31. Data Science – Machine Learning – GridSearchCV

Contents

1. Introduction 2

2. Hyper parameter tuning 3

3. Problem and solution 4

4. Hyper parameter tuning 5

5. Kernel Functions 6

6. Ways to tune parameters 8

7. RandomizedSearchCV 18

31. Data Science – Machine Learning – GridSearchCV

1. Introduction

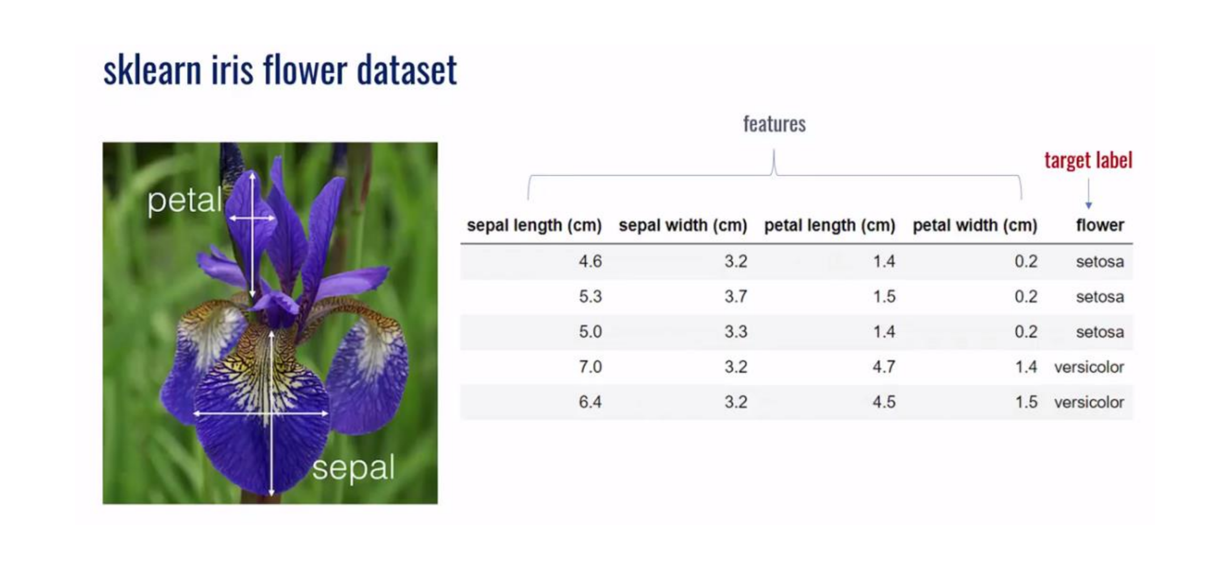
- ✓ While building a Machine learning model we always define two things that are,
  - o Model parameters
  - o Model hyperparameters of a predictive algorithm.
- ✓ Model parameters are the ones that are an internal part of the model and their value is computed automatically.
  - o Support vector machine.
- ✓ Hyperparameters are the ones that can be manipulated by the programmer to improve the performance of the model like the learning rate.
- ✓ Different parts of a hyperparameter approaches,
  - o GridSearchCV
  - o RandomizedSearchCV

2. Hyper parameter tuning

- ✓ Hyperparameter tuning is the process of tuning the parameters while building machine learning models.
- ✓ These parameters are defined by programmer wish; machine learning algorithms never learn these parameters.
- ✓ If parameters are tuned then we will get good performance to the model
- ✓ We define the hyperparameter as shown below for the random forest classifier model.
- ✓ These parameters are tuned randomly and results are checked.

