CSCI3230 (ESTR3108)
Fundamentals of Artificial Intelligence

Tutorial 3. Support Vector Machine

Wenao Ma

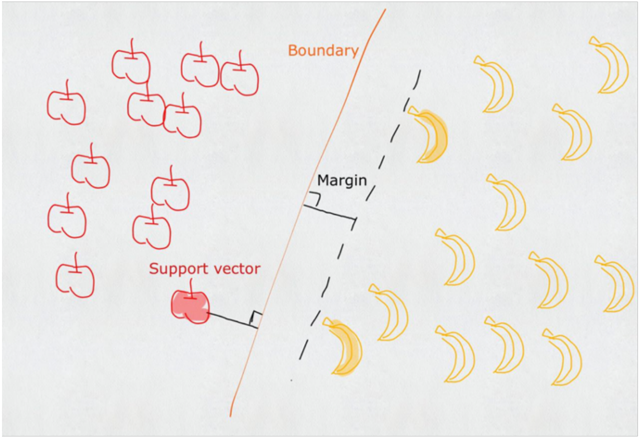
Email: wama@cse.cuhk.edu.hk
Office: Room 1024, 10/F, SHB

Dept. of Computer Science & Engineering
The Chinese University of Hong Kong

Concepts of SVM

What is SVM?

- One of the most theoretically well motivated and practically most effective linear classifier in machine learning.



Concepts of SVM

Still remember the derivation process of SVM?

$$L(\boldsymbol{w}, b, \boldsymbol{\alpha}) = \frac{1}{2}\|\boldsymbol{w}\|^2 + \sum_{i=1}^{m} \alpha_i \left(1 - y_i \left(\boldsymbol{w^T x_i} + b\right)\right)$$

$$\Downarrow$$

$$\max_{\boldsymbol{\alpha}} \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{m} \sum_{j=1}^{m} \alpha_i \alpha_j y_i y_j \boldsymbol{x_i^T x_j}$$

$$\text{s.t. } \sum_{i=1}^{m} \alpha_i y_i = 0, \alpha_i \geq 0, i = 1, 2, ..., m$$

$$\Downarrow$$

$$\begin{cases} \boldsymbol{w^*} = \sum_{i=1}^{m} \alpha_i y_i \boldsymbol{x_i} \\ b^* = \frac{1}{|S|} \sum_{s \in S} (\frac{1}{y_s} - \boldsymbol{w^T x_s}) \end{cases} \Rightarrow f(x) = \boldsymbol{w^T x} + b$$

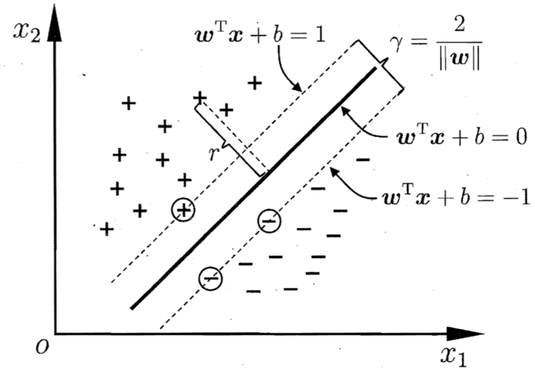
Concepts of SVM

Do you think these formulas are really hard to understand?

Concepts of SVM

Don't worry! You only need to know few things to understand today's tutorial.

- Recall that SVM is a kind of linear classifier: the algorithm creates a line or a hyperplane which separates the data into classes.
- We can use some optimization tool to get the optimal hyperplane given a set of data.



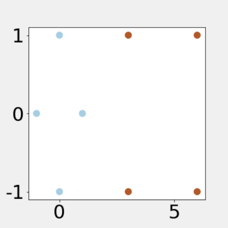
An example of SVM

- Assume that we have a set of two-dimensional vector samples, the positive data points are:

$$\left\{ \begin{pmatrix} 3 \\ 1 \end{pmatrix}, \begin{pmatrix} 3 \\ -1 \end{pmatrix}, \begin{pmatrix} 6 \\ 1 \end{pmatrix}, \begin{pmatrix} 6 \\ -1 \end{pmatrix} \right\}$$

  and the negative data points are:

$$\left\{ \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 1 \end{pmatrix}, \begin{pmatrix} -1 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ -1 \end{pmatrix} \right\}$$



