IPA □□ □□□□□□ □□(fundamental) □□

- GitHub link: here
- E-Mail: windkyle7@gmail.com

In [1]:
```
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split

iris = load_iris()
X_train, X_test, y_train, y_test =\
    train_test_split(iris.data, iris.target)
```

`GridSearchCV` □ □□ `LogisticRegression` □□□ □□□ Hyper parameter□ □□□.

In [2]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import GridSearchCV

lr = LogisticRegression(solver='lbfgs', multi_class='auto', max_iter=1000)
param_range = [0.001, 0.01, 0.1, 1.0, 10.0, 100.0, 1000.0]
param_grid = {'C': param_range}
params = {
    'estimator': lr,
    'param_grid': param_grid,
    'cv': 10,
    'iid': True,
    'return_train_score': True,
}
```

In [3]:
```
grid = GridSearchCV(**params)
grid.fit(X_train, y_train)
```

Out[3]:
```
GridSearchCV(cv=10, error_score='raise-deprecating',
             estimator=LogisticRegression(C=1.0, class_weight=None, dual=False,
                                          fit_intercept=True,
                                          intercept_scaling=1, l1_ratio=None,
                                          max_iter=1000, multi_class='auto',
                                          n_jobs=None, penalty='l2',
                                          random_state=None, solver='lbfgs',
                                          tol=0.0001, verbose=0,
                                          warm_start=False),
             iid=True, n_jobs=None,
             param_grid={'C': [0.001, 0.01, 0.1, 1.0, 10.0, 100.0, 1000.0]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=True,
             scoring=None, verbose=0)
```

□□ □□□ □□ □□□ Hyper parameter□ □□□ □□□ □□ □ □□.

In [4]:
```
import pandas as pd
pd.DataFrame(grid.cv_results_)
```

Out[4]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_C	params	split0_test_score	split1_test_score	split2_test_score	split3_test_score	...	s
0	0.003153	0.000772	0.000174	0.000052	0.001	{'C': 0.001}	0.750000	0.916667	0.750000	0.75	...	
1	0.005154	0.000359	0.000193	0.000072	0.01	{'C': 0.01}	0.916667	1.000000	0.833333	1.00	...	
2	0.009063	0.000708	0.000146	0.000018	0.1	{'C': 0.1}	1.000000	1.000000	0.916667	1.00	...	
3	0.013962	0.001782	0.000140	0.000003	1	{'C': 1.0}	1.000000	1.000000	1.000000	1.00	...	
4	0.017733	0.003538	0.000135	0.000007	10	{'C': 10.0}	1.000000	1.000000	1.000000	1.00	...	
5	0.024809	0.002786	0.000141	0.000016	100	{'C': 100.0}	1.000000	1.000000	1.000000	1.00	...	
6	0.037531	0.008949	0.000144	0.000026	1000	{'C': 1000.0}	1.000000	1.000000	1.000000	1.00	...	

7 rows × 31 columns

searchgrid

- Helps building parameter grids for scikit-learn grid search.
- https://searchgrid.readthedocs.io/en/latest/

In [5]:
```
from sklearn.base import BaseEstimator


class Dummy(BaseEstimator):
    def fit(self):
        pass

    def score(self):
        pass
```

In [6]:
```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.pipeline import Pipeline
```

```
knn = KNeighborsClassifier()
pipe = Pipeline([('clf', Dummy())])
param_grid = [
    {
        'clf': [lr]
    },
    {
        'clf': [knn],
        'clf__n_neighbors': [3, 4, 5, 6, 7]
    },
]
params['estimator'] = pipe
params['param_grid'] = param_grid
```

In [7]:
```
grid = GridSearchCV(**params)
grid.fit(X_train, y_train)
```

Out[7]:
```
GridSearchCV(cv=10, error_score='raise-deprecating',
             estimator=Pipeline(memory=None, steps=[('clf', Dummy())],
                                verbose=False),
             iid=True, n_jobs=None,
             param_grid=[{'clf': [LogisticRegression(C=1.0, class_weight=None,
                                                     dual=False,
                                                     fit_intercept=True,
                                                     intercept_scaling=1,
                                                     l1_ratio=None,
                                                     max_iter=1000,
                                                     multi_class='auto',
                                                     n_jobs=None, penalty='l2',
                                                     random_state=None,
                                                     solver='lbfgs', tol=0.0001,
                                                     verbose=0,
                                                     warm_start=False)]},
                          {'clf': [KNeighborsClassifier(algorithm='auto',
                                                        leaf_size=30,
                                                        metric='minkowski',
                                                        metric_params=None,
                                                        n_jobs=None,
                                                        n_neighbors=3, p=2,
                                                        weights='uniform')],
                           'clf__n_neighbors': [3, 4, 5, 6, 7]}],
             pre_dispatch='2*n_jobs', refit=True, return_train_score=True,
             scoring=None, verbose=0)
```

In [8]:
```
pd.DataFrame(grid.cv_results_)
```

Out[8]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_clf	param_clf__n_neighbors	param
0	0.013966	0.002585	0.000147	0.000025	LogisticRegression(C=1.0, class_weight=None, d...	NaN	{'clf': LogisticRegression(C=1. class_weight
1	0.000286	0.000050	0.000573	0.000044	KNeighborsClassifier(algorithm='auto', leaf_si...	3	{'cl KNeighborsClassifier(algorithm='auto',
2	0.000270	0.000003	0.000558	0.000011	KNeighborsClassifier(algorithm='auto', leaf_si...	4	{'cl KNeighborsClassifier(algorithm='auto',
3	0.000274	0.000007	0.000562	0.000013	KNeighborsClassifier(algorithm='auto', leaf_si...	5	{'cl KNeighborsClassifier(algorithm='auto',
4	0.000274	0.000012	0.000580	0.000026	KNeighborsClassifier(algorithm='auto', leaf_si...	6	{'cl KNeighborsClassifier(algorithm='auto',
5	0.000272	0.000004	0.000571	0.000013	KNeighborsClassifier(algorithm='auto', leaf_si...	7	{'cl KNeighborsClassifier(algorithm='auto',

6 rows × 32 columns