In [1]:

```
from sklearn import svm,datasets
iris = datasets.load_iris()
```

In [2]:

```
import pandas as pd
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[47:52]
```

Out[2]:

	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

In [3]:

```
#traditional method for split
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test = train_test_split(iris.data,iris.target,test_size=0.3)
```

In [4]:

```
#randomly initialize the parameter
model = svm.SVC(kernel='rbf',C=30,gamma='auto')
model.fit(X_train,y_train)
model.score(X_test, y_test)
```

Out[4]:

```
0.9777777777777777
```

As here the value change as we refresh the train test split so we use K-fold Cross validatioj

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 [5]:

```
from sklearn.model_selection import cross_val_score
```

In [6]:

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

Out[6]:

```
array([1.        , 1.        , 0.9       , 0.96666667, 1.        ])
```

In [7]:

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

Out[7]:

```
array([0.96666667, 1.        , 0.96666667, 0.96666667, 1.        ])
```

In [8]:

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

Out[8]:

```
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 [9]:

```
import numpy as np
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, iri
        avg_scores[kval + '_' + str(cval)] = np.average(cv_scores)

avg_scores
```

Out[9]:

```
{'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 GridSearchCV

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

In [10]:

```
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[10]:

```
{'mean_fit_time': array([0.00230327, 0.00090079, 0.00100074, 0.        , 0.
,
        0.        ]),
 'std_fit_time': array([0.00244543, 0.00019994, 0.00200148, 0.        , 0.
,
        0.        ]),
 'mean_score_time': array([0.00120039, 0.00050054, 0.        , 0.00100007,
0.00100012,
        0.00199981]),
 'std_score_time': array([1.91470652e-03, 1.16800773e-07, 0.00000000e+00, 2.
00014114e-03,
        2.00023651e-03, 2.44925390e-03]),
 '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,
        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,
0.96666667,
        0.93333333]),
 'split4_test_score': array([1., 1., 1., 1., 1., 1.]),
 'mean_test_score': array([0.98      , 0.98      , 0.98      , 0.97333333,
0.96666667,
        0.96666667]),
 'std_test_score': array([0.01632993, 0.01632993, 0.01632993, 0.03887301, 0.
03651484,
        0.0421637 ]),
 'rank_test_score': array([1, 1, 1, 4, 5, 6])}
```

In [11]:

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

Out[11]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_C	param_kernel	parar
0	0.002303	0.002445	0.001200	1.914707e-03	1	rbf	{'C': 'kern 'rb
1	0.000901	0.000200	0.000501	1.168008e-07	1	linear	{'C': 'kern 'linea
2	0.001001	0.002001	0.000000	0.000000e+00	10	rbf	{'C': 1 'kern 'rb
3	0.000000	0.000000	0.001000	2.000141e-03	10	linear	{'C': 1 'kern 'linea
4	0.000000	0.000000	0.001000	2.000237e-03	20	rbf	{'C': 2 'kern 'rb
5	0.000000	0.000000	0.002000	2.449254e-03	20	linear	{'C': 2 'kern 'linea

In [12]:

```
df[['param_C','param_kernel','mean_test_score']]
```

Out[12]:

	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 [13]:

```
clf.best_params_
```

Out[13]:

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

In [14]:

```
clf.best_score_
```

Out[14]:

```
0.9800000000000001
```

In [15]:

```
#dir(clf)
```

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 [16]:

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

Out[16]:

	param_C	param_kernel	mean_test_score
0	10	rbf	0.98
1	1	linear	0.98

How about different models with different hyperparameters?

In [17]:

```
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='auto'),
        'params': {
            'C': [1,5,10]
        }
    }
}
```

In [18]:

```
#use grid search
scores = []

for model_name, mp in model_params.items():
    clf =  GridSearchCV(mp['model'], mp['params'], cv=5, return_train_score=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[18]:

	model	best_score	best_params
0	svm	0.980000	{'C': 1, 'kernel': 'rbf'}
1	random_forest	0.966667	{'n_estimators': 10}
2	logistic_regression	0.966667	{'C': 5}

In [19]:

```
#use Randomizesearch
scores = []

for model_name, mp in model_params.items():
    clf =  RandomizedSearchCV(mp['model'], mp['params'], cv=5, return_train_score=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
```

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\model_selection\_search.p
y:281: UserWarning: The total space of parameters 6 is smaller than n_iter=1
0. Running 6 iterations. For exhaustive searches, use GridSearchCV.
  % (grid_size, self.n_iter, grid_size), UserWarning)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\model_selection\_search.p
y:281: UserWarning: The total space of parameters 3 is smaller than n_iter=1
0. Running 3 iterations. For exhaustive searches, use GridSearchCV.
  % (grid_size, self.n_iter, grid_size), UserWarning)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\model_selection\_search.p
y:281: UserWarning: The total space of parameters 3 is smaller than n_iter=1
0. Running 3 iterations. For exhaustive searches, use GridSearchCV.
  % (grid_size, self.n_iter, grid_size), UserWarning)
```

Out[19]:

	model	best_score	best_params
0	svm	0.980000	{'kernel': 'rbf', 'C': 1}
1	random_forest	0.960000	{'n_estimators': 10}
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

Example 2

For digits dataset in sklearn.dataset, please try following classifiers and find out the one that gives best performance. Also find the optimal parameters for that classifier.

In [20]:

```
from sklearn import datasets
digits = datasets.load_digits()
```

In [21]:

```
from sklearn import svm
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import GaussianNB
from sklearn.naive_bayes import MultinomialNB
from sklearn.tree import DecisionTreeClassifier
```

In [22]:

```
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='auto'),
        'params': {
            'C': [1,5,10]
        }
    },
    'naive_bayes_gaussian': {
        'model': GaussianNB(),
        'params': {}
    },
    'naive_bayes_multinomial': {
        'model': MultinomialNB(),
        'params': {}
    },
    'decision_tree': {
        'model': DecisionTreeClassifier(),
        'params': {
            'criterion': ['gini','entropy'],

        }
    }
}
```

In [23]:

```
from sklearn.model_selection import GridSearchCV
import pandas as pd
scores = []

for model_name, mp in model_params.items():
    clf =  GridSearchCV(mp['model'], mp['params'], cv=5, return_train_score=False)
    clf.fit(digits.data, digits.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[23]:

	model	best_score	best_params
0	svm	0.947697	{'C': 1, 'kernel': 'linear'}
1	random_forest	0.892621	{'n_estimators': 10}
2	logistic_regression	0.922114	{'C': 1}
3	naive_bayes_gaussian	0.806928	{}
4	naive_bayes_multinomial	0.870350	{}
5	decision_tree	0.803576	{'criterion': 'entropy'}

In [ ]:

```

```