- RandomForestRegressor(bootstrap=True, ccp_alpha=0.0, criterion='mse', max_depth=None, max_features='auto', max_leaf_nodes=None, max_samples=None, min_impurity_decrease=0.0, min_impurity_split=None, min_samples_leaf=1, min_samples_split=2, min_weight_fraction_leaf=0.0, n_estimators=100, n_jobs=None, oob_score=False, random_state=None, verbose=0, warm_start=False)

3. Problem and solution



sklearn iris flower dataset

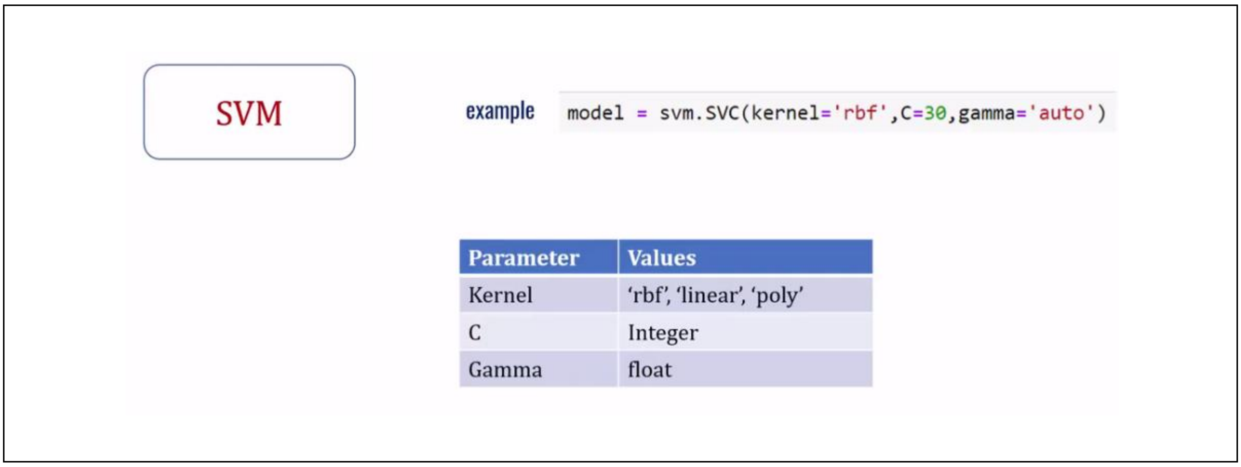
sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	flower
4.6	3.2	1.4	0.2	setosa
5.3	3.7	1.5	0.2	setosa
5.0	3.3	1.4	0.2	setosa
7.0	3.2	4.7	1.4	versicolor
6.4	3.2	4.5	1.5	versicolor

- ✓ Above problem we can solve by using different algorithms
  - ○ LogisticRegression
  - ○ SVM
  - ○ Decision Tree
  - ○ RandomForest
  - ○ NaiveBayes

- ✓ If we are trying to use SVM then it having different parameters

4. Hyper parameter tuning

- ✓ The process of choosing parameters called as hyper parameter tuning



SVM

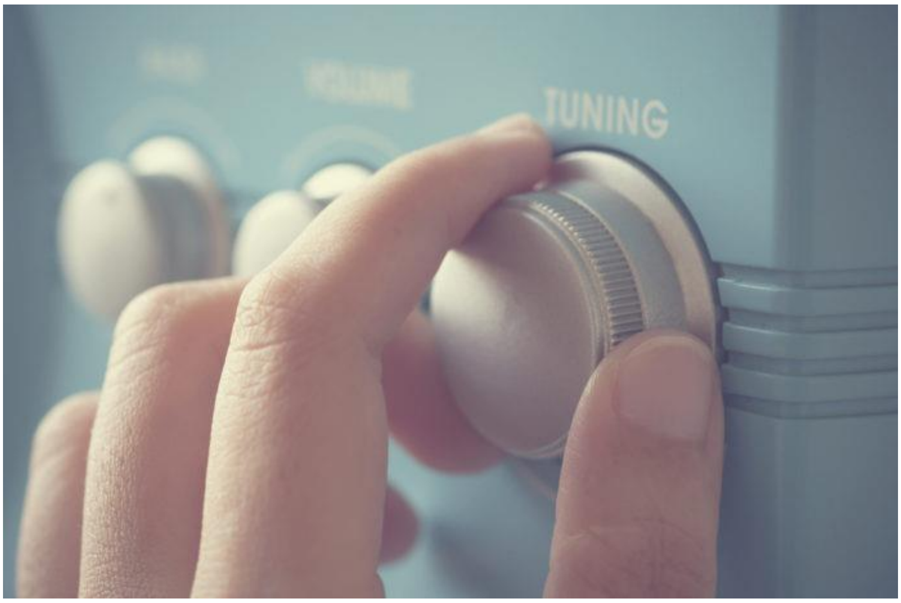
example

```
model = svm.SVC(kernel='rbf',C=30,gamma='auto')
```

Parameter	Values
Kernel	‘rbf’, ‘linear’, ‘poly’
C	Integer
Gamma	float

5. Kernel Functions

- ✓ SVM algorithms use a set of mathematical functions that are defined as the kernel.
    - The function of kernel is to take data as input and transform it into the required form.
    - For example linear, nonlinear, polynomial, radial basis function (RBF), and sigmoid.