An example of SVM

- Our objective is:

$$\max_{\boldsymbol{\alpha}} \sum_{j=1}^{8} \alpha_i - \frac{1}{2} \sum_{i=1}^{8} \sum_{j=1}^{8} \alpha_i \alpha_j y_i y_j \boldsymbol{x_i^T} \boldsymbol{x_j} \text{ s.t. } \sum_{i=1}^{8} \alpha_i y_i = 0, \alpha_i \geq 0$$

$$\Downarrow$$

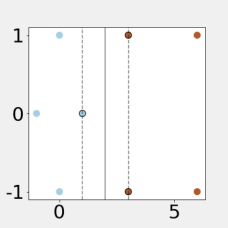
$$\max_\alpha \Big[\textstyle\sum_{j=1}^{8} \alpha_i - \frac{1}{2}\alpha_1^2 \cdot 1 \cdot 1 \left((3,1)^T (3,1) \right) - \frac{1}{2}\alpha_1\alpha_2 \cdot 1 \cdot 1 \cdot \left((3,1)^T \cdot (3,-1) \right) - \cdots - \frac{1}{2}\alpha_7\alpha_8 \cdot (-1) \cdot (-1) \cdot \left((0,-1)^T (-1,0) \right) - \frac{1}{2}\alpha_8^2 \cdot (-1) \cdot (-1) \left((-1,0)^T (-1,0) \right) \Big]$$

$$\text{s.t. } \alpha_1 + \alpha_2 + \alpha_3 + \alpha_4 - \alpha_5 - \alpha_6 - \alpha_7 - \alpha_8 = 0, \alpha_i \geq 0$$

- Use a SMO toolbox to solve this QP problem and get the optimal: $\alpha_1 = \alpha_2 = 0.25, \alpha_5 = 0.5, \alpha_3 = \alpha_4 = \alpha_6 = \alpha_7 = \alpha_8 = 0$
- Then, calculate the $\boldsymbol{w}^* = \sum_{i=1}^{8} \alpha_i y_i \boldsymbol{x_i} = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$

An example of SVM

- There are three support vectors:
$$\boldsymbol{x}_1 = \begin{pmatrix} 3 \\ 1 \end{pmatrix}, \boldsymbol{x}_2 = \begin{pmatrix} 3 \\ -1 \end{pmatrix}, \boldsymbol{x}_5 = \begin{pmatrix} 1 \\ 0 \end{pmatrix},$$
$$b^* = \frac{1}{|S|} \sum_{s \in S} \left(\frac{1}{y_s} - \boldsymbol{w}^{\boldsymbol{T}} \boldsymbol{x}_{\boldsymbol{s}}\right) = -2$$
- So we have: $\begin{cases} \boldsymbol{w} = \binom{1}{0} \\ b = -2 \end{cases}$
- We can get the hyperplane: $\binom{1}{0}\boldsymbol{x} - 2 = 0$



How to use SVM in Python

- We can implement SVM algorithm in Python by a library named **scikit-learn**
- Scikit-learn (also known as sklearn) is a free software machine learning library for the Python programming language
- Using this library, you just need a simple code to perform all the complex processes we have learned from the lecture (the derivation of SVM, finding the support vectors, SMO algorithm ...)

Implementation

- Let's see how to use scikit-learn to solve the question we have just mentioned.
- First let's import some packages:

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

Implementation

- Then, we write down our training set:

```
X = np.array([[3, 1], [3, -1],[6,1],[6,-1],[1,0],[0,1],[0,-1],[-1,0]])
y = [1,1,1,1,-1,-1,-1,-1]
```

- We next use a toolbox in scikit-learn to find the optimal hyperplane.

```
clf = svm.SVC(kernel='linear')
clf.fit(X, y)
```

Implementation

- Let's see the parameters of the hyperplane:

```
print("alpha:",clf.dual_coef_)
print("w:",clf.coef_)
print("support vectors: \n",clf.support_vectors_)
```

```
alpha: [[-0.50024083  0.25012042  0.25012042]]
w: [[ 1.00048166e+00 -1.66533454e-16]]
support vectors:
 [[ 1.  0.]
 [ 3.  1.]
 [ 3. -1.]]
```

