All-Hyperparamter-Optimization/Hyper Parameter Optimization.ipynb at master · krishnaik06/All-Hyperparamter-Optimization · GitHub
Live-Discussing All Hyperparameter Tuning Techniques Data Science Machine Learning - YouTube
Hyperparameter Tuning Techniques Genetic Algorithms And Optuna Data Science Machine Learning- Part 2 - YouTube

Hyperparameter Tuning

Hyperparameter tuning is crucial as they control the overall behaviour of a machine learning model. Every machine learning model will have different hyperparameters that can be set. Tuning these hyperparameters may improve the performance of the model.

Introduction

Every model has some hyperparameter that can be set to modify them. For example, a neural network model has the number of layers, number of neurons in a layer, and the activation function as hyperparameters. These hyperparameters can be found from the documentation of the model.

For random forest, the default parameters are:

{'bootstrap': True, 'ccp_alpha': 0.0, 'class_weight': None, 'criterion': 'gini', 'max_depth': None, 'max_features': 'auto', 'max_leaf_nodes': None, 'max_samples': 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': 100, 'n_jobs': None, 'oob_score': False, 'random_state': 1, 'verbose': 0, 'warm_start': False}

Randomized Search

This chooses the hyperparameter sample combinations randomly from grid space. Because of this reason, there is no guarantee that we will find the best result like Grid Search. But, this search can be extremely effective in practice as computational time is very less. Practically, this can be used before using grid search to narrow down the sample size of the best hyperparameter.

To use random search cv, define the range of parameter values. Then the search will take place in that range. Make dictionary to form random grid

```
import numpy as np
from sklearn.model_selection import RandomizedSearchCV
# Number of trees in random forest
n_estimators = [int(x) for x in np.linspace(start = 200, stop = 2000, num =
10)]
# Number of features to consider at every split
max_features = ['auto', 'sqrt','log2']
# Maximum number of levels in tree
max_depth = [int(x) for x in np.linspace(10, 1000,10)]
# Minimum number of samples required to split a node
min_samples_split = [2, 5, 10,14]
# Minimum number of samples required at each leaf node
min_samples_leaf = [1, 2, 4,6,8]
# Create the random grid
random_grid = {'n_estimators': n_estimators,
               'max_features': max_features,
               'max_depth': max_depth,
               'min_samples_split': min_samples_split,
               'min_samples_leaf': min_samples_leaf,
              'criterion':['entropy','gini']}
print(random_grid)
```

To run the random search cv,

```
rf=RandomForestClassifier()
rf_randomcv=RandomizedSearchCV(estimator=rf,param_distributions=random_grid
,n_iter=100,cv=3,verbose=2,
                               random_state=50,n_jobs=1)
### fit the randomized model
rf_randomcv.fit(X_train,y_train)
```

Here, cv and verbose refer to -

Grid Search CV

This method tries every possible combination of each set of hyper-parameters. Using this method, we can find the best set of values in the parameter search space. This usually uses more computational power and takes a long time to run since this method needs to try every combination in the grid size. Using this after random search cv is beneficial because the sample space has been reduced.

```
from sklearn.model_selection import GridSearchCV

param_grid = {
    'criterion': [rf_randomcv.best_params_['criterion']],
    'max_depth': [rf_randomcv.best_params_['max_depth']],
    'max_features': [rf_randomcv.best_params_['max_features']],
    'min_samples_leaf': [rf_randomcv.best_params_['min_samples_leaf'],
                         rf_randomcv.best_params_['min_samples_leaf']+2,
                         rf_randomcv.best_params_['min_samples_leaf'] + 4],
    'min_samples_split': [rf_randomcv.best_params_['min_samples_split'] -
2,
                          rf_randomcv.best_params_['min_samples_split'] -
1,
                          rf_randomcv.best_params_['min_samples_split'],
                          rf_randomcv.best_params_['min_samples_split'] +1,
                          rf_randomcv.best_params_['min_samples_split'] +
2],
    'n_estimators': [rf_randomcv.best_params_['n_estimators'] - 200,
rf_randomcv.best_params_['n_estimators'] - 100,
                     rf_randomcv.best_params_['n_estimators'],
                     rf_randomcv.best_params_['n_estimators'] + 100,
rf_randomcv.best_params_['n_estimators'] + 200]
}
```

Here, we make a grid by modifying the best parameter obtained by rando search cv and search with increased finetuning t get better parameters.

Bayesian Optimization

Bayesian optimization uses probability to find the minimum of a function. The final aim is to find the input value to a function which can gives us the lowest possible output value.It usually performs better than random,grid and manual search providing better performance in the testing phase and reduced optimization time. In Hyperopt, Bayesian Optimization can be implemented giving 3 three main parameters to the function fmin.

Objective Function = defines the loss function to minimize.

Domain Space = defines the range of input values to test (in Bayesian Optimization this space creates a probability distribution for each of the used Hyperparameters).

Optimization Algorithm = defines the search algorithm to use to select the best input values to use in each new iteration.

```
from hyperopt import hp,fmin,tpe,STATUS_OK,Trials
space = {'criterion': hp.choice('criterion', ['entropy', 'gini']),
        'max_depth': hp.quniform('max_depth', 10, 1200, 10),
        'max_features': hp.choice('max_features', ['auto', 'sqrt','log2',
None]),
        'min_samples_leaf': hp.uniform('min_samples_leaf', 0, 0.5),
        'min_samples_split' : hp.uniform ('min_samples_split', 0, 1),
        'n_estimators' : hp.choice('n_estimators', [10, 50, 300, 750,
1200,1300,1500])
    }
```

First, we define the space over which we will try to optimize the model. This is similar to defining the range in randomsearch and grid search.

```
def objective(space):
    model = RandomForestClassifier(criterion = space['criterion'],
max_depth = space['max_depth'],
                                 max_features = space['max_features'],
                                 min_samples_leaf =
space['min_samples_leaf'],
                                 min_samples_split =
space['min_samples_split'],
                                 n_estimators = space['n_estimators'],
                                 )

    accuracy = cross_val_score(model, X_train, y_train, cv = 5).mean()

    # We aim to maximize accuracy, therefore we return it as a negative
value
    return {'loss': -accuracy, 'status': STATUS_OK }
```

Here, we defined the objective function that is a model on which we will test the parameters. These will then be used in our trained model.

```
from sklearn.model_selection import cross_val_score
trials = Trials()
best = fmin(fn= objective,
            space= space,
            algo= tpe.suggest,
            max_evals = 80,
            trials= trials)
best
```

```
crit = {0: 'entropy', 1: 'gini'}
feat = {0: 'auto', 1: 'sqrt', 2: 'log2', 3: None}
est = {0: 10, 1: 50, 2: 300, 3: 750, 4: 1200,5:1300,6:1500}
print(crit[best['criterion']])
print(feat[best['max_features']])
print(est[best['n_estimators']])
```

Here, we get the best parameters for out model.

```
trainedforest = RandomForestClassifier(criterion = crit[best['criterion']],
max_depth = best['max_depth'],
                                       max_features =
feat[best['max_features']],
                                       min_samples_leaf =
best['min_samples_leaf'],
                                       min_samples_split =
best['min_samples_split'],
                                       n_estimators =
est[best['n_estimators']]).fit(X_train,y_train)
predictionforest = trainedforest.predict(X_test)
print(confusion_matrix(y_test,predictionforest))
print(accuracy_score(y_test,predictionforest))
print(classification_report(y_test,predictionforest))
acc5 = accuracy_score(y_test,predictionforest)
```