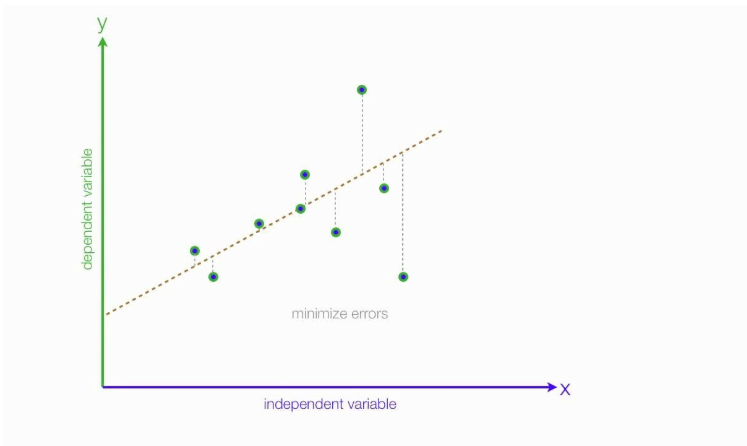
# 第23课：SVR——一种“宽容”的回归模型

**线性回归**：在向量空间里用线性函数去拟合样本。

该模型以所有样本实际位置到该线性函数的综合距离为损失，通过最小化损失来求取线性函数的参数。参见下图：



对于线性回归而言，一个样本只要不是正好落在最终作为模型的线性函数上，就要被计算损失。

## 宽容的支持向量回归（SVR）

### 模型函数

支持向量回归模型的**模型函数**也是一个线性函数： y = wx + b。

看起来和线性回归的模型函数一样！

但 **SVR 和线性回归**，却是**两个不同的回归模型**。

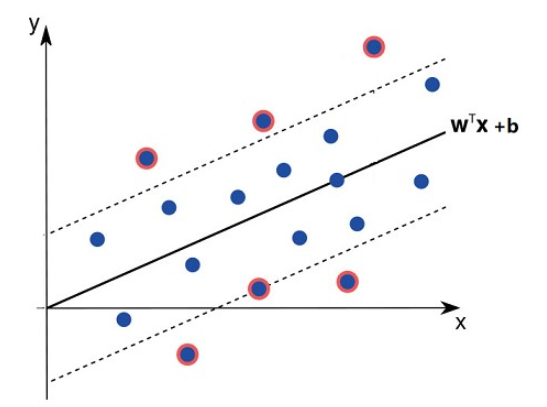
不同在哪儿呢？不同在学习过程。

说得更详细点，就是：**计算损失的原则不同，目标函数和最优化算法也不同**。

### 原理

**SVR** 在线性函数两侧制造了一个“间隔带”，对于所有落入到间隔带内的样本，都不计算损失；只有间隔带之外的，才计入损失函数。之后再通过最小化间隔带的宽度与总损失来最优化模型。

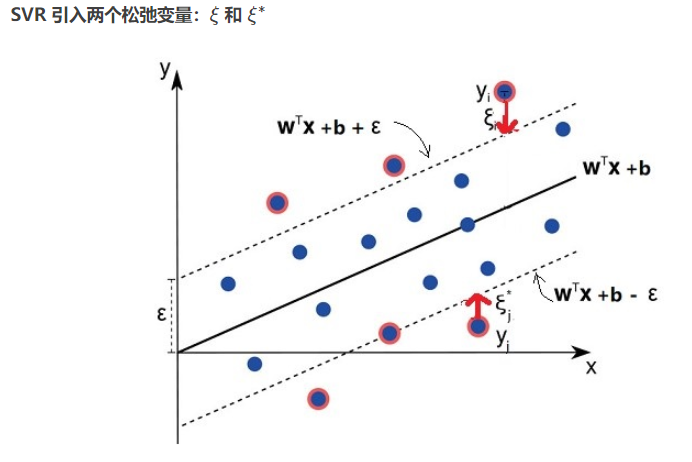
如下图这样，只有那些圈了红圈的样本（或在隔离带边缘之外，或落在隔离带边缘上），才被计入最后的损失：

