Applying Hyperparameter Optimization for ML Models

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split,cross_val_score
from sklearn.ensemble import RandomForestClassifier,RandomForestRegressor
from sklearn.metrics import classification_report,confusion_matrix,accuracy_score
from sklearn.neighbors import KNeighborsClassifier,KNeighborsRegressor
from sklearn.svm import SVC,SVR
from sklearn import datasets
import scipy.stats as stats


d = datasets.load_digits()
X = d.data
y = d.target


datasets.load_digits()
```

```
        pixel_7_2',
       'pixel_7_3',
       'pixel_7_4',
       'pixel_7_5',
       'pixel_7_6',
       'pixel_7_7'],
      'frame': None,
      'images': array([[[ 0.,  0.,  5., ...,  1.,  0.,  0.],
              [ 0.,  0., 13., ..., 15.,  5.,  0.],
              [ 0.,  3., 15., ..., 11.,  8.,  0.],
              ...,
              [ 0.,  4., 11., ..., 12.,  7.,  0.],
              [ 0.,  2., 14., ..., 12.,  0.,  0.],
              [ 0.,  0.,  6., ...,  0.,  0.,  0.]],

             [[ 0.,  0.,  0., ...,  5.,  0.,  0.],
              [ 0.,  0.,  0., ...,  9.,  0.,  0.],
              [ 0.,  0.,  3., ...,  6.,  0.,  0.],
              ...,
              [ 0.,  0.,  1., ...,  6.,  0.,  0.],
              [ 0.,  0.,  1., ...,  6.,  0.,  0.],
              [ 0.,  0.,  0., ..., 10.,  0.,  0.]],

             [[ 0.,  0.,  0., ..., 12.,  0.,  0.],
              [ 0.,  0.,  3., ..., 14.,  0.,  0.],
              [ 0.,  0.,  8., ..., 16.,  0.,  0.],

              ...,
              [ 0.,  9., 16., ...,  0.,  0.,  0.],
              [ 0.,  3., 13., ..., 11.,  5.,  0.],
              [ 0.,  0.,  0., ..., 16.,  9.,  0.]],

             ...,
```

```
        [[ 0.,  0.,  1., ...,  1.,  0.,  0.],
         [ 0.,  0., 13., ...,  2.,  1.,  0.],
         [ 0.,  0., 16., ..., 16.,  5.,  0.],
         ...,
         [ 0.,  0., 16., ..., 15.,  0.,  0.],
         [ 0.,  0., 15., ..., 16.,  0.,  0.],
         [ 0.,  0.,  2., ...,  6.,  0.,  0.]],

        [[ 0.,  0.,  2., ...,  0.,  0.,  0.],
         [ 0.,  0., 14., ..., 15.,  1.,  0.],
         [ 0.,  4., 16., ..., 16.,  7.,  0.],
         ...,
         [ 0.,  0.,  0., ..., 16.,  2.,  0.],
         [ 0.,  0.,  4., ..., 16.,  2.,  0.],
         [ 0.,  0.,  5., ..., 12.,  0.,  0.]],

        [[ 0.,  0., 10., ...,  1.,  0.,  0.],
         [ 0.,  2., 16., ...,  1.,  0.,  0.],
         [ 0.,  0., 15., ..., 15.,  0.,  0.],
         ...,
         [ 0.,  4., 16., ..., 16.,  6.,  0.],
         [ 0.,  8., 16., ..., 16.,  8.,  0.],
         [ 0.,  1.,  8., ..., 12.,  1.,  0.]]]),
 'target': array([0, 1, 2, ..., 8, 9, 8]),
```

Classifiers with Default Hyperparameters

```
#Random Forest
clf = RandomForestClassifier()
clf.fit(X,y)
scores = cross_val_score(clf, X, y, cv=3,scoring='accuracy')
print("Accuracy:"+ str(scores.mean()))
```

```
Accuracy:0.9398998330550917
```

```
#SVM
clf = SVC(gamma='scale')
clf.fit(X,y)
scores = cross_val_score(clf, X, y, cv=3,scoring='accuracy')
print("Accuracy:"+ str(scores.mean()))
```

```
Accuracy:0.9699499165275459
```

```
#KNN
clf = KNeighborsClassifier()
clf.fit(X,y)
scores = cross_val_score(clf, X, y, cv=3,scoring='accuracy')
print("Accuracy:"+ str(scores.mean()))
```

