.impute import SimpleImputer
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import GridSearchCV

pipe = Pipeline([
    ('Imputation', SimpleImputer()),
    ('Modelisation', DecisionTreeClassifier())
])

param_grid = {
    'Imputation__strategy': ['mean',
                             'median',
                             'most_frequent'],
    'Modelisation__max_depth': [3, 10, 20, 30],
}

grid = GridSearchCV(
    estimator=pipe,
    param_grid=param_grid,
    cv = 2

)
grid.fit(X=X_train, y=Y_train)
```

- ColumnTransformer
  - Permet de diriger des columns sur des traitements différents




```
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing
  import OrdinalEncoder, OneHotEncoder

encodage = ColumnTransformer([
    ('encoderOrdinal',
        OrdinalEncoder(categories=categories),
        ordinalColumns),
    ('encoderOneHot',
        OneHotEncoder(drop='first', sparse_output=False),
        nominalColumns)
], force_int_remainder_cols=False
```

Projet : Quoi rendre

- Un fichier notebook (.ipynb) executé. (avec vos noms prénom / pseudo kaggle)
  - une partie exploration des donnée
  - une partie nettoyage des données
  - une partie modélisation
  - une partie optimisation/analyse des résultats

Critères

- Reflexion sur les traitement que vous avez fait
- Pertinance des graphiques
- Qualité de l'analyse des données
- Propreté des graphiques/cohérance générale
- Qualité de la modélisation
- Bonus pour pipeline / originalité / Kaggle