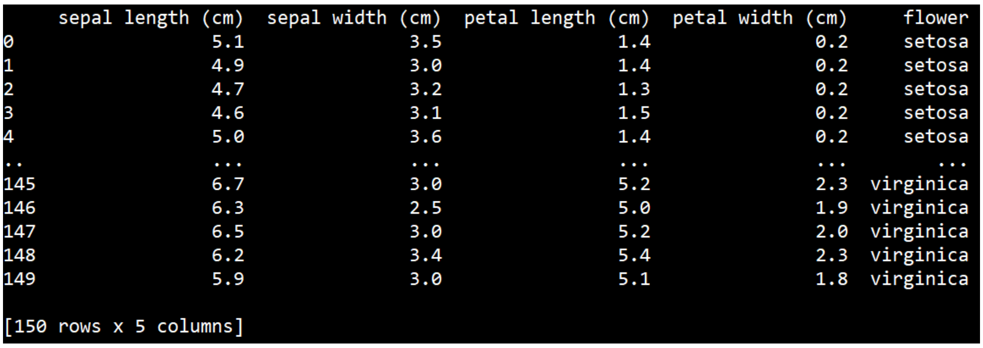


Program Name

Loading iris dataset
demo1.py

```
from sklearn import datasets
import pandas as pd

iris = datasets.load_iris()

df = pd.DataFrame(iris.data, columns=iris.feature_names)
df['flower'] = iris.target
df['flower'] = df['flower'].apply(lambda x: iris.target_names[x])

print(df)
```

Output



```
     sepal length (cm)  sepal width (cm)  petal length (cm)  petal width (cm)     flower
0                  5.1               3.5                1.4               0.2     setosa
1                  4.9               3.0                1.4               0.2     setosa
2                  4.7               3.2                1.3               0.2     setosa
3                  4.6               3.1                1.5               0.2     setosa
4                  5.0               3.6                1.4               0.2     setosa
..                 ...               ...                ...               ...        ...
145                6.7               3.0                5.2               2.3  virginica
146                6.3               2.5                5.0               1.9  virginica
147                6.5               3.0                5.2               2.0  virginica
148                6.2               3.4                5.4               2.3  virginica
149                5.9               3.0                5.1               1.8  virginica

[150 rows x 5 columns]
```

6. Ways to tune parameters

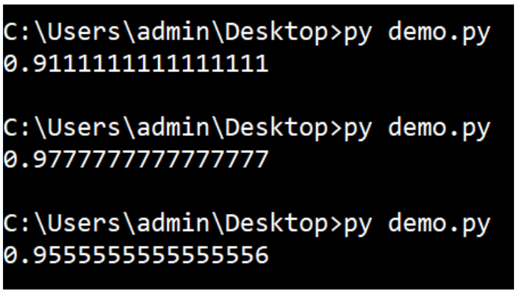
- ✓ Approach 1: Use train_test_split and manually tune parameters by trial and error
- ✓ Approach 2: Use K Fold Cross validation
- ✓ Approach 3: Use GridSearchCV

Program Approach 1: Use train_test_split and manually tune parameters by trial and error