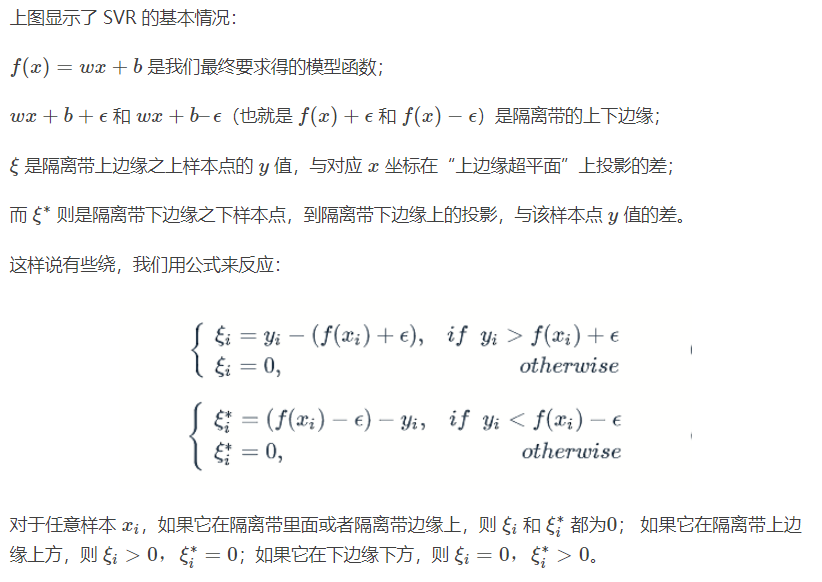


### SVR 的两个松弛变量

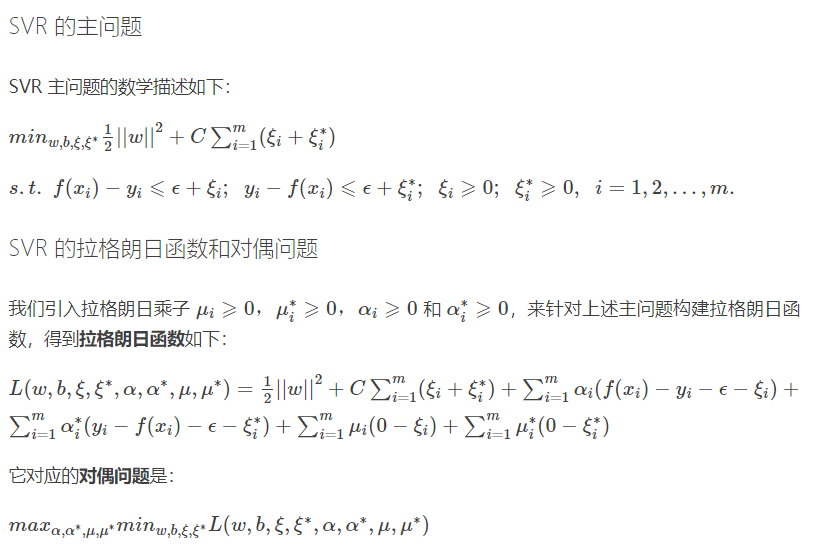
有一点 SVR 和 SVM **正相反**，那就是：SVR 巴不得所有的样本点都落在“隔离带”里面，而 SVM 则恰恰希望所有的样本点都在“隔离带”之外！

