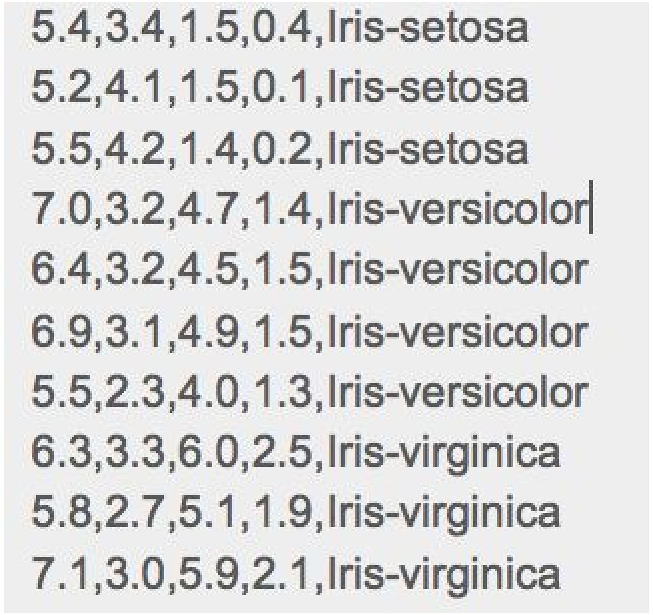
SVM

Study Case with Iris dataset

Dataset Link: https://archive.ics.uci.edu/ml/datasets/Iris

Number of Instances: 150 (50 in each of three classes)
Number of Attributes: 4 numeric, predictive attributes and the class
Attribute Information:

1. sepal length in cm
2. sepal width in cm
3. petal length in cm
4. petal width in cm
5. class:
   -- Iris Setosa
   -- Iris Versicolour
   -- Iris Virginica



```
5.4,3.4,1.5,0.4,Iris-setosa
5.2,4.1,1.5,0.1,Iris-setosa
5.5,4.2,1.4,0.2,Iris-setosa
7.0,3.2,4.7,1.4,Iris-versicolor
6.4,3.2,4.5,1.5,Iris-versicolor
6.9,3.1,4.9,1.5,Iris-versicolor
5.5,2.3,4.0,1.3,Iris-versicolor
6.3,3.3,6.0,2.5,Iris-virginica
5.8,2.7,5.1,1.9,Iris-virginica
7.1,3.0,5.9,2.1,Iris-virginica
```

Loading the iris sample dataset from sklearn

```
>>> import numpy as np
>>> from sklearn.model_selection import train_test_split
>>> from sklearn import datasets
>>> from sklearn import svm

>>> iris = datasets.load_iris()
>>> iris.data.shape, iris.target.shape
((150, 4), (150,))
```

We have total 150 data instances, each of which has 4 features + 1 class label
iris.data can be viewed as X, which is a 150*4 two-dimension array
iris.target can be viewed as y, which is a 150*1 array

For your own dataset, you need to read and store your data as 2 numpy ndarrays: X and y
https://docs.scipy.org/doc/numpy/reference/generated/numpy.ndarray.html

Cross Validation: Training Set and Testing Set

We can now quickly sample a training set while holding out 10% of the data for testing (evaluating) our classifier:

```
>>> X_train, X_test, y_train, y_test = train_test_split(
...     iris.data, iris.target, test_size=0.1, random_state=0)

>>> X_train.shape, y_train.shape
((135, 4), (135,))
>>> X_test.shape, y_test.shape
((15, 4), (15,))
```

15 data instances (X_test, y_test) are randomly sampled for testing

The rest 135 data instances (X_train, y_train) are used for training

Train the SVM Model

```
>>> clf = svm.SVC(kernel='linear', C=1).fit(X_train, y_train)
>>> clf.score(X_test, y_test)
0.96...
```

Here, we use the kernel=’linear’ and regularization parameter C= 1.
Note SVC is a class, and clf is an SVC object :

class sklearn.svm.**SVC**(*C=1.0*, *kernel='rbf'*, *degree=3*, *gamma='auto'*, *coef0=0.0*, *shrinking=True*, *probability=False*, *tol=0.001*, *cache_size=200*, *class_weight=None*, *verbose=False*, *max_iter=-1*, *decision_function_shape=None*, *random_state=None*)

