Grid Search

```
from sklearn.model_selection import GridSearchCV
param_grid = [
{'n_estimators': [3, 10, 30], 'max_features': [2, 4, 6, 8]},
{'bootstrap': [False], 'n_estimators': [3, 10], 'max_features': [2, 3, 4]},
]
forest_reg = RandomForestRegressor()
grid_search = GridSearchCV(forest_reg, param_grid, cv=5, n_job=6,
scoring='neg_mean_squared_error', return_train_score=True)
# scoring = 'neg_log_loss', 'roc_auc', 'f1' # n_job=-1 all cpus
grid_search.fit(X, Y)


grid_search.best_params_

grid_search.best_estimator_

cvres = grid_search.cv_results_
for mean_score, params in zip(cvres["mean_test_score"], cvres["params"]):
    print(np.sqrt(-mean_score), params)
```

Random Search

```
>>> from sklearn.model_selection import RandomizedSearchCV
>>> from scipy.stats import uniform
>>> iris = load_iris()
>>> logistic = LogisticRegression(solver='saga', tol=1e-2, max_iter=200,
...                               random_state=0)
>>> distributions = dict(C=uniform(loc=0, scale=4),
...                      penalty=['l2', 'l1'])
>>> clf = RandomizedSearchCV(logistic, distributions, random_state=0)
```

parameters

```
parameters_to_tune = {'min_samples_split': [2, 4, 6, 10, 15, 25],
                      'min_samples_leaf': [1, 2, 4, 10],
                      'max_depth': [None, 4, 10, 15],
                      'splitter' : ('best', 'random'),
                      'max_features':[None, 2, 4, 6, 8, 10, 12, 14],
                      'class_weight':['balanced', {1:2},{1:3},{1:4},{1:5},{1:10},{1:20},{1:50}]}
```

example

```
# Creating a dict of the models
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import LinearSVC
from sklearn.model_selection import cross_val_score
import seaborn as sns
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import LinearSVC
from sklearn.model_selection import cross_val_score
from xgboost import XGBClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.multiclass import OneVsRestClassifier

from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, roc_curve,



model_dict = {'Random Forest': RandomForestClassifier(n_estimators=200, random_state=0),
              'XGBClassifier': XGBClassifier(),
              'LogisticRegression': LogisticRegression(random_state=0),
              'K Nearest Neighbor': KNeighborsClassifier(),
              'linearSVM': LinearSVC()}

# Train test split
X_train, X_test, y_train, y_test = train_test_split(X,
                                                    y,
                                                    test_size=.2,
                                                    random_state=3)

# Function to get the scores for each model in a df
def model_score_df(model_dict):
    model_name, ac_score_list, p_score_list, r_score_list, f1_score_list = [], [], [], [], []
    for k, v in model_dict.items():
        model_name.append(k)
        v.fit(X_train, y_train)
        y_pred = v.predict(X_test)
        ac_score_list.append(accuracy_score(y_test, y_pred))
        p_score_list.append(precision_score(y_test, y_pred, average='macro'))
        r_score_list.append(recall_score(y_test, y_pred, average='macro'))
        f1_score_list.append(f1_score(y_test, y_pred, average='macro'))
        model_comparison_df = pd.DataFrame(
            [model_name, ac_score_list, p_score_list, r_score_list, f1_score_list]).T
        model_comparison_df.columns = [
            'model_name', 'accuracy_score', 'precision_score', 'recall_score', 'f1_score']
        model_comparison_df = model_comparison_df.sort_values(
            by='f1_score', ascending=False)
    return model_comparison_df
```