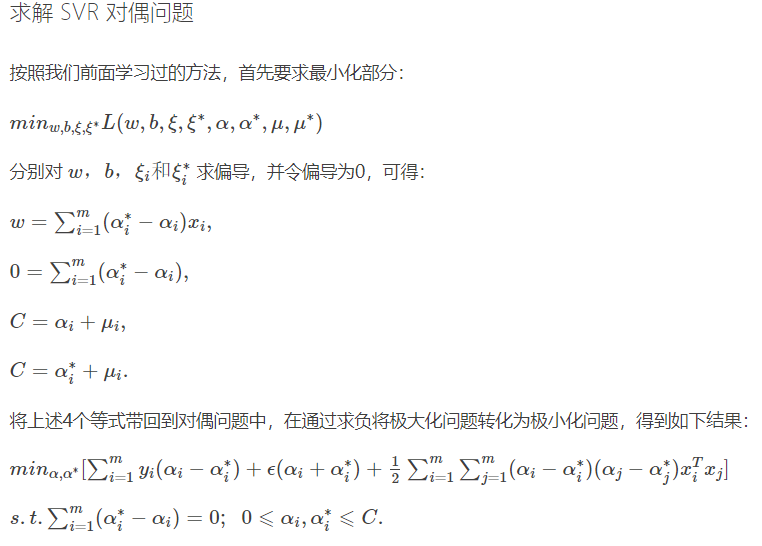
正是这一点区别，导致 SVR 要同时引入两个而不是一个松弛变量。

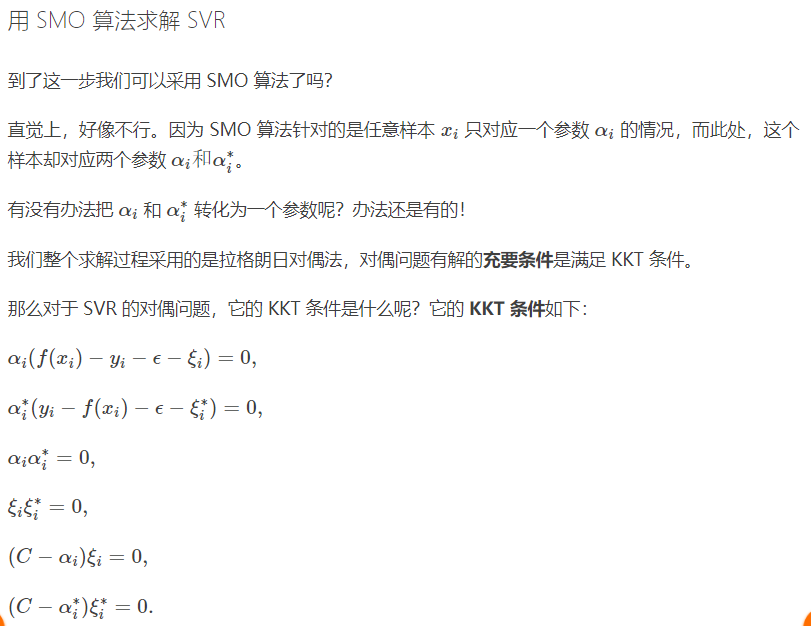


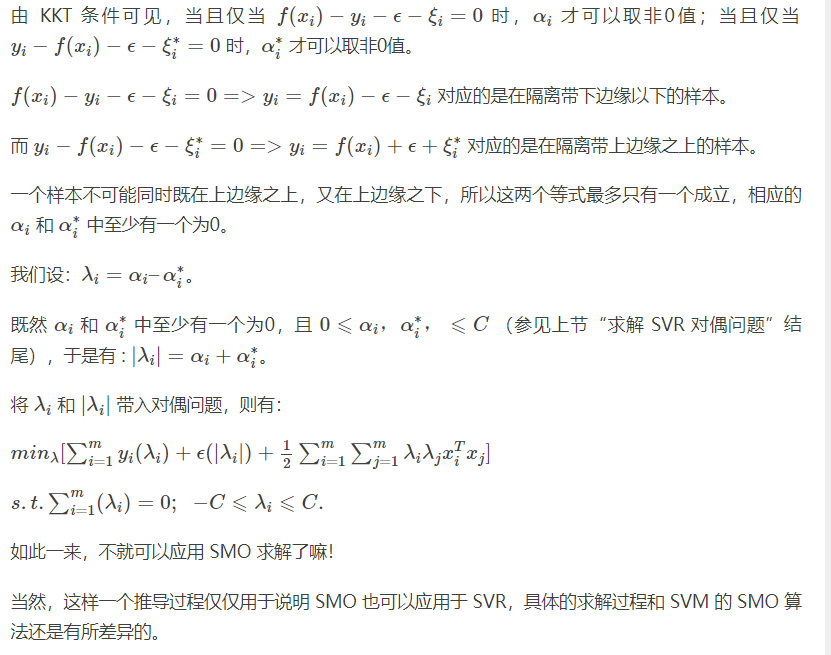


## SVR 的主问题和对偶问题

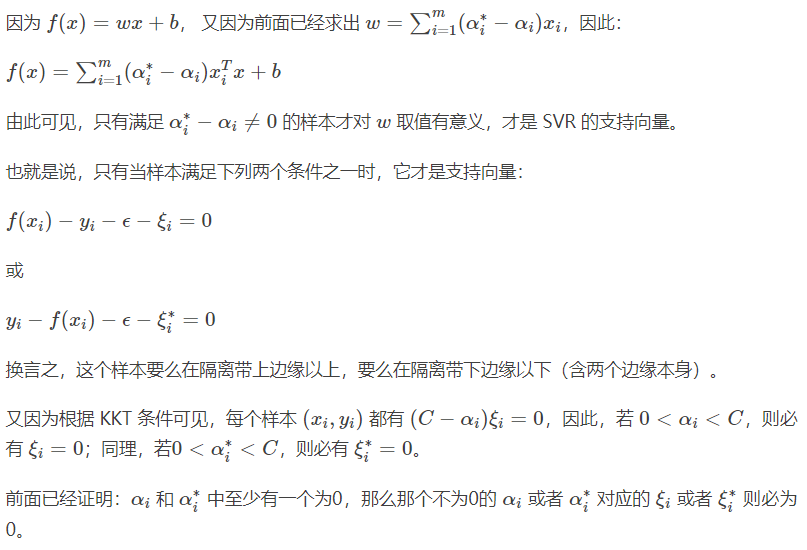


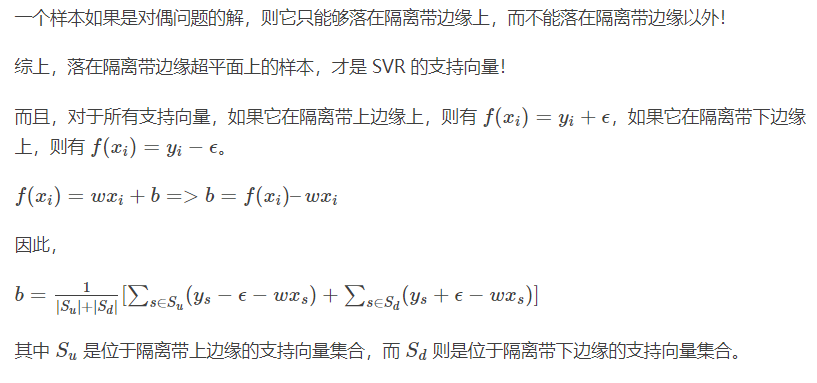




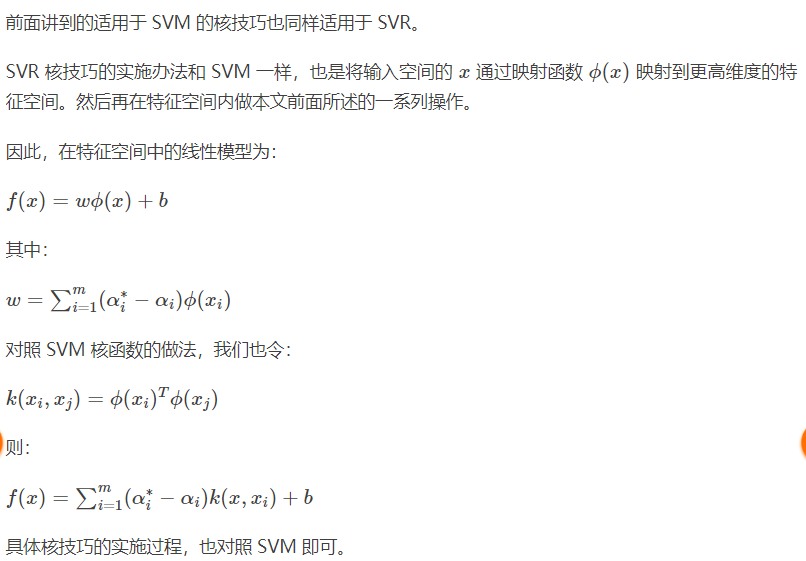


## 支持向量与求解线性模型参数





## SVR 的核技巧



# 第24课：直观认识 SVM 和 SVR

## SVM 实例

### 线性可分 SVM

import numpy as np

import matplotlib.pyplot as plt

from sklearn.svm import SVC # "Support vector classifier"

# 定义函数plot\_svc\_decision\_function用于绘制分割超平面和其两侧的辅助超平面

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()

# 创建网格用于评价模型

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)

#绘制超平面

ax.contour(X, Y, P, colors='k',

