



MACHINE LEARNING WITH TREE-BASED MODELS IN PYTHON

# Tuning a CART's hyperparameters

Elie Kawerk  
Data Scientist



# Hyperparameters

Machine learning model:

- **parameters**: learned from data
  - CART example: split-point of a node, split-feature of a node, ...
- **hyperparameters**: not learned from data, set prior to training
  - CART example: max\_depth, min\_samples\_leaf, splitting criterion ...



# What is hyperparameter tuning?

- **Problem:** search for a set of optimal hyperparameters for a learning algorithm.
- **Solution:** find a set of optimal hyperparameters that results in an optimal model.
- **Optimal model:** yields an optimal **score**.
- **Score:** in sklearn defaults to accuracy (classification) and  $R^2$  (regression).
- Cross validation is used to estimate the generalization performance.



# Why tune hyperparameters?

- In sklearn, a model's default hyperparameters are not optimal for all problems.
- Hyperparameters should be tuned to obtain the best model performance.



# Approaches to hyperparameter tuning

- Grid Search
- Random Search
- Bayesian Optimization
- Genetic Algorithms
- ....



# Grid search cross validation

- Manually set a grid of discrete hyperparameter values.
- Set a metric for scoring model performance.
- Search exhaustively through the grid.
- For each set of hyperparameters, evaluate each model's CV score.
- The optimal hyperparameters are those of the model achieving the best CV score.



# Grid search cross validation: example

- Hyperparameters grids:
  - $\text{max\_depth} = \{2, 3, 4\},$
  - $\text{min\_samples\_leaf} = \{0.05, 0.1\}$
- hyperparameter space =  $\{ (2, 0.05) , (2, 0.1) , (3, 0.05), \dots \}$
- CV scores =  $\{ \text{score}_{(2, 0.05)} , \dots \}$
- optimal hyperparameters = set of hyperparameters corresponding to the best CV score.



# Inspecting the hyperparameters of a CART in sklearn

```
# Import DecisionTreeClassifier
In [1]: from sklearn.tree import DecisionTreeClassifier

# Set seed to 1 for reproducibility
In [2]: SEED = 1

# Instantiate a DecisionTreeClassifier 'dt'
In [3]: dt = DecisionTreeClassifier(random_state=SEED)
```





# Inspecting the hyperparameters of a CART in sklearn

```
# Print out 'dt's hyperparameters
```

```
In [4]: print(dt.get_params())
```

```
Out[4]:
```

```
{'class_weight': None,  
 'criterion': 'gini',  
 'max_depth': None,  
 'max_features': None,  
 'max_leaf_nodes': None,  
 'min_impurity_decrease': 0.0,  
 'min_impurity_split': None,  
 'min_samples_leaf': 1,  
 'min_samples_split': 2,  
 'min_weight_fraction_leaf': 0.0,  
 'presort': False,  
 'random_state': 1,  
 'splitter': 'best'}
```

# Grid search CV in sklearn (Breast Cancer dataset)

```
# Import GridSearchCV
In [5]: from sklearn.model_selection import GridSearchCV

# Define the grid of hyperparameters 'params_dt'
In [6]: params_dt = {
        'max_depth': [3, 4, 5, 6],
        'min_samples_leaf': [0.04, 0.06, 0.08],
        'max_features': [0.2, 0.4, 0.6, 0.8]
        }

# Instantiate a 10-fold CV grid search object 'grid_dt'
In [7]: grid_dt = GridSearchCV(estimator=dt,
                               param_grid=params_dt,
                               scoring='accuracy',
                               cv=10,
                               n_jobs=-1)

# Fit 'grid_dt' to the training data
In [8]: grid_dt.fit(X_train, y_train)
```

# Extracting the best hyperparameters

```
# Extract best hyperparameters from 'grid_dt'
In [9]: best_hyperparams = grid_dt.best_params_

In [10]: print('Best hyperparameters:\n', best_hyperparams)

Out[10]: Best hyperparameters:
         {'max_depth': 3, 'max_features': 0.4, 'min_samples_leaf': 0.06}

# Extract best CV score from 'grid_dt'
In [11]: best_cv_score = grid_dt.best_score_

In [12]: print('Best CV accuracy'.format(best_cv_score))

Out[12]: Best CV accuracy: 0.938
```

# Extracting the best estimator

```
# Extract best model from 'grid_dt'
In [13]: best_model = grid_dt.best_estimator_

# Evaluate test set accuracy
In [14]: test_acc = best_model.score(X_test, y_test)

# Print test set accuracy
In [15]: print("Test set accuracy of best model: {:.3f}".format(test_acc))

Out[15]: Test set accuracy of best model: 0.947
```



## MACHINE LEARNING WITH TREE-BASED MODELS IN PYTHON

**Let's practice!**



MACHINE LEARNING WITH TREE-BASED MODELS IN PYTHON

# Tuning an RF's Hyperparameters

Elie Kawerk  
Data Scientist



# Random Forests Hyperparameters

- CART hyperparameters
- number of estimators
- bootstrap
- ....