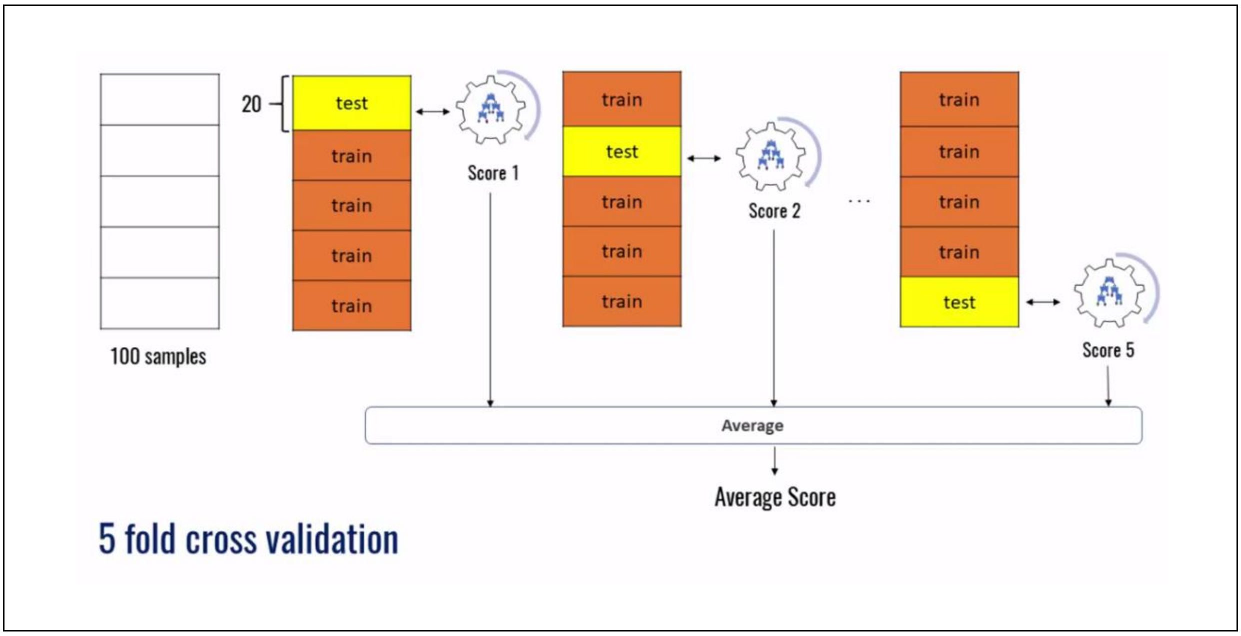
Name demo2.py

```
from sklearn import svm, datasets
import pandas as pd
from sklearn.model_selection import train_test_split

iris = datasets.load_iris()

df = pd.DataFrame(iris.data, columns=iris.feature_names)
df['flower'] = iris.target
df['flower'] = df['flower'].apply(lambda x: iris.target_names[x])

X_train, X_test, y_train, y_test = train_test_split(iris.data,
iris.target, test_size = 0.3)

model = svm.SVC(kernel = 'rbf', C = 30, gamma = 'auto')
model.fit(X_train, y_train)

print(model.score(X_test, y_test))
```

Output



```
C:\Users\admin\Desktop>py demo.py
0.9111111111111111

C:\Users\admin\Desktop>py demo.py
0.9777777777777777

C:\Users\admin\Desktop>py demo.py
0.9555555555555556
```



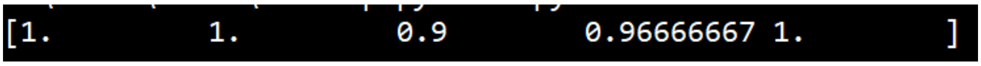
20
test
train
train
train
train
Score 1
train
test
train
train
train
Score 2
. . .
train
train
train
train
test
Score 5
100 samples
Average
Average Score
5 fold cross validation

Program Name

Approach 2: Use K Fold Cross validation
demo3.py

```
from sklearn import svm, datasets
import pandas as pd
from sklearn.model_selection import cross_val_score

iris = datasets.load_iris()

df = pd.DataFrame(iris.data, columns=iris.feature_names)
df['flower'] = iris.target
df['flower'] = df['flower'].apply(lambda x: iris.target_names[x])

sc1 = cross_val_score(svm.SVC(kernel='linear', C=10,
gamma='auto'), iris.data, iris.target, cv=5)

print(sc1)
```

Output



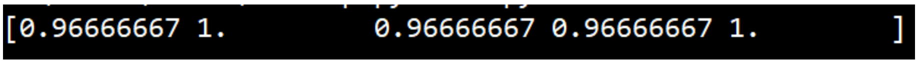
```
[1.         1.         0.9        0.96666667 1.        ]
```

