There are 2 packages for Bayesian Optimization. They are “bayes\_opt” and “hyperopt” (Distributed Asynchronous Hyper-parameter Optimization).

We will compare the two in terms of the time to run, accuracy, and output.

Hyper Parameter tuning Using GridSearchCV

Now, I want to perform hyperparameter-tuning on GradientBoostingClassifier. The hyperparameters to tune are “max\_depth”, “max\_features”, “learning\_rate”, “n\_estimators”, and “subsample”.

Note that as mentioned above, these hyperparameters are only for GradientBoostingClassifier, not for the other algorithms. The accuracy metric is the accuracy score. I will run 5 fold cross-validation.

Below is the code for GridSearchCV. We can see that the value options for each hyperparameter are set in the “param\_grid”

For example, the GridSearchCV will try to run with n\_estimators of 80, 100, and so on until 150. To know how many times the GridSearchCV will run, just multiply the number of value options in each hyperparameter with one another. It will be 8 x 3 x 3 x 5 x 3 = 1080. And for each of the 1080 GridSearchCV, there will be 5 fold cross-validation. That makes 1080 x 5 = 5400 models should be built to find which is the best.

# Load packages

from scipy.stats import uniform

from sklearn.model\_selection import cross\_val\_score

from sklearn.ensemble import GradientBoostingClassifier

from sklearn.model\_selection import GridSearchCV

**# GridSearchCV**

param\_grid = {'max\_depth':[3,4,5,6,7,8,9,10],

'max\_features':[0.8,0.9,1],

'learning\_rate':[0.01,0.1,1],

'n\_estimators':[80,100,120,140,150],

'subsample': [0.8,0.9,1]}

grid = GridSearchCV(estimator=GradientBoostingClassifier(), param\_grid=param\_grid, scoring=acc\_score, cv=5)

grid.fit(X\_train.iloc[1:100,], y\_train.iloc[1:100,])

**Disadvantages**

We can miss good hyperparameter values not set in the beginning. For instance, we do not set an option for the max\_features to be 0.85 or the learning\_rate to be 0.05. We do not know if that combination can give better accuracy.

To overcome this, we can try RandomizedSearchCV.

Hyper Parameter-Tuning Using RandomizedSearchCV

The code sets a range of possible values for each hyperparameter. For example, the learning\_rate can have any values from 0.01 to 1 distributed uniformly.

from sklearn.model\_selection import RandomizedSearchCV

**# RandomizedSearhCV**

param\_rand = {'max\_depth':uniform(3,10),

'max\_features':uniform(0.8,1),

'learning\_rate':uniform(0.01,1),

'n\_estimators':uniform(80,150),

'subsample':uniform(0.8,1)}

rand = RandomizedSearchCV(estimator=GradientBoostingClassifier(), param\_distributions=param\_rand, scoring=acc\_score, cv=5)

rand.fit(X\_train.iloc[1:100,], y\_train.iloc[1:100,])

Problem With Uninformed Search

The problem with uninformed search is that it takes **relatively a long time** to build all the models. Informed search can solve this problem. In informed search, the previous models with a certain set of hyperparameter values can inform the later model which hyperparameter values better to select.

One of the methods to do this is coarse-to-fine. This involves running GridSearchCV or RandomizedSearchCV more than once. Each time, the hyperparameter value range is more specific.

For example, we start RandomizedSearchCV with learning\_rate ranging from 0.01 to 1. Then, we find out that high accuracy models have their learning\_rate around 0.1 to 0.3. Hence, we can run again GridSearchCV focusing on the learning\_rate between 0.1 and 0.3. This process can continue until a satisfactory result is achieved. The first trial is coarse because the value range is large, from 0.01 to 1. The later trial is fine as the value range is focused on 0.1 to 0.3.

The drawback of the coarse-to-fine method is that we need to run the code repeatedly and observe the value range of hyperparameters-tuning. You might be thinking if there is a way to automate this. Yes, that is why my favorite is Bayesian Optimization.

Baysian Optimization

Bayesian Optimization also runs models many times with different sets of hyperparameter values, but it evaluates the past model information to select hyperparameter values to build the newer model. This is said to spend less time to reach the highest accuracy model than the previously discussed methods.

bayes\_opt

As mentioned in the beginning, there are two packages in python that I usually use for Bayesian Optimization. The first one is bayes\_opt. Here is the code to run it.

from bayes\_opt import BayesianOptimization

# Gradient Boosting Machine

def gbm\_cl\_bo(max\_depth, max\_features, learning\_rate, n\_estimators, subsample):

params\_gbm = {}

params\_gbm['max\_depth'] = round(max\_depth)

params\_gbm['max\_features'] = max\_features

params\_gbm['learning\_rate'] = learning\_rate

params\_gbm['n\_estimators'] = round(n\_estimators)

params\_gbm['subsample'] = subsample

scores = cross\_val\_score(GradientBoostingClassifier(random\_state=123, \*\*params\_gbm),