You can also use other kernels such as
- ‘poly’,
- ‘rbf’,
- ‘sigmoid’,
- ‘precomputed’

and other parameter values.
Please refers to http://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html

Train the SVM model and Predict with Linear Kernel

```
>>> clf = svm.SVC(kernel='linear', C=1).fit(X_train, y_train)
>>> clf.score(X_test, y_test)
0.96...
```

We need to randomly separate the training set and testing set several times. The simplest way to use cross-validation is to call the **cross_val_score** helper function on the estimator and the dataset.

```
>>> from sklearn.model_selection import cross_val_score
>>> clf = svm.SVC(kernel='linear', C=1)
>>> scores = cross_val_score(clf, iris.data, iris.target, cv=10)
>>> scores
array([ 0.96...,  1.  ...,  0.96...,  0.96...,  1.    ...,0.96...,  1.  ...,  0.96...,  0.96...,  1.       ])
```

Here, it estimates the accuracy of a linear kernel support vector machine on the iris dataset by splitting the data, fitting a model and computing the score 10 consecutive times (with different splits each time):

Cross-Validation: Metric

The mean score and the 95% confidence interval of the score estimate are hence given by:

```
>>> print("Accuracy: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
Accuracy: 0.98 (+/- 0.03)
```

For more details:

http://scikit-learn.org/stable/modules/cross_validation.html

Parameter estimation using grid search with cross-validation

Cross validation iterators can also be used to directly perform model selection using Grid Search for the optimal hyperparameters of the model, such as kernel, C, and gamma.

```python
from sklearn.model_selection import GridSearchCV
# Set the parameters by cross-validation
tuned_parameters = [{'kernel': ['rbf'], 'gamma': [1e-3, 1e-4],'C': [1, 10, 100, 1000]},
                    {'kernel': ['linear'], 'C': [1, 10, 100, 1000]}]
scores = ['precision', 'recall']
svr = SVC(C=1)
for score in scores:
    print("# Tuning hyper-parameters for %s" % score)
    clf = GridSearchCV(svr, tuned_parameters, cv=10,scoring='%s_macro' % score)
    clf.fit(X_train, y_train)
    print("best parameters %s" % clf.best_params_)
    means = clf.cv_results_['mean_test_score']
    stds = clf.cv_results_['std_test_score']
    for mean, std, params in zip(means, stds, clf.cv_results_['params']):
        print("%0.3f (+/-%0.03f) for %r" % (mean, std * 2, params))

    y_true, y_pred = y_test, clf.predict(X_test)
```

