Hyperparameter Optimization (HPO) of Machine Learning Models

In [1]:
```
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_sc
from sklearn.neighbors import KNeighborsClassifier,KNeighborsRegressor
from sklearn.svm import SVC,SVR
from sklearn import datasets
import scipy.stats as stats
```

Load Boston Housing dataset

We will take the Housing dataset which contains information about different houses in Boston. There are 506 samples and 13 feature variables in this Boston dataset. The main goal is to predict the value of prices of the house using the given features.

In [2]:
```
import numpy as np
import pandas as pd

data_url = "http://lib.stat.cmu.edu/datasets/boston"
raw_df = pd.read_csv(data_url, sep="\s+", skiprows=22, header=None)

# Extract features and target variable
X = np.hstack([raw_df.values[::2, :], raw_df.values[1::2, :2]])
y = raw_df.values[1::2, 2]

# Create a DataFrame with features and target variable
columns = [f'feature_{i}' for i in range(X.shape[1])]
df = pd.DataFrame(data=X, columns=columns)
df['Price'] = y
```

Baseline Machine Learning Models: Regressors with Default Hyperparameters

Using 3-Fold Cross-Validation

In [3]:
```
#Random Forest
clf = RandomForestRegressor()
scores = cross_val_score(clf, X, y, cv=3,scoring='neg_mean_squared_error') # 
print("MSE:"+ str(-scores.mean()))
```

```
MSE:29.45697265416314
```

In [4]:
```
#SVM
clf = SVR()
scores = cross_val_score(clf, X, y, cv=3,scoring='neg_mean_squared_error')
print("MSE:"+ str(-scores.mean()))
```

```
MSE:77.42951812579331
```

In [5]:
```
#KNN
clf = KNeighborsRegressor()
scores = cross_val_score(clf, X, y, cv=3,scoring='neg_mean_squared_error')
print("MSE:"+ str(-scores.mean()))
```

```
MSE:81.48773186343571
```

HPO Algorithm 1: Grid Search

Search all the given hyper-parameter configurations

Advantages:

- Simple implementation.

Disadvantages:

- Time-consuming,
- Only efficient with categorical HPs.

In [6]:
```
#Random Forest
from sklearn.model_selection import GridSearchCV
# Define the hyperparameter configuration space
rf_params = {
    'n_estimators': [10, 20, 30],
    #'max_features': ['sqrt',0.5],
    'max_depth': [15,20,30,50],
    #'min_samples_leaf': [1,2,4,8],
    #"bootstrap":[True,False],
    #"criterion":['mse','mae']
}
clf = RandomForestRegressor(random_state=0)
grid = GridSearchCV(clf, rf_params, cv=3, scoring='neg_mean_squared_error')
grid.fit(X, y)
print(grid.best_params_)
print("MSE:"+ str(-grid.best_score_))
```

```
{'max_depth': 30, 'n_estimators': 10}
MSE:28.068229100920448
```

In [7]:
```
#SVM
from sklearn.model_selection import GridSearchCV
rf_params = {
    'C': [1,10, 100],
    "kernel":['poly','rbf','sigmoid'],
    "epsilon":[0.01,0.1,1]
}
clf = SVR(gamma='scale')
grid = GridSearchCV(clf, rf_params, cv=3, scoring='neg_mean_squared_error')
grid.fit(X, y)
print(grid.best_params_)
print("MSE:"+ str(-grid.best_score_))
```

```
{'C': 100, 'epsilon': 0.01, 'kernel': 'poly'}
MSE:67.07644831331959
```

In [8]:
```
#KNN
from sklearn.model_selection import GridSearchCV
rf_params = {
    'n_neighbors': [2, 3, 5, 7, 10]
}
clf = KNeighborsRegressor()
grid = GridSearchCV(clf, rf_params, cv=3, scoring='neg_mean_squared_error')
grid.fit(X, y)
print(grid.best_params_)
print("MSE:"+ str(-grid.best_score_))
```

```
{'n_neighbors': 5}
MSE:81.48773186343571
```

HPO Algorithm 2: Random Search

Randomly search hyper-parameter combinations in the search space

Advantages:

- More efficient than GS.
- Enable parallelization.

Disadvantages:

- Not consider previous results.
- Not efficient with conditional HPs.

In [9]:

```
#Random Forest
from scipy.stats import randint as sp_randint
from sklearn.model_selection import RandomizedSearchCV
# Define the hyperparameter configuration space
rf_params = {
    'n_estimators': sp_randint(10,100),
    "max_features":sp_randint(1,13),
    'max_depth': sp_randint(5,50),
    "min_samples_split":sp_randint(2,11),
    "min_samples_leaf":sp_randint(1,11),
    "criterion":['squared_error']
}
n_iter_search=20 #number of iterations is set to 20, you can increase this nur
clf = RandomForestRegressor(random_state=0)
Random = RandomizedSearchCV(clf, param_distributions=rf_params,n_iter=n_iter_s
Random.fit(X, y)
print(Random.best_params_)
print("MSE:"+ str(-Random.best_score_))
```

```
{'criterion': 'squared_error', 'max_depth': 26, 'max_features': 9, 'min_samp
les_leaf': 4, 'min_samples_split': 3, 'n_estimators': 99}
MSE:27.3384038771797
```

In [10]:

```
#SVM
from scipy.stats import randint as sp_randint
from sklearn.model_selection import RandomizedSearchCV
rf_params = {
    'C': stats.uniform(0,50),
    "kernel":['poly','rbf','sigmoid'],
    "epsilon":stats.uniform(0,1)
}
n_iter_search=20
clf = SVR(gamma='scale')
Random = RandomizedSearchCV(clf, param_distributions=rf_params,n_iter=n_iter_s
Random.fit(X, y)
print(Random.best_params_)
print("MSE:"+ str(-Random.best_score_))
```

```
{'C': 23.30432587063403, 'epsilon': 0.34293172061229715, 'kernel': 'poly'}
MSE:60.84236393967016
```

In [11]:
```
#KNN
from scipy.stats import randint as sp_randint
from sklearn.model_selection import RandomizedSearchCV
rf_params = {
    'n_neighbors': sp_randint(1,20),
}
n_iter_search=10
clf = KNeighborsRegressor()
Random = RandomizedSearchCV(clf, param_distributions=rf_params,n_iter=n_iter_s
Random.fit(X, y)
print(Random.best_params_)
print("MSE:"+ str(-Random.best_score_))
```

```
{'n_neighbors': 13}
MSE:80.74121499347262
```