X\_train, y\_train, scoring=acc\_score, cv=5).mean()

score = scores.mean()

return score

# Run Bayesian Optimization

start = time.time()

params\_gbm ={

'max\_depth':(3, 10),

'max\_features':(0.8, 1),

'learning\_rate':(0.01, 1),

'n\_estimators':(80, 150),

'subsample': (0.8, 1)

}

gbm\_bo = BayesianOptimization(gbm\_cl\_bo, params\_gbm, random\_state=111)

gbm\_bo.maximize(init\_points=20, n\_iter=4)

print('It takes %s minutes' % ((time.time() - start)/60))

output:

| iter | target | learni... | max\_depth | max\_fe... | n\_esti... | subsample |

-------------------------------------------------------------------------------------

| 1 | 0.7647 | 0.616 | 4.183 | 0.8872 | 133.8 | 0.8591 |

| 2 | 0.7711 | 0.1577 | 3.157 | 0.884 | 96.71 | 0.8675 |

| 3 | 0.7502 | 0.9908 | 4.664 | 0.8162 | 126.9 | 0.9242 |

| 4 | 0.7681 | 0.2815 | 6.264 | 0.8237 | 85.18 | 0.9802 |

| 5 | 0.7107 | 0.796 | 8.884 | 0.963 | 149.4 | 0.9155 |

| 6 | 0.7442 | 0.8156 | 5.949 | 0.8055 | 111.8 | 0.8211 |

| 7 | 0.7286 | 0.819 | 7.884 | 0.9131 | 99.2 | 0.9997 |

| 8 | 0.7687 | 0.1467 | 7.308 | 0.897 | 108.4 | 0.9456 |

| 9 | 0.7628 | 0.3296 | 5.804 | 0.8638 | 146.3 | 0.9837 |

| 10 | 0.7668 | 0.8157 | 3.239 | 0.9887 | 146.5 | 0.9613 |

| 11 | 0.7199 | 0.4865 | 9.767 | 0.8834 | 102.3 | 0.8033 |

| 12 | 0.7708 | 0.0478 | 3.372 | 0.8256 | 82.34 | 0.8453 |

| 13 | 0.7679 | 0.5485 | 4.25 | 0.8359 | 90.47 | 0.9366 |

| 14 | 0.7409 | 0.4743 | 8.378 | 0.9338 | 110.9 | 0.919 |

| 15 | 0.7216 | 0.467 | 9.743 | 0.8296 | 143.5 | 0.8996 |

| 16 | 0.7306 | 0.5966 | 7.793 | 0.8355 | 140.5 | 0.8964 |

**| 17 | 0.772 | 0.07865 | 5.553 | 0.8723 | 113.0 | 0.8359 |**

| 18 | 0.7589 | 0.1835 | 9.644 | 0.9311 | 89.45 | 0.9856 |

| 19 | 0.7662 | 0.8434 | 3.369 | 0.8407 | 141.1 | 0.9348 |

| 20 | 0.7566 | 0.3043 | 8.141 | 0.9237 | 94.73 | 0.9604 |

| 21 | 0.7683 | 0.02841 | 9.546 | 0.9055 | 140.5 | 0.8805 |

| 22 | 0.7717 | 0.05919 | 4.285 | 0.8093 | 92.7 | 0.9528 |

| 23 | 0.7676 | 0.1946 | 7.351 | 0.9804 | 108.3 | 0.929 |

| 24 | 0.7602 | 0.7131 | 5.307 | 0.8428 | 91.74 | 0.9193 |

=================================================================================

It takes 20.90080655813217 minutes

params\_gbm = gbm\_bo.max['params']

params\_gbm['max\_depth'] = round(params\_gbm['max\_depth'])

params\_gbm['n\_estimators'] = round(params\_gbm['n\_estimators'])

params\_gbm

Output:

{'learning\_rate': 0.07864837617488214,

'max\_depth': 6,

'max\_features': 0.8723008386644597,

'n\_estimators': 113,

'subsample': 0.8358969695415375}

The package bayes\_opt takes 20 minutes to build 24 models. The best accuracy is 0.772.

hyperopt

Another package is hyperopt. Here is the code.