Program Name

Approach 2: Use K Fold Cross validation
demo4.py

```
from sklearn import svm, datasets
import pandas as pd
from sklearn.model_selection import cross_val_score

iris = datasets.load_iris()

df = pd.DataFrame(iris.data, columns=iris.feature_names)
df['flower'] = iris.target
df['flower'] = df['flower'].apply(lambda x: iris.target_names[x])

sc2 = cross_val_score(svm.SVC(kernel='rbf', C=10, gamma='auto'),
iris.data, iris.target, cv=5)

print(sc2)
```

Output



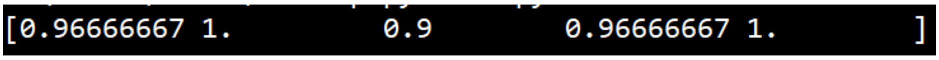
```
[0.96666667 1.         0.96666667 0.96666667 1.        ]
```

Program Name

Approach 2: Use K Fold Cross validation
demo5.py

```
from sklearn import svm, datasets
import pandas as pd
from sklearn.model_selection import cross_val_score

iris = datasets.load_iris()

df = pd.DataFrame(iris.data, columns=iris.feature_names)
df['flower'] = iris.target
df['flower'] = df['flower'].apply(lambda x: iris.target_names[x])

sc3 = cross_val_score(svm.SVC(kernel='rbf', C=20, gamma='auto'),
iris.data, iris.target, cv=5)

print(sc3)
```

Output



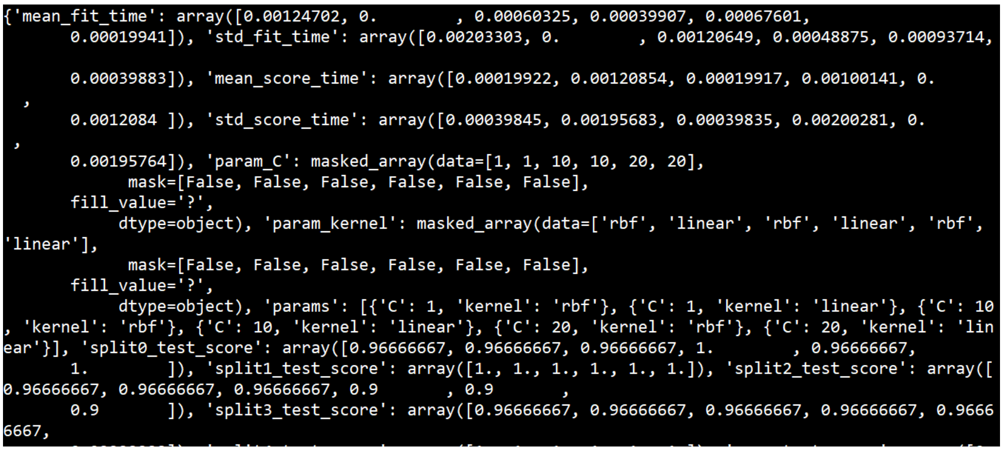
```
[0.96666667 1.         0.9        0.96666667 1.        ]
```

Program Name

Approach: Use GridSearchCV
demo6.py

```
from sklearn import svm, datasets
import pandas as pd
from sklearn.model_selection import GridSearchCV

iris = datasets.load_iris()

df = pd.DataFrame(iris.data, columns=iris.feature_names)
df['flower'] = iris.target
df['flower'] = df['flower'].apply(lambda x: iris.target_names[x])

d = {
   'C': [1, 10, 20],
   'kernel': ['rbf', 'linear']
}

clf = GridSearchCV(svm.SVC(gamma='auto'), d, cv = 5,
return_train_score = False)

clf.fit(iris.data, iris.target)

print(clf.cv_results_)
```

Output



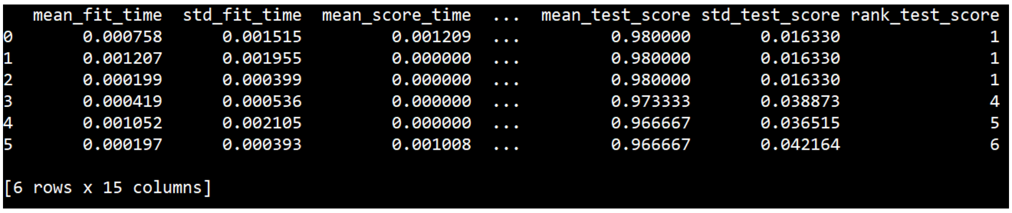
```
{'mean_fit_time': array([0.00124702, 0.        , 0.00060325, 0.00039907, 0.00067601,
       0.00019941]), 'std_fit_time': array([0.00203303, 0.        , 0.00120649, 0.00048875, 0.00093714,

       0.00039883]), 'mean_score_time': array([0.00019922, 0.00120854, 0.00019917, 0.00100141, 0.
  ,
       0.0012084 ]), 'std_score_time': array([0.00039845, 0.00195683, 0.00039835, 0.00200281, 0.
 ,
       0.00195764]), 'param_C': masked_array(data=[1, 1, 10, 10, 20, 20],
             mask=[False, False, False, False, False, False],
       fill_value='?',
            dtype=object), 'param_kernel': masked_array(data=['rbf', 'linear', 'rbf', 'linear', 'rbf',
'linear'],
             mask=[False, False, False, False, False, False],
       fill_value='?',
            dtype=object), 'params': [{'C': 1, 'kernel': 'rbf'}, {'C': 1, 'kernel': 'linear'}, {'C': 10
, 'kernel': 'rbf'}, {'C': 10, 'kernel': 'linear'}, {'C': 20, 'kernel': 'rbf'}, {'C': 20, 'kernel': 'lin
ear'}], 'split0_test_score': array([0.96666667, 0.96666667, 0.96666667, 1.        , 0.96666667,
       1.        ]), 'split1_test_score': array([1., 1., 1., 1., 1., 1.]), 'split2_test_score': array([
0.96666667, 0.96666667, 0.96666667, 0.9       , 0.9       ,
       0.9       ]), 'split3_test_score': array([0.96666667, 0.96666667, 0.96666667, 0.96666667, 0.9666
6667,
```

Program Name

Approach: Use GridSearchCV, creating DataFrame with results
demo7.py

```
from sklearn import svm, datasets
import pandas as pd
from sklearn.model_selection import GridSearchCV

iris = datasets.load_iris()

df = pd.DataFrame(iris.data, columns=iris.feature_names)
df['flower'] = iris.target
df['flower'] = df['flower'].apply(lambda x: iris.target_names[x])

d = {
   'C': [1, 10, 20],
   'kernel': ['rbf', 'linear']
}

clf = GridSearchCV(svm.SVC(gamma = 'auto'), d, cv = 5,
return_train_score = False)

clf.fit(iris.data, iris.target)

df = pd.DataFrame(clf.cv_results_)

print(df)
```

Output



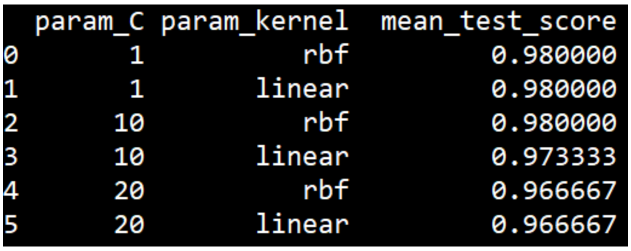
```
   mean_fit_time  std_fit_time  mean_score_time  ...  mean_test_score std_test_score rank_test_score
0       0.000758      0.001515         0.001209  ...         0.980000       0.016330               1
1       0.001207      0.001955         0.000000  ...         0.980000       0.016330               1
2       0.000199      0.000399         0.000000  ...         0.980000       0.016330               1
3       0.000419      0.000536         0.000000  ...         0.973333       0.038873               4
4       0.001052      0.002105         0.000000  ...         0.966667       0.036515               5
5       0.000197      0.000393         0.001008  ...         0.966667       0.042164               6

[6 rows x 15 columns]
```

Program Name

Approach: Use GridSearchCV, creating DataFrame with results
demo8.py

```
from sklearn import svm, datasets
import pandas as pd
from sklearn.model_selection import GridSearchCV

iris = datasets.load_iris()

df = pd.DataFrame(iris.data, columns=iris.feature_names)
df['flower'] = iris.target
df['flower'] = df['flower'].apply(lambda x: iris.target_names[x])

d = {
   'C': [1,10,20],
   'kernel': ['rbf', 'linear']
}

clf = GridSearchCV(svm.SVC(gamma='auto'), d, cv=5,
return_train_score = False)

clf.fit(iris.data, iris.target)

df = pd.DataFrame(clf.cv_results_)

result = df[['param_C', 'param_kernel', 'mean_test_score']]

print(result)
```

Output



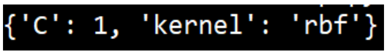
```
  param_C param_kernel  mean_test_score
0       1          rbf         0.980000
1       1       linear         0.980000
2      10          rbf         0.980000
3      10       linear         0.973333
4      20          rbf         0.966667
5      20       linear         0.966667
```

Program Name

Approach: Use GridSearchCV.
demo9.py

```
from sklearn import svm, datasets
import pandas as pd
from sklearn.model_selection import GridSearchCV

iris = datasets.load_iris()

df = pd.DataFrame(iris.data, columns=iris.feature_names)
df['flower'] = iris.target
df['flower'] = df['flower'].apply(lambda x: iris.target_names[x])

d = {
   'C': [1,10,20],
   'kernel': ['rbf', 'linear']
}

clf = GridSearchCV(svm.SVC(gamma='auto'), d, cv=5,
return_train_score = False)

clf.fit(iris.data, iris.target)

df = pd.DataFrame(clf.cv_results_)

print(clf.best_params_)
```

Output



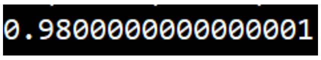
```
{'C': 1, 'kernel': 'rbf'}
```

Program Name

Approach: Use GridSearchCV.
demo10.py

```
from sklearn import svm, datasets
import pandas as pd
from sklearn.model_selection import GridSearchCV

iris = datasets.load_iris()

df = pd.DataFrame(iris.data, columns=iris.feature_names)
df['flower'] = iris.target
df['flower'] = df['flower'].apply(lambda x: iris.target_names[x])

d = {
   'C': [1,10,20],
   'kernel': ['rbf', 'linear']
}

clf = GridSearchCV(svm.SVC(gamma='auto'), d, cv=5,
return_train_score = False)

clf.fit(iris.data, iris.target)

df = pd.DataFrame(clf.cv_results_)

print(clf.best_score_)
```

Output



```
0.9800000000000001
```

7. RandomizedSearchCV

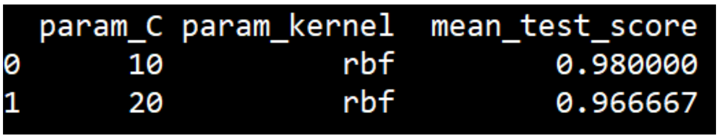
- ✓ Use RandomizedSearchCV to reduce number of iterations and with random combination of parameters.
- ✓ This is useful when you have too many parameters to try and your training time is longer. It helps reduce the cost of computation

Program Name

Approach: Use RandomizedSearchCV
demo11.py

```
from sklearn import svm, datasets
import pandas as pd
from sklearn.model_selection import RandomizedSearchCV

iris = datasets.load_iris()

df = pd.DataFrame(iris.data, columns=iris.feature_names)
df['flower'] = iris.target
df['flower'] = df['flower'].apply(lambda x: iris.target_names[x])

d = {
        'C': [1, 10, 20],
        'kernel': ['rbf', 'linear']
    }

rs = RandomizedSearchCV(svm.SVC(gamma = 'auto'), d, cv = 5,
    return_train_score = False,
    n_iter=2
)

rs.fit(iris.data, iris.target)
cols = ['param_C', 'param_kernel', 'mean_test_score']
df = pd.DataFrame(rs.cv_results_)[cols]

print(df)
```

Output



```
  param_C param_kernel  mean_test_score
0      10          rbf         0.980000
1      20          rbf         0.966667
```