Output

```
# Tuning hyper-parameters for precision
best parameters {'kernel': 'rbf', 'C': 100, 'gamma': 0.001}

0.479 (+/-0.038) for {'kernel': 'rbf', 'C': 1, 'gamma': 0.001}
0.149 (+/-0.209) for {'kernel': 'rbf', 'C': 1, 'gamma': 0.0001}
0.951 (+/-0.093) for {'kernel': 'rbf', 'C': 10, 'gamma': 0.001}
0.479 (+/-0.038) for {'kernel': 'rbf', 'C': 10, 'gamma': 0.0001}
0.971 (+/-0.076) for {'kernel': 'rbf', 'C': 100, 'gamma': 0.001}
0.951 (+/-0.093) for {'kernel': 'rbf', 'C': 100, 'gamma': 0.0001}
0.971 (+/-0.076) for {'kernel': 'rbf', 'C': 1000, 'gamma': 0.001}
0.971 (+/-0.076) for {'kernel': 'rbf', 'C': 1000, 'gamma': 0.0001}
0.967 (+/-0.089) for {'kernel': 'linear', 'C': 1}
0.971 (+/-0.076) for {'kernel': 'linear', 'C': 10}
0.959 (+/-0.075) for {'kernel': 'linear', 'C': 100}
0.965 (+/-0.077) for {'kernel': 'linear', 'C': 1000}
```

```
# Tuning hyper-parameters for recall
best parameters {'kernel': 'rbf', 'C': 100, 'gamma': 0.001}

0.667 (+/-0.000) for {'kernel': 'rbf', 'C': 1, 'gamma': 0.001}
0.365 (+/-0.197) for {'kernel': 'rbf', 'C': 1, 'gamma': 0.0001}
0.939 (+/-0.113) for {'kernel': 'rbf', 'C': 10, 'gamma': 0.001}
0.667 (+/-0.000) for {'kernel': 'rbf', 'C': 10, 'gamma': 0.0001}
0.963 (+/-0.104) for {'kernel': 'rbf', 'C': 100, 'gamma': 0.001}
0.939 (+/-0.113) for {'kernel': 'rbf', 'C': 100, 'gamma': 0.0001}
0.963 (+/-0.104) for {'kernel': 'rbf', 'C': 1000, 'gamma': 0.001}
0.963 (+/-0.104) for {'kernel': 'rbf', 'C': 1000, 'gamma': 0.0001}
0.957 (+/-0.119) for {'kernel': 'linear', 'C': 1}
0.963 (+/-0.104) for {'kernel': 'linear', 'C': 10}
0.948 (+/-0.102) for {'kernel': 'linear', 'C': 100}
0.955 (+/-0.104) for {'kernel': 'linear', 'C': 1000}
```

For details:
http://scikit-learn.org/stable/auto_examples/model_selection/grid_search_digits.html#sphx-glr-auto-examples-model-selection-grid-search-digits-py

Visualization: Linear Kernel

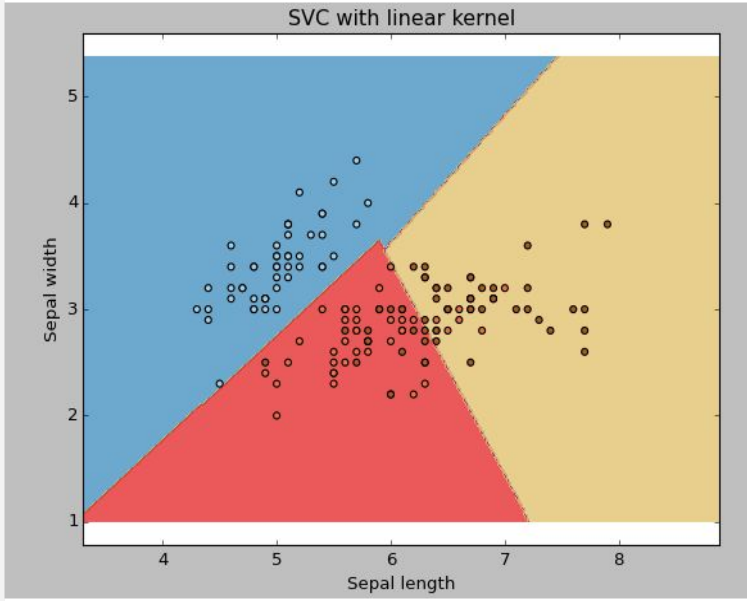
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn import svm, datasets
```

In order to conveniently visualize the classification, now we only use the first two features

```
iris = datasets.load_iris()

X = iris.data[:, :2]