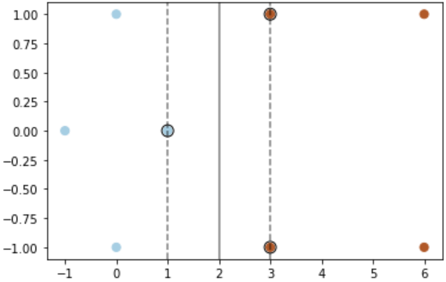
Implementation

- We can plot the figure via matplotlib (you can download this code after class):

```
def plot_svc_decision_function(model, ax=None, plot_support=True):
    """Plot the decision function for a 2D SVC"""
    if ax is None:
        ax = plt.gca()
    xlim = ax.get_xlim()
    ylim = ax.get_ylim()

    # create grid to evaluate model
    x = np.linspace(xlim[0], xlim[1], 30)
    y = np.linspace(ylim[0], ylim[1], 30)
    Y, X = np.meshgrid(y, x)
    xy = np.vstack([X.ravel(), Y.ravel()]).T
    P = model.decision_function(xy).reshape(X.shape)

    # plot decision boundary and margins
    ax.contour(X, Y, P, colors='k',
               levels=[-1, 0, 1], alpha=0.5,
               linestyles=['--', '-', '--'])

    # plot support vectors
    if plot_support:
        ax.scatter(model.support_vectors_[:, 0],
                   model.support_vectors_[:, 1],
                   s=300, linewidth=1, facecolors='none');
    ax.set_xlim(xlim)
    ax.set_ylim(ylim)
plt.scatter(X[:, 0], X[:, 1], c=y, s=50, cmap=plt.cm.Paired)
plot_svc_decision_function(clf)
```

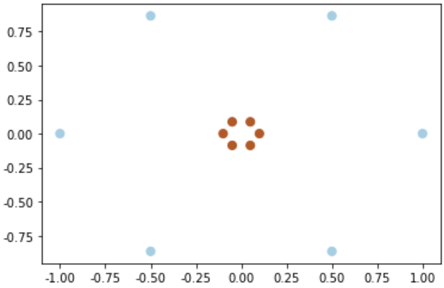
Implementation

- The output:



An example of kernel function

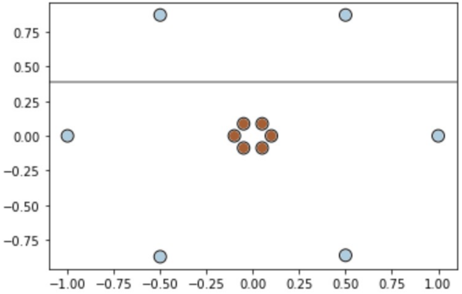
- Then let's consider a more complicated task. Assume we have 12 points as follow (brown denotes positive sample while blue denotes negative):



An example of kernel function

- As usual, we import the packages, input the training data and use the toolbox to find the hyperplane (but the result seems not good):

```
X = np.array([[-0.5, -0.87], [0.1, 0],[1,0],[0.5,-0.86],[-0.5,0.87],
              [-0.05,-0.087],[-0.1,0],[0.05,0.087],[0.05,-0.087],
              [-0.05,0.087],[0.5,0.87],[-1,0]])
y = [-1,1,-1,-1,-1,1,1,1,1,1,-1,-1]
clf = svm.SVC(kernel='linear')
clf.fit(X, y)
plt.scatter(X[:, 0], X[:, 1], c=y, s=50, cmap=plt.cm.Paired)
plot_svc_decision_function(clf, plot_support=True)
```



An example of kernel function

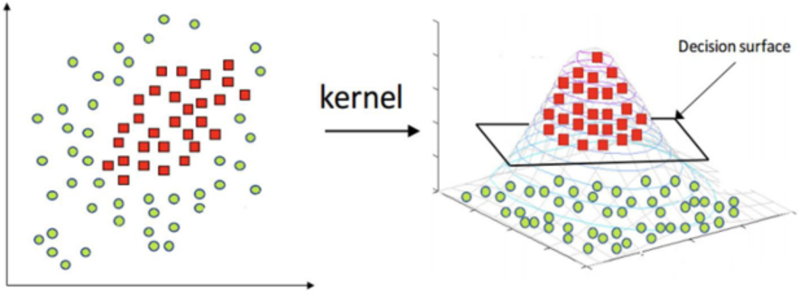
- How to handle this inseparable data?