levels=[-1, 0, 1], alpha=0.5,

linestyles=['--', '-', '--'])

#标识出支持向量

if plot\_support:

ax.scatter(model.support\_vectors\_[:, 0],

model.support\_vectors\_[:, 1],

s=300, linewidth=1, edgecolors='blue', facecolors='none');

ax.set\_xlim(xlim)

ax.set\_ylim(ylim)

# 用make\_blobs生成样本数据

from sklearn.datasets.samples\_generator import make\_blobs

X, y = make\_blobs(n\_samples=50, centers=2,

random\_state=0, cluster\_std=0.60)

# 将样本数据绘制在直角坐标中

plt.scatter(X[:, 0], X[:, 1], c=y, s=50, cmap='autumn');

plt.show()

# 用线性核函数的SVM来对样本进行分类

model = SVC(kernel='linear')

model.fit(X, y)

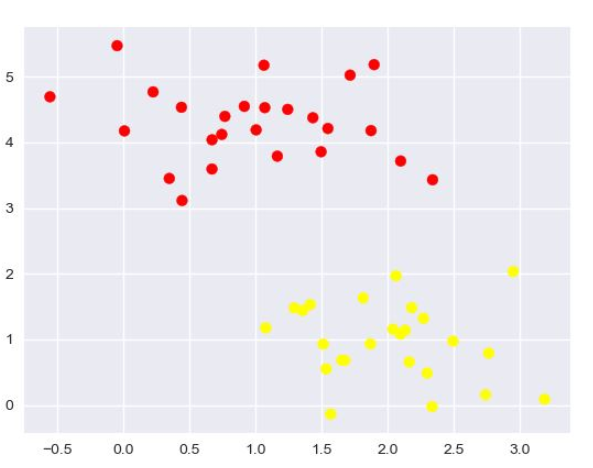
# 在直角坐标中绘制出分割超平面、辅助超平面和支持向量

plt.scatter(X[:, 0], X[:, 1], c=y, s=50, cmap='autumn')