y = iris.target
```

Visualization: linear kernel

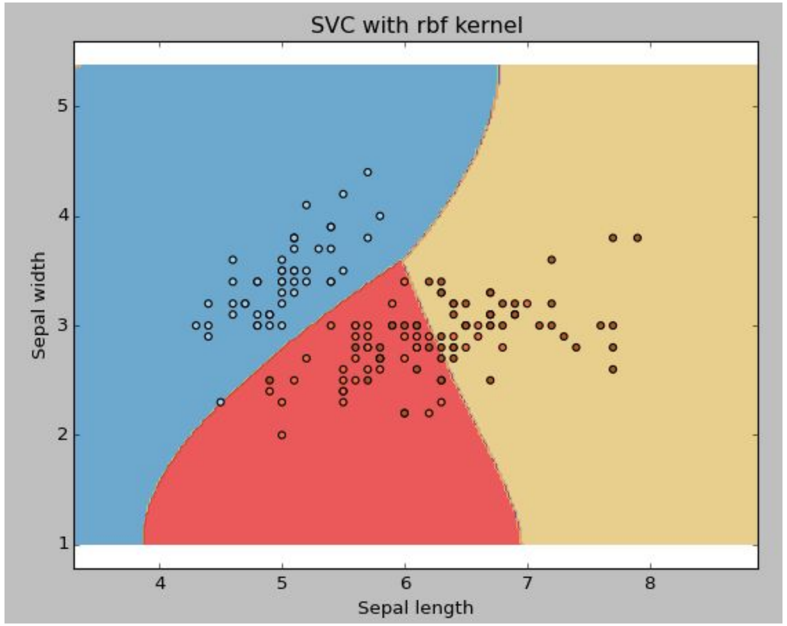
```
# we create an instance of SVM and fit out data.

C = 1.0 # SVM regularization parameter
svc = svm.SVC(kernel='linear', C=1,gamma=0).fit(X, y)

# create a mesh to plot in
x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
h = (x_max / x_min)/100
xx, yy = np.meshgrid(np.arange(x_min, x_max, h),np.arange(y_min, y_max,
h))

plt.subplot(1, 1, 1)
Z = svc.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)
plt.contourf(xx, yy, Z, cmap=plt.cm.Paired, alpha=0.8)