```
Accuracy:0.9627156371730662
```

Hyperparameter Optimization Using PSO

Partical swarm optimization (PSO): Each particle in a swarm communicates with other particles to detect and update the current global optimum in each iteration until the final optimum is detected

```
!pip install optunity
```

```
Requirement already satisfied: optunity in /usr/local/lib/python3.7/dist-packages (1.1.1)
```

```
#Random Forest
import optunity
import optunity.metrics

data=X
labels=y.tolist()
# Define the hyperparameter configuration space
search = {
    'n_estimators': [10, 100],
    'max_features': [1, 64],
    'max_depth': [5,50],
    "min_samples_split":[2,11],
    "min_samples_leaf":[1,11],
    "criterion":[0,1]
        }
# Define the objective function
@optunity.cross_validated(x=data, y=labels, num_folds=3)
def performance(x_train, y_train, x_test, y_test,n_estimators=None, max_features=None,max_depth=None,mi
    # fit the model
    if criterion<0.5:
      cri='gini'
    else:
        cri='entropy'
    model = RandomForestClassifier(n_estimators=int(n_estimators),
                                   max_features=int(max_features),
                                   max_depth=int(max_depth),
                                   min_samples_split=int(min_samples_split),
                                   min_samples_leaf=int(min_samples_leaf),
                                   criterion=cri,
                                  )
    #predictions = model.predict(x_test)
    scores=np.mean(cross_val_score(model, X, y, cv=3, n_jobs=-1,
                                    scoring="accuracy"))
    #return optunity.metrics.roc_auc(y_test, predictions, positive=True)
    return scores#optunity.metrics.accuracy(y_test, predictions)

optimal_configuration, info, _ = optunity.maximize(performance,
                                                   solver_name='particle swarm',
                                                   num_evals=20,
                                                    **search
                                                   )
print(optimal_configuration)
print("Accuracy:"+ str(info.optimum))
```

```
{'n_estimators': 48.57278714806698, 'max_features': 23.75307455666614, 'max_depth': 23.1889648437
```

```
Accuracy:0.9222778705249489
```

```
#SVM
import optunity
import optunity.metrics

data=X
labels=y.tolist()

search = {
    'C': (0,50),
    'kernel':[0,4]
         }
@optunity.cross_validated(x=data, y=labels, num_folds=3)
def performance(x_train, y_train, x_test, y_test,C=None,kernel=None):
    # fit the model
    if kernel<1:
        ke='linear'
    elif kernel<2:
        ke='poly'
    elif kernel<3:
        ke='rbf'
    else:
        ke='sigmoid'
    model = SVC(C=float(C),
                kernel=ke
                                  )
    #predictions = model.predict(x_test)
    scores=np.mean(cross_val_score(model, X, y, cv=3, n_jobs=-1,
                                   scoring="accuracy"))
    #return optunity.metrics.roc_auc(y_test, predictions, positive=True)
    return scores#optunity.metrics.accuracy(y_test, predictions)

optimal_configuration, info, _ = optunity.maximize(performance,
                                                   solver_name='particle swarm',
                                                   num_evals=20,
                                                       **search
                                                   )
print(optimal_configuration)
print("Accuracy:"+ str(info.optimum))
```

```
{'C': 15.2099609375, 'kernel': 2.435546875}
Accuracy:0.9738452977184195
```

```
#KNN
import optunity
import optunity.metrics

data=X
labels=y.tolist()

search = {
    'n_neighbors': [1, 20],
```

```
        }
@optunity.cross_validated(x=data, y=labels, num_folds=3)
def performance(x_train, y_train, x_test, y_test,n_neighbors=None):
    # fit the model
    model = KNeighborsClassifier(n_neighbors=int(n_neighbors),
                                 )
    scores=np.mean(cross_val_score(model, X, y, cv=3, n_jobs=-1,
                                   scoring="accuracy"))
    return scores

optimal_configuration, info, _ = optunity.maximize(performance,
                                                   solver_name='particle swarm',
                                                   num_evals=10,
                                                    **search
                                                   )
print(optimal_configuration)
print("Accuracy:"+ str(info.optimum))
```

```
{'n_neighbors': 3.49560546875}
Accuracy:0.9682804674457429
```