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[47:150]
```

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
...	...	...	...	...	...
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

103 rows × 5 columns

In [6]:

```
#Traditional Method of Train and 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 [7]:

```
# randomly initialize these parameters, at this point i am not confident about best paramet
# score changes every time with different samples
model = svm.SVC(kernel='rbf',C=30,gamma='auto')
model.fit(X_train,y_train)
model.score(X_test, y_test)
```

Out[7]:

```
0.9333333333333333
```

In [10]:

```
# finding optimal value for parameters
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([0.00199585, 0.00079746, 0.00079756, 0.00039783,
0.00079789,
        0.00060554]),
 'std_fit_time': array([0.00063309, 0.00039873, 0.00039878, 0.00048724, 0.
00039895,
        0.00049455]),
 'mean_score_time': array([0.00059714, 0.00059834, 0.00019951, 0.00059838,
0.00059152,
        0.00019965]),
 'std_score_time': array([0.00048758, 0.00048854, 0.00039902, 0.00048858,
0.00048311,
        0.0003993 ]),
 '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', 'rb
f', '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, 0.9666666
7, 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]:

```
# Results above are not easy to view, so lets import these results into dataframe
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	paran
0	0.001996	0.000633	0.000597	0.000488	1	rbf	{'C': 'kerne 'rb
1	0.000797	0.000399	0.000598	0.000489	1	linear	{'C': 'kerne 'linea
2	0.000798	0.000399	0.000200	0.000399	10	rbf	{'C': 1 'kerne 'rb
3	0.000398	0.000487	0.000598	0.000489	10	linear	{'C': 1 'kerne 'linea
4	0.000798	0.000399	0.000592	0.000483	20	rbf	{'C': 2 'kerne 'rb
5	0.000606	0.000495	0.000200	0.000399	20	linear	{'C': 2 'kerne 'linea

In [12]:

```
#looking for only parameter vales and mean score
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]:

```
#Finding best parameter combination
clf.best_params_
```

Out[13]:

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

In [14]:

```
clf.best_score_
```

Out[14]:

```
0.9800000000000001
```

In [18]:

```
dir(clf)
```

Out[18]:

```
['__abstractmethods__',
 '__class__',
 '__delattr__',
 '__dict__',
 '__dir__',
 '__doc__',
 '__eq__',
 '__format__',
 '__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',
 '_check_n_features',
 '_estimator_type',
 '_format_results',
 '_get_param_names',
 '_get_tags',
 '_more_tags',
 '_pairwise',
 '_repr_html_',
 '_repr_html_inner',
 '_repr_mimebundle_',
 '_required_parameters',
 '_run_search',
 '_validate_data',
 'best_estimator_',
 'best_index_',
 'best_params_',
 'best_score_',
```

```
 'classes_',
 'cv',
 'cv_results_',
 'decision_function',
 'error_score',
 'estimator',
 'fit',
 'get_params',
 'iid',
 'inverse_transform',
 'multimetric_',
 'n_features_in_',
 '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']
```

In [ ]:

```
# We have tested only 3 value of c, limited parametrs...Can be hard if we test c as a range
```

In [15]:

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

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[15]:

	param_C	param_kernel	mean_test_score
0	20	linear	0.966667
1	1	linear	0.980000

In [16]:

```
# Dictionary with classifiers and parametrs
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 [17]:

```
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[17]:

	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 [ ]:

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