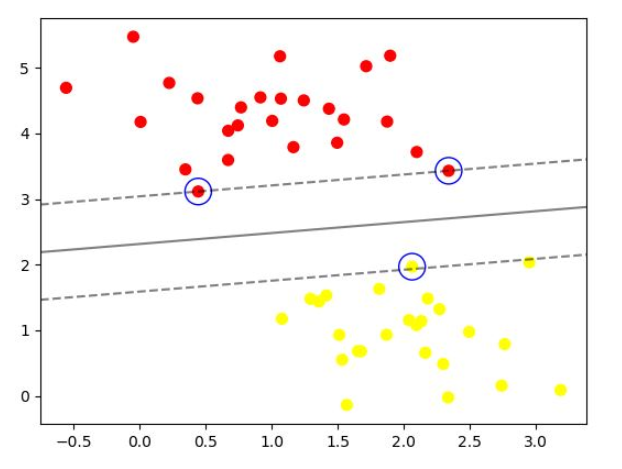
plot\_svc\_decision\_function(model);

plt.show()

用上面的代码生成的数据显示如下：



用线性核 SVM 分类后，显示如下：



我们可以看到，因为本来就是线性可分的两类样本，因此最大分割超平面和相应的辅助超平面都很清晰，相应的支持变量也正好落在两类样本的边缘处。

### 线性 SVM

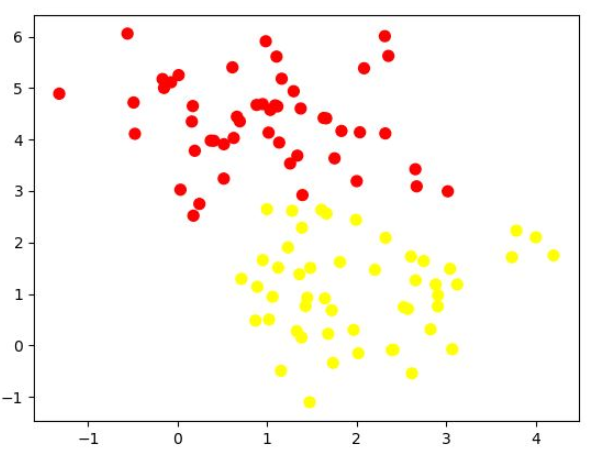
X, y = make\_blobs(n\_samples=100, centers=2,

random\_state=0, cluster\_std=0.9)

plt.scatter(X[:, 0], X[:, 1], c=y, s=50, cmap='autumn');

plt.show()

这段代码生成的数据显示如下：



我们再用同样的方法去分割它们：

model = SVC(kernel='linear')

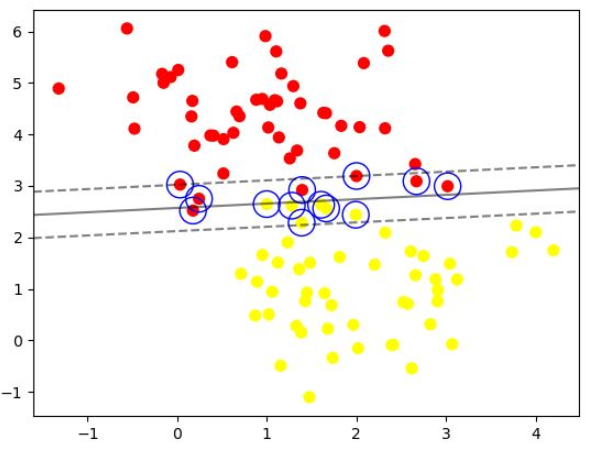
model.fit(X, y)

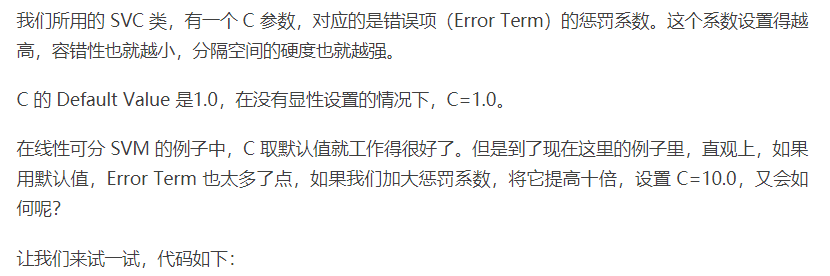
plt.scatter(X[:, 0], X[:, 1], c=y, s=50, cmap='autumn')

plot\_svc\_decision\_function(model)

plt.show()

结果是这样的：





model = SVC(kernel='linear', C=10.0)

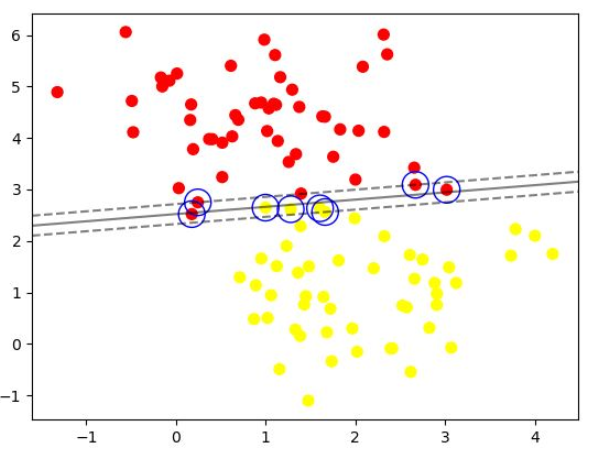
model.fit(X, y)

plt.scatter(X[:, 0], X[:, 1], c=y, s=50, cmap='autumn')

plot\_svc\_decision\_function(model)

plt.show()

