Finding best model and hyper parameter tunning using GridSearchCV

For iris flower dataset in sklearn library, we are going to find out best model and best hyper parameters using GridSearchCV

Load iris flower dataset

In [51]:
```
from sklearn import svm,datasets
iris = datasets.load_iris()
```

In [52]:
```
import pandas as pd
import numpy as np
df = pd.DataFrame(iris.data,columns=iris.feature_names)
df['flower'] = iris.target
df['flower'] = df['flower'].apply(lambda x: iris.target_names[x])
df[47:150]
```

Out[52]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	flower
47	4.6	3.2	1.4	0.2	setosa
48	5.3	3.7	1.5	0.2	setosa
49	5.0	3.3	1.4	0.2	setosa
50	7.0	3.2	4.7	1.4	versicolor
51	6.4	3.2	4.5	1.5	versicolor
...	...	...	...	...	...
145	6.7	3.0	5.2	2.3	virginica
146	6.3	2.5	5.0	1.9	virginica

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	flower
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

103 rows × 5 columns

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

In [53]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(iris.data, iris.tar
get, test_size=0.3)
```

In [54]:
```
model = svm.SVC(kernel='rbf',C=30,gamma='auto')
model.fit(X_train,y_train)
model.score(X_test, y_test)
```

Out[54]:
```
0.9777777777777777
```

Approach 2: Use K Fold Cross validation

Manually try suppling models with different parameters to cross_val_score function with 5 fold cross validation

In [43]:
```
from sklearn.model_selection import cross_val_score
```

In [44]:
```
cross_val_score(svm.SVC(kernel="linear",C=10,gamma='auto'),iris.data,ir
is.target,cv=5)
```

Out[44]:
```
array([1.        , 1.        , 0.9       , 0.96666667, 1.        ])
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

In [45]:
```
cross_val_score(svm.SVC(kernel="rbf",C=10,gamma='auto'),iris.data,iris.target,cv=5)
```

Out[45]:
```
array([0.96666667, 1.        , 0.96666667, 0.96666667, 1.        ])
```

In [46]:
```
cross_val_score(svm.SVC(kernel="rbf",C=20,gamma='auto'),iris.data,iris.target,cv=5)
```

Out[46]:
```
array([0.96666667, 1.        , 0.9       , 0.96666667, 1.        ])
```

Above approach is tiresome and very manual. We can use for loop as an alternative

In [55]:
```
kernels = ['rbf', 'linear']
C = [1,10,20]
avg_scores = {}
for kval in kernels:
    for cval in C:
        cv_scores = cross_val_score(svm.SVC(kernel=kval,C=cval,gamma='auto'),iris.data, iris.target, cv=5)
        avg_scores[kval + '_' + str(cval)]=np.average(cv_scores)

avg_scores
```

Out[55]:
```
{'rbf_1': 0.9800000000000001,
 'rbf_10': 0.9800000000000001,
 'rbf_20': 0.9666666666666668,
 'linear_1': 0.9800000000000001,
 'linear_10': 0.9733333333333334,
 'linear_20': 0.9666666666666666}
```

From above results we can say that rbf with C=1 or 10 or linear with C=1 will give best performance

Approach 3: Use GridSearchSV

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

GridSearchCV does exactly same thing as for loop above but in a single line of code

In [56]:
```
from sklearn.model_selection import GridSearchCV
clf = GridSearchCV(svm.SVC(gamma='auto'), {
    'C': [1,10,20],
    'kernel': ['rbf','linear']
}, cv=5, return_train_score=False)
clf.fit(iris.data, iris.target)
clf.cv_results_
```

Out[56]:
```
{'mean_fit_time': array([0.00299001, 0.00099287, 0.00079889, 0.0008015
6, 0.00039978,
        0.0009903 ]),
 'std_fit_time': array([1.30846090e-03, 6.17344729e-04, 3.99452991e-04,
4.00786922e-04,
        4.89628932e-04, 1.91470159e-05]),
 'mean_score_time': array([0.0015913 , 0.00039978, 0.00060115, 0.000200
03, 0.00098834,
        0.00079966]),
 'std_score_time': array([4.83022969e-04, 4.89630326e-04, 4.90840050e-0
4, 4.00066376e-04,
        2.70709992e-05, 3.99828455e-04]),
 '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': 'linear'}],
 'split0_test_score': array([0.96666667, 0.96666667, 0.96666667, 1.
    , 0.96666667,
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

```
        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, 0.9666
6667, 0.96666667,
        0.93333333]),
 'split4_test_score': array([1., 1., 1., 1., 1., 1.]),
 'mean_test_score': array([0.98      , 0.98      , 0.98      , 0.973333
33, 0.96666667,
        0.96666667]),
 'std_test_score': array([0.01632993, 0.01632993, 0.01632993, 0.0388730
1, 0.03651484,
        0.0421637 ]),
 'rank_test_score': array([1, 1, 1, 4, 5, 6])}
```

In [57]:

```
df = pd.DataFrame(clf.cv_results_)
df
```

Out[57]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_C	param_kernel	params
0	0.002990	0.001308	0.001591	0.000483	1	rbf	{'C': 1, 'kernel': 'rbf'}
1	0.000993	0.000617	0.000400	0.000490	1	linear	{'C': 1, 'kernel': 'linear'}
2	0.000799	0.000399	0.000601	0.000491	10	rbf	{'C': 10, 'kernel': 'rbf'}
3	0.000802	0.000401	0.000200	0.000400	10	linear	{'C': 10, 'kernel': 'linear'}
4	0.000400	0.000490	0.000988	0.000027	20	rbf	{'C': 20, 'kernel': 'rbf'}

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_C	param_kernel	params
5	0.000990	0.000019	0.000800	0.000400	20	linear	{'C': 20, 'kernel': 'linear'}

In [58]:
```
df[['param_C','param_kernel','mean_test_score']]
```

Out[58]:

	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

In [59]:
```
clf.best_params_
```

Out[59]:
```
{'C': 1, 'kernel': 'rbf'}
```

In [60]:
```
clf.best_score_
```

Out[60]:
```
0.9800000000000001
```

In [61]:
```
dir(clf)
```

Out[61]:
```
['__abstractmethods__',
 '__class__',
 '__delattr__',
 '__dict__',
 '__dir__',
 '__doc__',
 '__eq__',
 '__format__',
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

```
'__ge__',
'__getattribute__',
'__getstate__',
'__gt__',
'__hash__',
'__init__',
'__init_subclass__',
'__le__',
'__lt__',
'__module__',
'__ne__',
'__new__',
'__reduce__',
'__reduce_ex__',
'__repr__',
'__setattr__',
'__setstate__',
'__sizeof__',
'__str__',
'__subclasshook__',
'__weakref__',
'_abc_impl',
'_check_is_fitted',
'_estimator_type',
'_format_results',
'_get_param_names',
'_get_tags',
'_more_tags',
'_pairwise',
'_required_parameters',
'_run_search',
'best_estimator_',
'best_index_',
'best_params_',
'best_score_',
'classes_',
'cv',
'cv_results_',
'decision_function',
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

```
 'error_score',
 'estimator',
 'fit',
 'get_params',
 'iid',
 'inverse_transform',
 'multimetric_',
 'n_jobs',
 'n_splits_',
 'param_grid',
 'pre_dispatch',
 'predict',
 'predict_log_proba',
 'predict_proba',
 'refit',
 'refit_time_',
 'return_train_score',
 'score',
 'scorer_',
 'scoring',
 'set_params',
 'transform',
 'verbose']
```

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

In [62]:
```
from sklearn.model_selection import RandomizedSearchCV
rs = RandomizedSearchCV(svm.SVC(gamma='auto'), {
        'C': [1,10,20],
        'kernel': ['rbf','linear']
    },
    cv=5,
    return_train_score=False,
    n_iter=2
)
rs.fit(iris.data, iris.target)
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

```
pd.DataFrame(rs.cv_results_)[['param_C','param_kernel','mean_test_scor
e']]
```

Out[62]:

	param_C	param_kernel	mean_test_score
0	1	rbf	0.98
1	1	linear	0.98

How about different models with different hyperparameters?

In [64]:

```
from sklearn import svm
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression

model_params = {
    'svm': {
        'model': svm.SVC(gamma='auto'),
        'params' : {
            'C': [1,10,20],
            'kernel': ['rbf','linear']
        }
    },
    'random_forest': {
        'model': RandomForestClassifier(),
        'params' : {
            'n_estimators': [1,5,10]
        }
    },
    'logistic_regression' : {
        'model': LogisticRegression(solver='liblinear',multi_class='aut
o'),
        'params': {
            'C': [1,5,10]
        }
    }
}
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

In [65]:

```
scores = []

for model_name, mp in model_params.items():
    clf =  GridSearchCV(mp['model'], mp['params'], cv=5, return_train_s
core=False)
    clf.fit(iris.data, iris.target)
    scores.append({
        'model': model_name,
        'best_score': clf.best_score_,
        'best_params': clf.best_params_
    })

df = pd.DataFrame(scores,columns=['model','best_score','best_params'])
df
```

Out[65]:

	model	best_score	best_params
0	svm	0.980000	{'C': 1, 'kernel': 'rbf'}
1	random_forest	0.960000	{'n_estimators': 5}
2	logistic_regression	0.966667	{'C': 5}

Based on above, I can conclude that SVM with C=1 and kernel='rbf' is the best model for solving my problem of iris flower classification

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD