Model Tuning

Each machine learning model has a set of options or configurations that can be tuned to optimize the model when fitting to data. These configurations are called **hyper-parameters**. Hence, for each hyper-parameter, there exist a range of values that can be chosen. Taking into consideration the number of hyper-parameters that an algorithm has, the entire space can become exponentially large and infeasible to explore all of them. Scikit-learn provides two convenient modules for searching through the hyper-parameter space of an algorithm to find the best values for each hyper-parameter that optimizes the model.

These modules are the

- Grid search
- Randomized search

Grid Search

Grid search comprehensively explores all the specified hyper-parameter values for an estimator. It is implemented using the **GridSearchCV** module. Let's see an example using the Random forest for regression. The hyper-parameters we'll search over are

- The number of trees in the forest, **n_estimators**
- The maximum depth of the tree, **max_depth**
- The minimum number of samples required to split an internal node, **min_samples_leaf**

```
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import RandomForestRegressor
from sklearn import datasets

# load dataset
data = datasets.load_boston()

# separate features and target
X = data.data
y = data.target
```

```
# construct grid search parameters in a dictionary
parameters = {
    'n_estimators': [2, 4, 6, 8, 10, 12, 14, 16],
    'max_depth': [2, 4, 6, 8],
    'min_samples_leaf': [1,2,3,4,5]
    }

# create the model
rf_model = RandomForestRegressor()

# run the grid search
grid_search = GridSearchCV(estimator=rf_model, param_grid=parameters)

# fit the model
grid_search.fit(X,y)
'Output':
GridSearchCV(cv=None, error_score='raise',
       estimator=RandomForestRegressor(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=None, verbose=0, warm_
           start=False),
       fit_params=None, iid=True, n_jobs=1,
       param_grid={'n_estimators': [2, 4, 6, 8, 10, 12, 14, 16],
       'max_depth': [2, 4, 6, 8], 'min_samples_leaf': [1, 2, 3, 4, 5]},
       pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
       scoring=None, verbose=0)

# evaluate the model performance
print("Best Accuracy: %.3f%%" %  (grid_search.best_score_*100.0))
'Output':
Best Accuracy: 57.917%
```

```
# best set of hyper-parameter values
print("Best n_estimators: %d \nBest max_depth: %d \nBest min_samples_leaf:
%d " %  \
            (grid_search.best_estimator_.n_estimators, \
            grid_search.best_estimator_.max_depth, \
            grid_search.best_estimator_.min_samples_leaf))
'Output':
Best n_estimators: 14
Best max_depth: 8
Best min_samples_leaf: 1
```

Randomized Search

As opposed to grid search, not all the provided hyper-parameter values are evaluated, but rather a determined number of hyper-parameter values are sampled from a random uniform distribution. The number of hyper-parameter values that can be evaluated is determined by the **n_iter** attribute of the **RandomizedSearchCV** module.

In this example, we will use the same scenario as in the grid search case.

```
from sklearn.model_selection import RandomizedSearchCV
from sklearn.ensemble import RandomForestRegressor
from sklearn import datasets

# load dataset
data = datasets.load_boston()

# separate features and target
X = data.data
y = data.target

# construct grid search parameters in a dictionary
parameters = {
    'n_estimators': [2, 4, 6, 8, 10, 12, 14, 16],
    'max_depth': [2, 4, 6, 8],
    'min_samples_leaf': [1,2,3,4,5]
    }
```