# Tuning is expensive

Hyperparameter tuning:

- computationally expensive,
- sometimes leads to very slight improvement,

Weight the impact of tuning on the whole project.





# Inspecting RF Hyperparameters in sklearn

```
# Import RandomForestRegressor
In [1]: from sklearn.ensemble import RandomForestRegressor

# Set seed for reproducibility
In [2]: SEED = 1

# Instantiate a random forests regressor 'rf'
In [3]: rf = RandomForestRegressor(random_state= SEED)
```



# Inspecting RF Hyperparameters in sklearn

```
# Inspect rf's hyperparameters
In [4]: rf.get_params()

Out[4]: {'bootstrap': True,
         'criterion': 'mse',
         'max_depth': None,
         'max_features': 'auto',
         'max_leaf_nodes': None,
         'min_impurity_decrease': 0.0,
         'min_impurity_split': None,
         'min_samples_leaf': 1,
         'min_samples_split': 2,
         'min_weight_fraction_leaf': 0.0,
         'n_estimators': 10,
         'n_jobs': -1,
         'oob_score': False,
         'random_state': 1,
         'verbose': 0,
         'warm_start': False}
```

# GridSearchCV in sklearn (auto dataset)

```
# Basic imports
In [5]: from sklearn.metrics import mean_squared_error as MSE
In [6]: from sklearn.model_selection import GridSearchCV

# Define a grid of hyperparameter 'params_rf'
In [7]: params_rf = {
        'n_estimators': [300, 400, 500],
        'max_depth': [4, 6, 8],
        'min_samples_leaf': [0.1, 0.2],
        'max_features': ['log2', 'sqrt']
        }

# Instantiate 'grid_rf'
In [8]: grid_rf = GridSearchCV(estimator=rf,
                               param_grid=params_rf,
                               cv=3,
                               scoring='neg_mean_squared_error',
                               verbose=1,
                               n_jobs=-1)
```

# Searching for the best hyperparameters

```
# Fit 'grid_rf' to the training set  
In [9]: grid_rf.fit(X_train, y_train)
```

Out[9]:

```
Fitting 3 folds for each of 36 candidates, totalling 108 fits  
[Parallel(n_jobs=-1)]: Done 42 tasks | elapsed: 10.0s  
[Parallel(n_jobs=-1)]: Done 108 out of 108 | elapsed: 24.3s finished  
RandomForestRegressor(bootstrap=True, criterion='mse', max_depth=4,  
                        max_features='log2', max_leaf_nodes=None,  
                        min_impurity_decrease=0.0, min_impurity_split=None,  
                        min_samples_leaf=0.1, min_samples_split=2,  
                        min_weight_fraction_leaf=0.0, n_estimators=400, n_jobs=1,  
                        oob_score=False, random_state=1, verbose=0, warm_start=False)
```

# Extracting the best hyperparameters

```
# Extract best hyperparameters from 'grid_rf'
In [10]: best_hyperparams = grid_rf.best_params_

In [11]: print('Best hyperparameters:\n', best_hyperparams)

Out[11]: Best hyperparameters:
        {'max_depth': 4,
         'max_features': 'log2',
         'min_samples_leaf': 0.1,
         'n_estimators': 400}
```



# Evaluating the best model performance

```
# Extract best model from 'grid_rf'
In [12]: best_model = grid_rf.best_estimator_

# Predict the test set labels
In [13]: y_pred = best_model.predict(X_test)

# Evaluate the test set RMSE
In [14]: rmse_test = MSE(y_test, y_pred)**(1/2)

# Print the test set RMSE
In [15]: print('Test set RMSE of rf: {:.2f}'.format(rmse_test))

Out[15]: Test set RMSE of rf: 3.89
```



## MACHINE LEARNING WITH TREE-BASED MODELS IN PYTHON

**Let's practice!**



MACHINE LEARNING WITH TREE-BASED MODELS IN PYTHON

# Congratulations!

Elie Kawerk  
Data Scientist





# How far you have come

- **Chapter 1:** Decision-Tree Learning
- **Chapter 2:** Generalization Error, Cross-Validation, Ensembling
- **Chapter 3:** Bagging and Random Forests
- **Chapter 4:** AdaBoost and Gradient-Boosting
- **Chapter 5:** Model Tuning



MACHINE LEARNING WITH TREE-BASED MODELS IN PYTHON

**Thank you!**