plt.scatter(X[:, 0], X[:, 1], c=y, cmap=plt.cm.Paired)
plt.xlabel('Sepal length')
plt.ylabel('Sepal width')
plt.xlim(xx.min(), xx.max())
plt.title('SVC with linear kernel')
plt.show()
```



Visualization: rbf Kernel

Change the kernel type to rbf in below line and look at the impact.

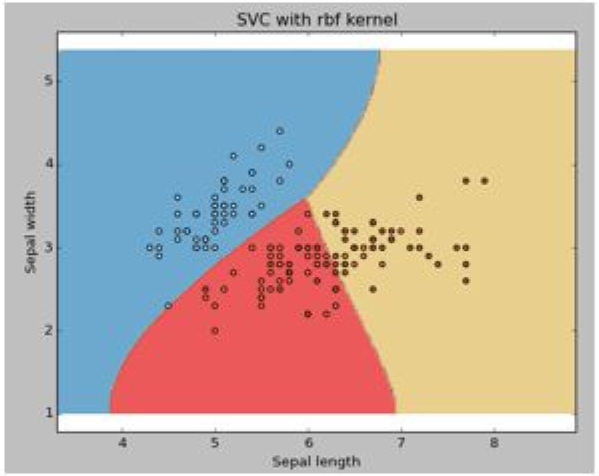
```
svc = svm.SVC(kernel='rbf', C=1,gamma=0).fit(X, y)
```



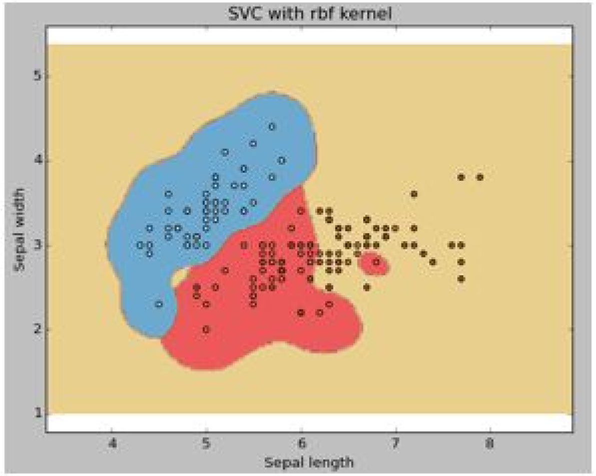
Visualization: Different gamma

gamma: Kernel coefficient for ‘rbf’, ‘poly’ and ‘sigmoid’. Higher the value of gamma, will try to exact fit the as per training data set i.e. generalization error and cause over-fitting problem.

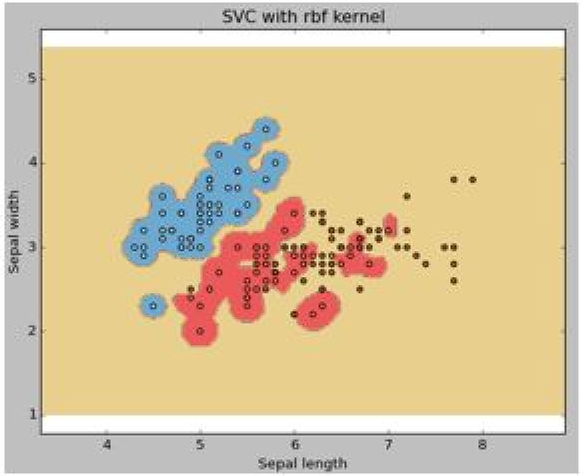
gamma =0



gamma =10



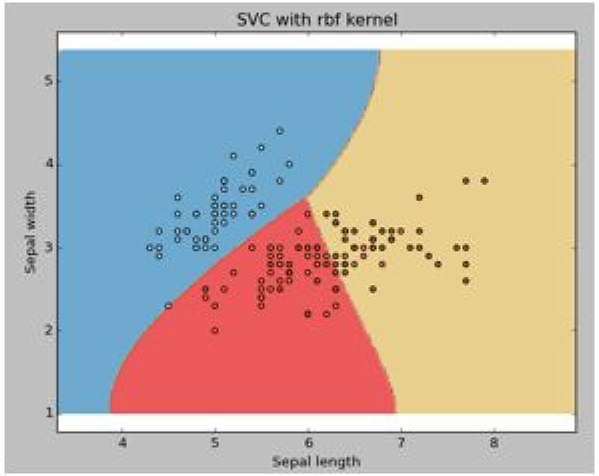
gamma =100



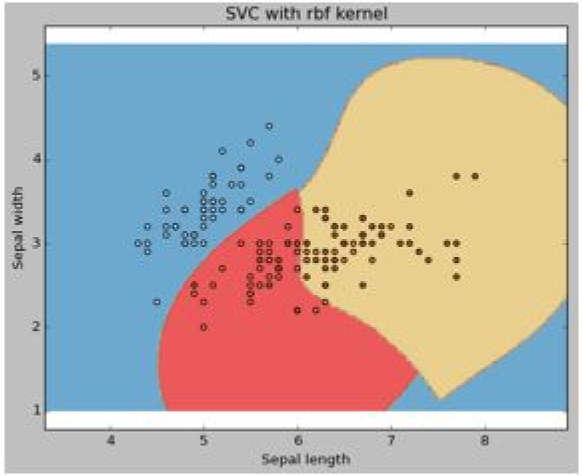
Visualization: Different C

C: Penalty parameter C of the error term. It also controls the trade off between smooth decision boundary and classifying the training points correctly.

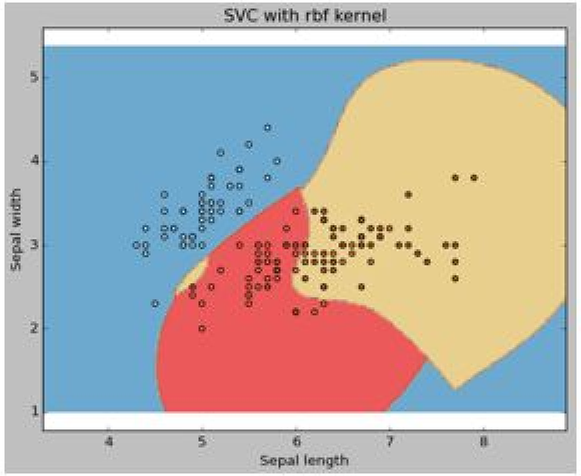
c =1



C =100



c =1000


SVC with rbf kernel
Sepal width
Sepal length