An example of kernel function

- How to handle this inseparable data?
- Kernel function!

An example of kernel function

- 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.



An example of kernel function

- We use kernel function to alleviate the memory and computational cost.

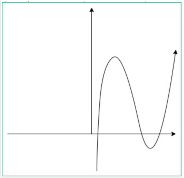
$$\max_{\boldsymbol{\alpha}} \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i=1}^{m} \sum_{j=1}^{m} \alpha_i \alpha_j y_i y_j \cancel{\boldsymbol{x_i^T x_j}}$$

$$\cancel{\phi(\boldsymbol{x_i})^T \phi(\boldsymbol{x_j})} \longrightarrow \boldsymbol{\kappa}(\boldsymbol{x_i}, \boldsymbol{x_j})$$

An example of kernel function

- Recall that we have seen some kernel function in the slide of lec 4:

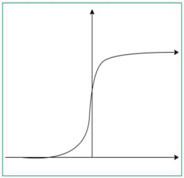
Polynomial Kernel :



$$\kappa(\boldsymbol{x_i}, \boldsymbol{x_j}) = (\boldsymbol{x_i^T x_j})^d$$

It represents the similarity of vectors in training set of data in a feature space over polynomials of the original variables used in kernel.

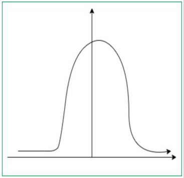
Sigmoid Kernel :



$$\kappa(\boldsymbol{x_i}, \boldsymbol{x_j}) = tahn(\beta \boldsymbol{x_i^T x_j} + \theta)$$

This function is equivalent to a two-layer, perceptron model of neural network, which is used as activation function for artificial neurons.

Gaussian Kernel :



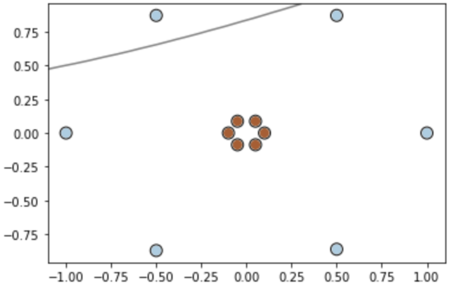
$$\kappa(\boldsymbol{x_i}, \boldsymbol{x_j}) = exp(-\frac{\|\boldsymbol{x_i} - \boldsymbol{x_j}\|^2}{2\sigma^2})$$

It is used to perform transformation, when there is no prior knowledge about data.

An example of kernel function

- In scikit-learn, we can implement kernel method by simply change one parameter:

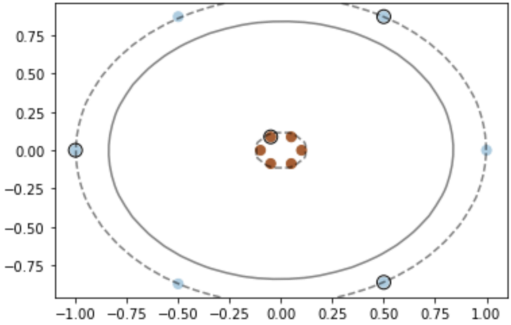
```
clf = svm.SVC(kernel='poly')
clf.fit(X, y)

plt.scatter(X[:, 0], X[:, 1], c=y, s=50, cmap=plt.cm.Paired)
plot_svc_decision_function(clf, plot_support=True)
```



An example of kernel function

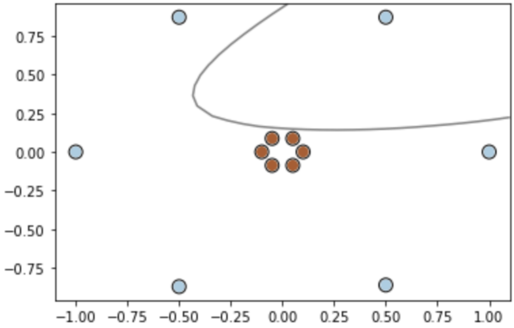
- Still not good? As for the polynomial Kernel, we can get the degree of it's function by just inputting an extra parameter to that function:

```
clf = svm.SVC(kernel='poly',degree = 4)
clf.fit(X, y)

plt.scatter(X[:, 0], X[:, 1], c=y, s=50, cmap=plt.cm.Paired)
plot_svc_decision_function(clf, plot_support=True)
```