from hyperopt import hp, fmin, tpe

# Run Bayesian Optimization from hyperopt

start = time.time()

space\_lr = {'max\_depth': hp.randint('max\_depth', 3, 10),

'max\_features': hp.uniform('max\_features', 0.8, 1),

'learning\_rate': hp.uniform('learning\_rate',0.01, 1),

'n\_estimators': hp.randint('n\_estimators', 80,150),

'subsample': hp.uniform('subsample',0.8, 1)}

def gbm\_cl\_bo2(params):

params = {'max\_depth': params['max\_depth'],

'max\_features': params['max\_features'],

'learning\_rate': params['learning\_rate'],

'n\_estimators': params['n\_estimators'],

'subsample': params['subsample']}

gbm\_bo2 = GradientBoostingClassifier(random\_state=111, \*\*params)

best\_score = cross\_val\_score(gbm\_bo2, X\_train, y\_train, scoring=acc\_score, cv=5).mean()

return 1 - best\_score

gbm\_best\_param = fmin(fn=gbm\_cl\_bo2,

space=space\_lr,

max\_evals=24,

rstate=np.random.RandomState(42),

algo=tpe.suggest)

print('It takes %s minutes' % ((time.time() - start)/60))

Output:

100%|██████████| 24/24 [19:53<00:00, 49.74s/trial, best loss: 0.22769091027055077]

It takes 19.897333371639252 minutes

gbm\_best\_param

Output:

{'learning\_rate': 0.03516615427790515,

'max\_depth': 6,

'max\_features': 0.8920776081423815,

'n\_estimators': 148,

'subsample': 0.9981549036976672}

The package hyperopt takes 19.9 minutes to run 24 models. The best loss is 0.228. It means that the best accuracy is 1 – 0.228 = 0.772. The duration to run bayes\_opt and hyperopt is almost the same.

The accuracy is also almost the same although the results of the best hyperparameters are different. But, there is another difference. The package bayes\_opt shows the process of tuning the values of the hyperparameters. We can see which values are used for each iteration. The package hyperopt only shows one line of the progress bar, best loss, and duration.

In my opinion, I prefer bayes\_opt because, in reality, we may feel the tuning process takes too long time and just want to terminate the process. After stopping the processing, we just want to take the best hyperparameter-tuning result. We can do that with bayes\_opt, but not with hyperopt.

There are still other ways of automatic hyperparameter-tuning. Not only the hyperparameter-tuning, but choosing the Machine Learning algorithms also can be automated. I will discuss that next time. The above code is available [here](https://www.kaggle.com/rendyk/bayesian-optimization-bayes-opt-or-fmin" \t "_blank).

Connect with me [here](https://www.linkedin.com/in/rendy-kurnia/" \t "_blank).

***The media shown in this article on Top Machine Learning Libraries in Julia are not owned by Analytics Vidhya and is used at the Author’s discretion.***

**Related**

[A picture containing text

Description automatically generated](https://www.analyticsvidhya.com/blog/2020/11/hyperparameter-tuning-using-optuna/?relatedposts_hit=1&relatedposts_origin=78851&relatedposts_position=0&relatedposts_hit=1&relatedposts_origin=78851&relatedposts_position=0&relatedposts_hit=1&relatedposts_origin=78851&relatedposts_position=0)

[Hyperparameter Tuning using Optuna](https://www.analyticsvidhya.com/blog/2020/11/hyperparameter-tuning-using-optuna/?relatedposts_hit=1&relatedposts_origin=78851&relatedposts_position=0&relatedposts_hit=1&relatedposts_origin=78851&relatedposts_position=0&relatedposts_hit=1&relatedposts_origin=78851&relatedposts_position=0)

[A picture containing text

Description automatically generated](https://www.analyticsvidhya.com/blog/2020/09/alternative-hyperparameter-optimization-technique-you-need-to-know-hyperopt/?relatedposts_hit=1&relatedposts_origin=78851&relatedposts_position=1&relatedposts_hit=1&relatedposts_origin=78851&relatedposts_position=1&relatedposts_hit=1&relatedposts_origin=78851&relatedposts_position=1)

[Alternative Hyperparameter Optimization Technique You need to Know – Hyperopt](https://www.analyticsvidhya.com/blog/2020/09/alternative-hyperparameter-optimization-technique-you-need-to-know-hyperopt/?relatedposts_hit=1&relatedposts_origin=78851&relatedposts_position=1&relatedposts_hit=1&relatedposts_origin=78851&relatedposts_position=1&relatedposts_hit=1&relatedposts_origin=78851&relatedposts_position=1)

[A picture containing text, dome

Description automatically generated](https://www.analyticsvidhya.com/blog/2021/11/tune-ml-models-in-no-time-with-optuna/?relatedposts_hit=1&relatedposts_origin=78851&relatedposts_position=2&relatedposts_hit=1&relatedposts_origin=78851&relatedposts_position=2&relatedposts_hit=1&relatedposts_origin=78851&relatedposts_position=2)

[Tune ML Models in No Time with Optuna](https://www.analyticsvidhya.com/blog/2021/11/tune-ml-models-in-no-time-with-optuna/?relatedposts_hit=1&relatedposts_origin=78851&relatedposts_position=2&relatedposts_hit=1&relatedposts_origin=78851&relatedposts_position=2&relatedposts_hit=1&relatedposts_origin=78851&relatedposts_position=2)