Hyper-parameter tuning

Hyper-parameter optimization or **tuning** is the problem of choosing a set of optimal hyper-parameters for a learning algorithm. The same kind of machine learning model can require different constraints, weights or learning rates to generalize different data patterns. These measures are called hyper-parameters, and have to be tuned so that the model can optimally solve the machine learning problem. Hyper-parameter optimization finds a tuple of hyper-parameters that yields an optimal model which minimizes a predefined loss function on given independent data. The objective function takes a tuple of hyper-parameters and returns the associated loss. Cross-validation is often used to estimate this generalization performance.

Here let me illustrate tuning of parameters **max_depth** and **min_child_weight** of **XGBClassifier**.

max_depth (int) — Maximum tree depth for base learners.

- Used to control over-fitting as higher depth will allow model to learn relations very specific to a particular sample.
- Typical values: 3–9

min_child_weight (int) — Minimum sum of instance weight(hessian) needed in a child.

- Used to control over-fitting. Higher values prevent a model from learning relations which might be highly specific to the particular sample selected for a tree.
- Too high values can lead to under-fitting hence, it should be tuned using CV. Typical values: 1–5

```
from sklearn.preprocessing import RobustScaler
from xgboost import XGBClassifier

# Create pipeline
pipeline = make_pipeline(\
ce.BinaryEncoder(),
RobustScaler(),
XGBClassifier(learning_rate=0.1, n_estimators=1000,
max_depth=4, min_child_weight=6,
gamma=0, subsample=0.8,
colsample_bytree=0.8,
objective= 'multi:softmax', num_class=3,
scale_pos_weight=1,
seed=42, n_jobs=4))

# Model validation.
```

```python
from sklearn.model_selection import GridSearchCV

param_grid = {
'xgbclassifier__max_depth': range(3, 10, 2),
'xgbclassifier__min_child_weight': range(1, 6, 2)
}

gridsearch1 = GridSearchCV(pipeline, param_grid=param_grid, cv=2,
scoring='accuracy', verbose=20)

gridsearch1.fit(X_train, y_train)

# Interpret the results.

# Best cross validation score
print('Cross Validation Score:', gridsearch1.best_score_)

# Best parameters which resulted in the best score
print('Best Parameters:', gridsearch1.best_params_)
```

During second iteration best value for **max_depth was 5**. Now it is **6**. So in the third iteration let us reconfirm the value of max_depth. Now let us keep **min_child_weight** parameter fixed to 1.

```python
param_grid3 = {
'xgbclassifier__max_depth': [5, 6, 7],
```

```
'xgbclassifier__min_child_weight': [1]
}

gridsearch3 = GridSearchCV(pipeline, param_grid=param_grid3, cv=2,
scoring='accuracy', verbose=20)

gridsearch3.fit(X_train, y_train)

# Interpret the results of Iteration 3

# Best cross validation score
print('Cross Validation Score:', gridsearch3.best_score_)

# Best parameters which resulted in the best score
print('Best Parameters:', gridsearch3.best_params_)
```

Output from third iteration of parameter tuning:

- **Cross Validation Score:** 0.793625140291807
- **Best Parameters:** {'xgbclassifier__max_depth': 6, 'xgbclassifier__min_child_weight': 1}

From third iteration we can confirm the values of **max_depth** as **6**. In the same way, we can pickup other parameters and tune for their optimized values.

As the model performance increases, it becomes exponentially difficult to tune.