- Good enough? Let's try some other kernel:

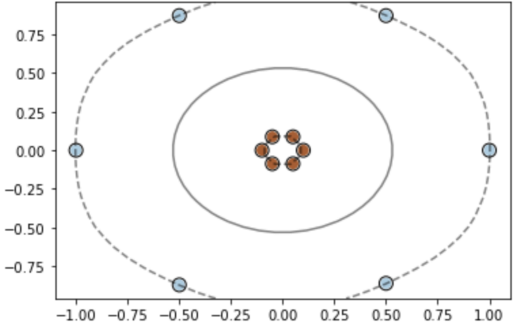
```
clf = svm.SVC(kernel='sigmoid')
clf.fit(X, y)

plt.scatter(X[:, 0], X[:, 1], c=y, s=50, cmap=plt.cm.Paired)
plot_svc_decision_function(clf, plot_support=True)
```



An example of kernel function

- Emmmmm. Sigmoid kernel seems not a good choice. Let's try Gaussian kernel:

```
clf = svm.SVC(kernel='rbf')
clf.fit(X, y)

plt.scatter(X[:, 0], X[:, 1], c=y, s=50, cmap=plt.cm.Paired)
plot_svc_decision_function(clf, plot_support=True)
```



An example of kernel function

- Then, let's try to evaluate our model on test data!

```
X_test = np.array([[-0.77, -0.51],
       [-0.037,  0.17],
       [ 0.14, -1.09],
       [-0.039,  0.14],
       [ 0.36,  0.84],
       [ 0.051,  0.19],
       [-0.10,  0.16,],
       [ 0.022,  0.12],
       [ 1.068, -0.16],
       [-0.64,  0.47]])
y_test = np.array([-1, 1, -1, 1, -1, 1, 1, 1, -1, -1])
```

An example of kernel function

- We can simply get the prediction by:

```
prediction = clf.predict(X_test)
print(prediction)
print(y_test)
```

```
[-1  1 -1  1 -1  1  1  1 -1 -1]
[-1  1 -1  1 -1  1  1  1 -1 -1]
```

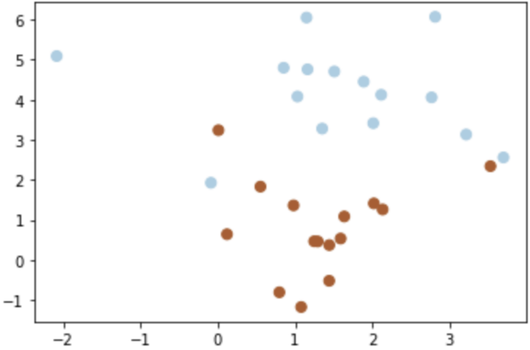
An example of soft-margin SVM

- But sometimes we are faced with more complicate situation. Let's create some data with noise:

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

def plot_svc_decision_function(model, ax=None, plot_support=True):
    """Plot the decision function for a 2D SVC"""
    if ax is None:
        ax = plt.gca()
    xlim = ax.get_xlim()
    ylim = ax.get_ylim()

    # create grid to evaluate model
    x = np.linspace(xlim[0], xlim[1], 30)
    y = np.linspace(ylim[0], ylim[1], 30)
    Y, X = np.meshgrid(y, x)
    xy = np.vstack([X.ravel(), Y.ravel()]).T
    P = model.decision_function(xy).reshape(X.shape)

    # plot decision boundary and margins
    ax.contour(X, Y, P, colors='k',
               levels=[-1, 0, 1], alpha=0.5,
               linestyles=['--', '-', '--'])

    # plot support vectors
    if plot_support:
        ax.scatter(model.support_vectors_[:, 0],
                   model.support_vectors_[:, 1],
                   s=100, linewidth=1, facecolors='none',edgecolors='k')
    ax.set_xlim(xlim)
    ax.set_ylim(ylim)