生成图为：



### 完全线性不可分的数据

生成数据：

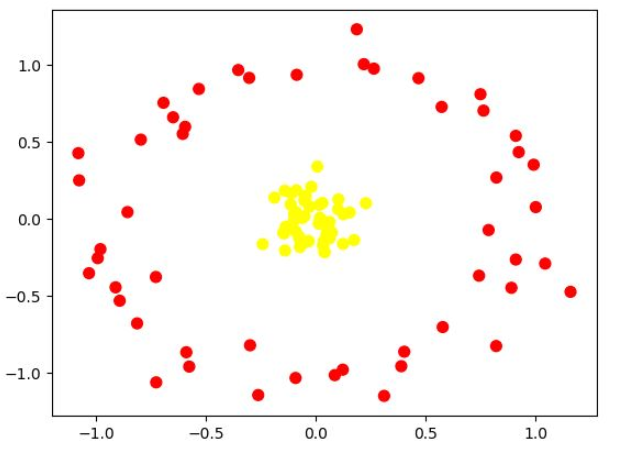
**from** sklearn.datasets.samples\_generator **import** make\_circles

X, y = make\_circles(100, factor=.1, noise=.1)

plt.scatter(X[:, 0], X[:, 1], c=y, s=50, cmap='autumn');

plt.show()

效果图：



如果用二维空间中线性分隔他们：

model = SVC(kernel='linear')

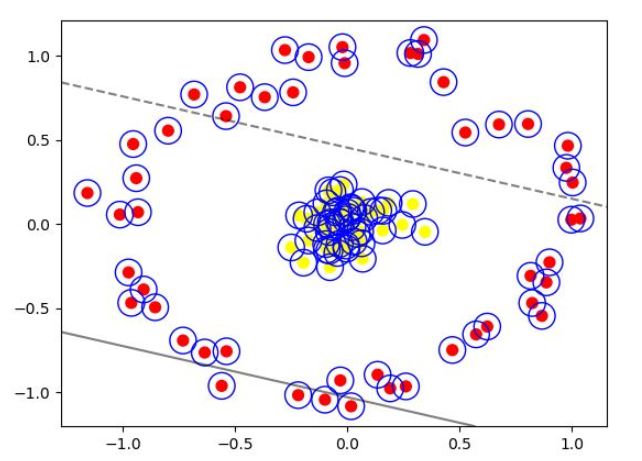
model.fit(X, y)

plt.scatter(X[:, 0], X[:, 1], c=y, s=50, cmap='autumn')

plot\_svc\_decision\_function(model);

plt.show()

结果：



### 核函数的作用

上面的数据，用投射到三维后：

**from** mpl\_toolkits **import** mplot3d

**def** **plot\_3D**(elev=30, azim=30, X=None, y=None):

ax = plt.subplot(projection='3d')

r = np.exp(-(X \*\* 2).sum(1))

ax.scatter3D(X[:, 0], X[:, 1], r, c=y, s=50, cmap='autumn')

ax.view\_init(elev=elev, azim=azim)

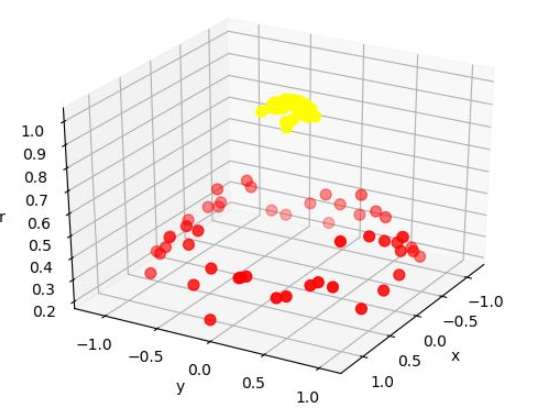
ax.set\_xlabel('x')

ax.set\_ylabel('y')

ax.set\_zlabel('r')

plot\_3D(X=X, y=y)

plt.show()



正负例样本在三维空间被分为了两簇。从直观角度看，如果我们在两簇中间的位置“横切一刀”，完全有可能将它们分开。

如何将这些样本在更高维度空间的投射体现在代码中呢？我们需要重新生成二维的 X 再用 SVC fit 吗？

其实不用那么麻烦，我们可以直接采用 **RBF 核**！

只要将上面的代码修改一个参数，改为：

model = SVC(kernel='rbf')

