Hyperparameter Tuning

Hyperparameter tuning is the adjustment of various pre-execution parameters passed to our Machine Learning models that affect their training/execution. Here we use two automated methods of choosing from a wide set of specified parameters - Grid Search (which exhaustively tries all combinations of specified parameters) and Random Search (which tries randomly sampled combinations of parameters).

Grid Search

Grid Search (exhaustive) hyperparameter tuning.

In [69]:
```
%%script false --no-raise-error
# ^ disables this cell in jupyter notebook

from sklearn.model_selection import GridSearchCV

for model in models:
    model_name = model.__class__.__name__

    #----Logistic Regression Hyperparameter Tuning----

    if model_name == 'LogisticRegression':

        penalty = ['l1', 'l2']
        C = [0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000]
        class_weight = [{1:0.5, 0:0.5}, {1:0.4, 0:0.6}, {1:0.6, 0:0.4}, {1:0.7, 0:0.3}]
        solver = ['liblinear', 'saga']

        param_grid = dict(penalty=penalty,
                          C=C,
                          class_weight=class_weight,
                          solver=solver)

        grid = GridSearchCV(estimator=model,
                            param_grid=param_grid,
                            scoring='roc_auc',
                            verbose=1,
                            n_jobs=-1)
        grid_result = grid.fit(X_train, Y_train)
        print('Model Name: ', model_name)
        print('Best Score: ', grid_result.best_score_)
        print('Best Params: ', grid_result.best_params_)

        #----Gradient Boosting Hyperparameter Tuning----

    if model_name == 'GradientBoostingClassifier':

        learning_rate = [0.15,0.1,0.05,0.01,0.005,0.001]
        n_estimators = [100,250,500,750,1000,1250,1500,1750]
        max_depth = [2,3,4,5,6,7]

        param_grid = dict(learning_rate=learning_rate,
                          n_estimators=n_estimators,
                          max_depth=max_depth,)

        grid = GridSearchCV(estimator=model,
                            param_grid=param_grid,
                            scoring='roc_auc',
                            verbose=1,
                            n_jobs=-1)
        grid_result = grid.fit(X_train, Y_train)
        print('Model Name: ', model_name)
        print('Best Score: ', grid_result.best_score_)
        print('Best Params: ', grid_result.best_params_)
```

```
Fitting 5 folds for each of 128 candidates, totalling 640 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 4 concurrent workers.
[Parallel(n_jobs=-1)]: Done  42 tasks      | elapsed:   47.4s
[Parallel(n_jobs=-1)]: Done 192 tasks      | elapsed:  4.5min
[Parallel(n_jobs=-1)]: Done 442 tasks      | elapsed: 17.0min
[Parallel(n_jobs=-1)]: Done 640 out of 640 | elapsed: 27.6min finished
C:\ProgramData\Anaconda3\envs\project10\lib\site-packages\sklearn\utils\validation.py:760: DataConver
sionWarning: A column-vector y was passed when a 1d array was expected. Please change the shape of y
to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)

Model Name:  LogisticRegression
Best Score:  0.883580552276247
Best Params:  {'C': 1, 'class_weight': {1: 0.4, 0: 0.6}, 'penalty': 'l1', 'solver': 'liblinear'}
```

Random Search

Random Search hyperparameter tuning.

In [1]:
```
%%script false --no-raise-error
# ^ disables this cell in jupyter notebook

#exploration of specifying a range of values using numpy's logspace function.
#Used to specify range of C values in logistic regression.

import numpy as np

np.set_printoptions(precision=10)
np.set_printoptions(suppress=True)

np.logspace(0, 4, num=10)
```

Out[1]:
```
array([    1.          ,     2.7825594022,     7.7426368268,
          21.5443469003,    59.9484250319,   166.81005372  ,
         464.1588833613,  1291.5496650149,  3593.8136638046,
       10000.          ])
```

In [133]:

```
%%script false --no-raise-error
# ^ disables this cell in jupyter notebook

from sklearn.model_selection import RandomizedSearchCV

for model in models:
    model_name = model.__class__.__name__

    print(model)
    if model_name == 'LogisticRegression':

        penalty = ['l1', 'l2']
        C = np.logspace(0, 4, num=10)
        class_weight = [{1:0.5, 0:0.5}, {1:0.4, 0:0.6}, {1:0.6, 0:0.4}, {1:0.7, 0:0.3}]
        solver = ['liblinear', 'saga']

        param_distributions = dict(penalty=penalty,
                          C=C,
                          class_weight=class_weight,
                          solver=solver)

        random = RandomizedSearchCV(estimator=model,
                                    param_distributions=param_distributions,
                                    scoring='roc_auc',
                                    verbose=1, n_jobs=-1,
                                    n_iter=100)
        random_result = random.fit(X_train, Y_train)

        print('Model Name: ', model_name)
        print('Best Score: ', random_result.best_score_)
        print('Best Params: ', random_result.best_params_)

    if model_name == 'GradientBoostingClassifier':

        loss=["deviance"]
        learning_rate=np.linspace(0.05, 0.2, num=4)
        max_depth=[3,5,8]
        max_features=["log2","sqrt"]
        criterion=["friedman_mse",  "mae"]
        subsample=[0.5, 0.618, 0.8, 0.85, 0.9, 0.95, 1.0]
        n_estimators=[10]

        param_distributions = dict(loss=loss,
                          learning_rate=learning_rate,
                        #  min_samples_split=min_samples_split,
                        #  min_samples_leaf=min_samples_leaf,
                          max_depth=max_depth,
                          max_features=max_features,
                          criterion=criterion,
                          subsample=subsample,
                          n_estimators=n_estimators)

        random = RandomizedSearchCV(estimator=model,
                                    param_distributions=param_distributions,
                                    cv=3,
                                    scoring='roc_auc',
                                    verbose=1,
                                    n_jobs=-1,
                                    n_iter=100)
        random_result = random.fit(X_train, Y_train)

        print('Model Name: ', model_name)
        print('Best Score: ', random_result.best_score_)
        print('Best Params: ', random_result.best_params_)
```

```
GradientBoostingClassifier(criterion='friedman_mse', init=None,
              learning_rate=0.1, loss='deviance', max_depth=3,
              max_features=None, max_leaf_nodes=None,
              min_impurity_decrease=0.0, min_impurity_split=None,
              min_samples_leaf=14, min_samples_split=140,
              min_weight_fraction_leaf=0.0, n_estimators=100,
              n_iter_no_change=None, presort='auto', random_state=0,
              subsample=1.0, tol=0.0001, validation_fraction=0.1,
              verbose=0, warm_start=False)
Fitting 3 folds for each of 100 candidates, totalling 300 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 4 concurrent workers.
[Parallel(n_jobs=-1)]: Done  42 tasks      | elapsed:  3.3min
[Parallel(n_jobs=-1)]: Done 192 tasks      | elapsed: 16.9min
[Parallel(n_jobs=-1)]: Done 300 out of 300 | elapsed: 29.0min finished

Model Name:  GradientBoostingClassifier
Best Score:  0.864653979518611
Best Params:  {'subsample': 0.9, 'n_estimators': 10, 'max_features': 'sqrt', 'max_depth': 8, 'loss':
'deviance', 'learning_rate': 0.15000000000000002, 'criterion': 'friedman_mse'}
```