X, y = make_blobs(n_samples=30, centers=2,
                  random_state=0, cluster_std=1.2)
plt.scatter(X[:, 0], X[:, 1], c=y, s=50,cmap=plt.cm.Paired)
```

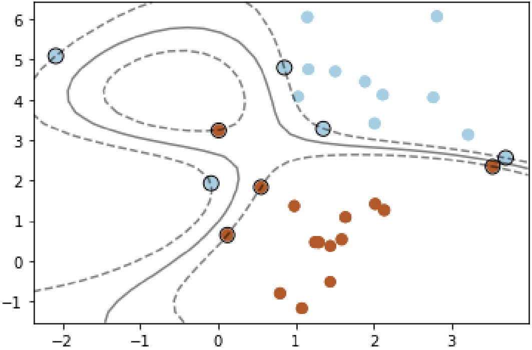
An example of soft-margin SVM

- Then we figure out the data:



An example of soft-margin SVM

- For this data set, even though we find a kernel function to make the data separable, it may be faced with overfitting problem.



- In this case, we can give up some noisy examples (Soft-margin SVM).

An example of soft-margin SVM

- Recall that in soft-margin SVM, our objective is:

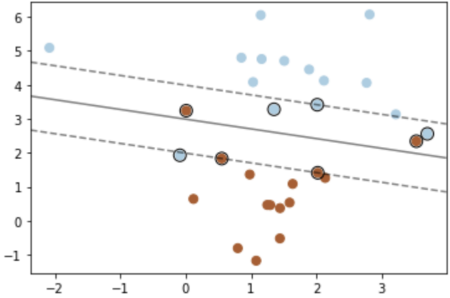
$$\min_{\boldsymbol{w},b} \frac{1}{2}\|\boldsymbol{w}\|^2 + C\sum_{i=1}^{m} l_{0/1}\left(y_i\left(\boldsymbol{w^T x_i} + b\right) - 1\right)$$

- We can adjust C to control the penalty item. Smaller C means we can give up more noisy samples.

An example of soft-margin SVM

- In scikit-learn, we can adjust the hyper-parameter "C" to achieve soft-margin SVM.
- The default of "C"" in scikit-learn is 1.0, which means this toolbox uses it by default:

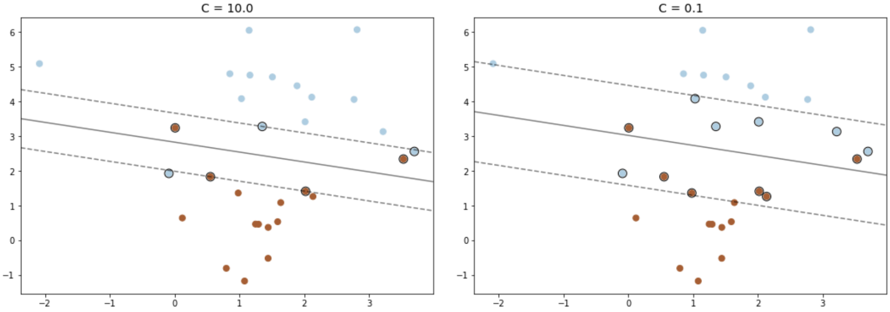
```
clf = svm.SVC(kernel='linear')
clf.fit(X, y)
plt.scatter(X[:, 0], X[:, 1], c=y, s=50, cmap=plt.cm.Paired)
plot_svc_decision_function(clf, plot_support=True)
```



An example of soft-margin SVM

- Then we can adjust "C"" to see how it works:

```
fig, ax = plt.subplots(1, 2, figsize=(16, 6))
fig.subplots_adjust(left=0.0625, right=0.95, wspace=0.1)

for axi, C in zip(ax, [10.0, 0.1]):
    model = svm.SVC(kernel='linear', C=C).fit(X, y)
    axi.scatter(X[:, 0], X[:, 1], c=y, s=50, cmap=plt.cm.Paired)
    plot_svc_decision_function(model, axi)
    axi.scatter(model.support_vectors_[:, 0],
                model.support_vectors_[:, 1],
                s=300, lw=1, facecolors='none');
    axi.set_title('C = {0:.1f}'.format(C), size=14)
```



Summary

- You can also try to change other parameters in sklearn.svm.SVC (don't forget to read its API documentation).
- And don't forget to use test data to evaluate your model.
- Good luck!

sklearn.svm.SVC ¶

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