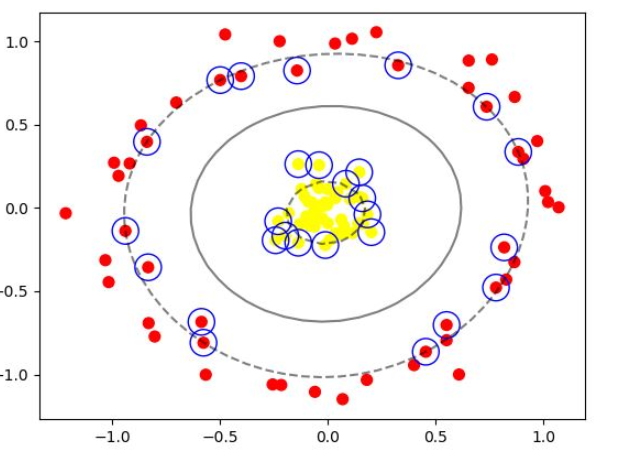
model.fit(X, y)

plt.scatter(X[:, 0], X[:, 1], c=y, s=50, cmap='autumn')

plot\_svc\_decision\_function(model);

plt.show()

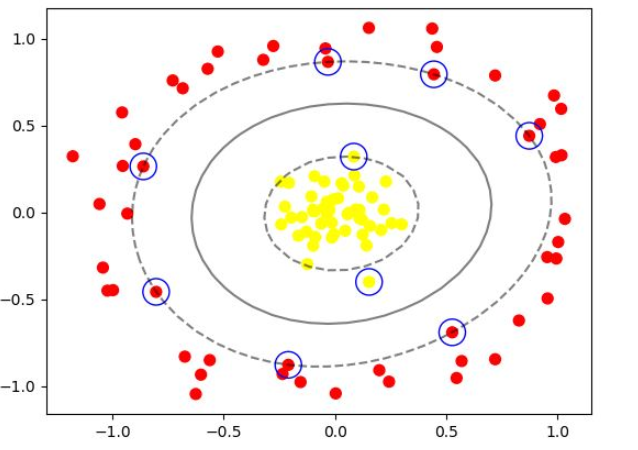
结果：



调高一些惩罚系数：

model = SVC(kernel='rbf', C=10)

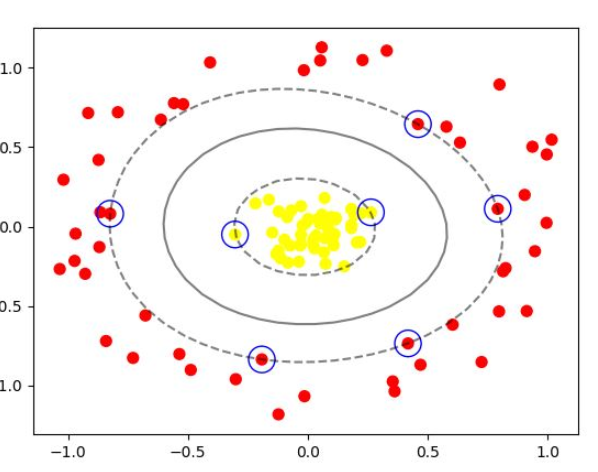
结果：



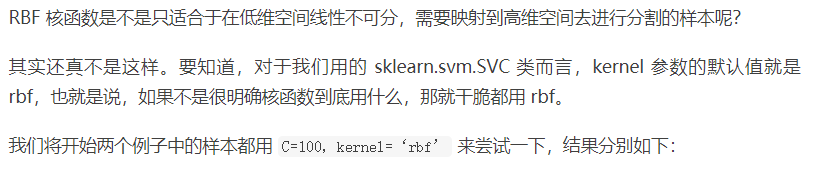
把惩罚系数调高10倍

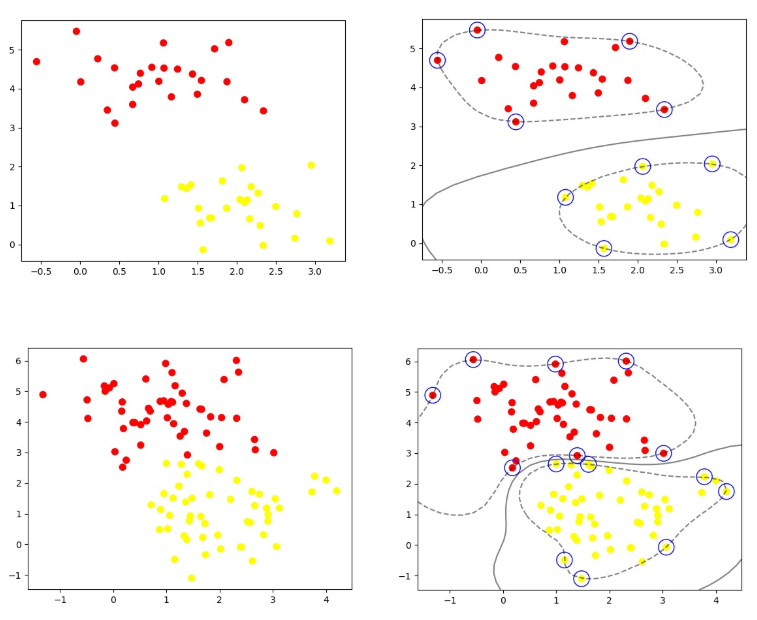
model = SVC(kernel='rbf', C=100)

结果：

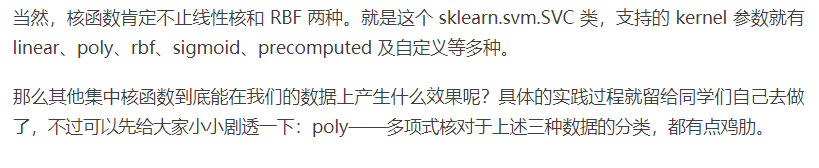


### RBF 核函数的威力





### 其他核函数



# SVR 实例

**import** numpy **as** np

**from** sklearn.svm **import** SVR

**import** matplotlib.pyplot **as** plt

*# 生成样本数据*

X = np.sort(5 \* np.random.rand(40, 1), axis=0)

y = np.ravel(2\*X + 3)

*# 加入部分噪音*

y[::5] += 3 \* (0.5 - np.random.rand(8))

*# 调用模型*

svr\_rbf = SVR(kernel='rbf', C=1e3, gamma=0.1)

svr\_lin = SVR(kernel='linear', C=1e3)

svr\_poly = SVR(kernel='poly', C=1e3, degree=2)

y\_rbf = svr\_rbf.fit(X, y).predict(X)

y\_lin = svr\_lin.fit(X, y).predict(X)

y\_poly = svr\_poly.fit(X, y).predict(X)

*# 可视化结果*

lw = 2

plt.scatter(X, y, color='darkorange', label='data')

plt.plot(X, y\_rbf, color='navy', lw=lw, label='RBF model')

plt.plot(X, y\_lin, color='c', lw=lw, label='Linear model')

plt.plot(X, y\_poly, color='cornflowerblue', lw=lw, label='Polynomial model')

plt.xlabel('data')

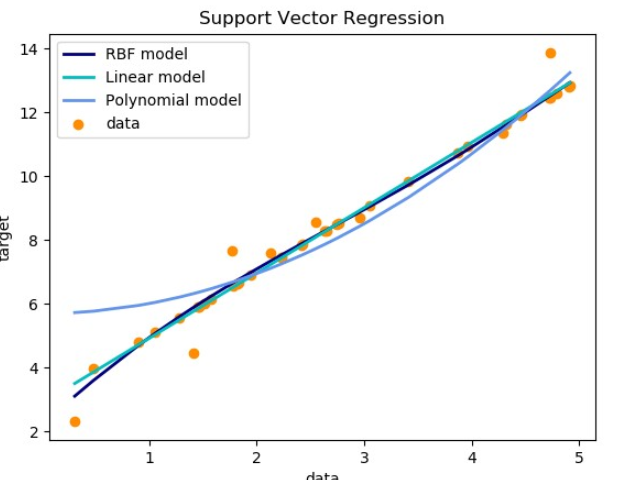
plt.ylabel('target')

plt.title('Support Vector Regression')

plt.legend()

plt.show()

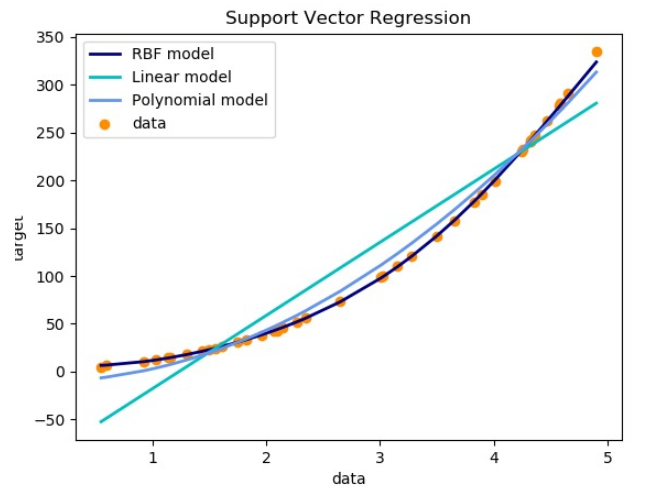
结果：



将 y = np.ravel(2\*X + 3) 替换为：

y = np.polyval([2,3,5,2], X).ravel()

结果为：



如果替换为：

y = np.**sin**(X).ravel()

结果：

