AI学习笔记--sklearn--SVM算法

- SVM算法解释

Svm 算法又叫支持向量机（Support Vector Machine, SVM）是一类按监督学习（supervised learning）方式对数据进行二元分类（binary classification）的广义线性分类器（generalized linear classifier），其决策边界是对学习样本求解的最大边距超平面（maximum-margin hyperplane）[1-3]。

SVM使用铰链损失函数（hinge loss）计算经验风险（empirical risk）并在求解系统中加入了正则化项以优化结构风险（structural risk），是一个具有稀疏性和稳健性的分类器[2]。SVM可以通过核方法（kernel method）进行非线性分类，是常见的核学习（kernel learning）方法之一[4]。

SVM被提出于1963年，在二十世纪90年代后得到快速发展并衍生出一系列改进和扩展算法，包括多分类SVM[5]、最小二乘SVM（Least-Square SVM, LS-SVM）[6]、支持向量回归（Support Vector Regression, SVR）[2]、支持向量聚类（support vector clustering）[7]、半监督SVM（semi-supervised SVM, S3VM）[8]等，在人像识别（face recognition）、文本分类（text categorization）等模式识别（pattern recognition）问题中有广泛应用[9-10]。

- SVM算法理论

线性分类

- 线性可分性（linear separability）

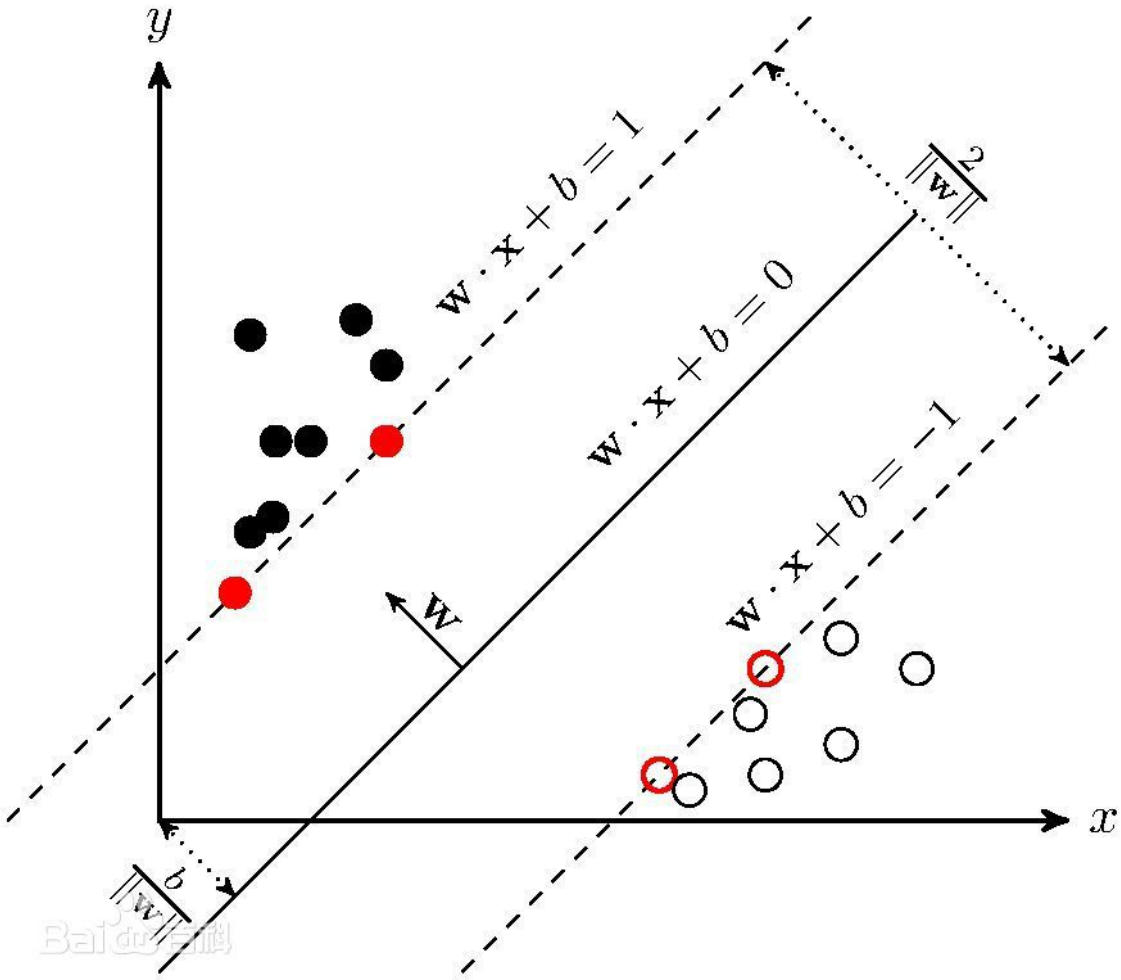
在分类问题中给定输入数据和学习目标 $\boldsymbol{X}=\{\boldsymbol{X}_1,\ldots,\boldsymbol{X}_N\},\boldsymbol{y}=\{y_1,\ldots,y_N\}$，其中输入数据的每个样本都包含多个特征并由此构成特征空间（feature space）：$\boldsymbol{X}_i=[x_1,\ldots,x_n]\in\chi$，而学习目标为二元变量 $\boldsymbol{y}\in\{-1,1\}$ 表示负类（negative class）和正类（positive class）。若输入数据所在的特征空间存在作为决策边界（decision boundary）的超平面（hyperplane）：$\boldsymbol{w}^{\mathrm{T}}\boldsymbol{X}+b=0$ 将学习目标按正类和负类分开，并使任意样本的点到平面距离大于等于1[2]：$y_i\left(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{X}_i+b\right)\geq 1$ 则称该分类问题具有线性可分性，参数 $\boldsymbol{w},b$ 分别为超平面的法向量和截距。

满足该条件的决策边界实际上构造了2个平行的超平面：$\boldsymbol{w}^{\mathrm{T}}\boldsymbol{X}+b=\pm 1$ 作为间隔边界以判别样本的分类：

$$\begin{aligned}\boldsymbol{w}^{\mathrm{T}}\boldsymbol{X}_i+b-1\geq+1,\quad \text{if}\quad y_i=+1\\ \boldsymbol{w}^{\mathrm{T}}\boldsymbol{X}_i+b+1\leq-1,\quad \text{if}\quad y_i=-1\end{aligned}$$

所有在上间隔边界上方的样本属于正类，在下间隔边界下方的样本属于负类。两个间隔边界的距离 $d=\dfrac{2}{\|\boldsymbol{w}\|}$ 被定义为边距（margin），位于间隔边界上的正类和负类样本为支持向量

（support vector）。



- 损失函数（**loss function**）

在一个分类问题不具有线性可分性时，使用超平面作为决策边界会带来分类损失，即部分支持向量不再位于间隔边界上，而是进入了间隔边界内部，或落入决策边界的错误一侧。损失函数可以对分类损失进行量化，其按数学意义可以得到的形式是0-1损失函数：

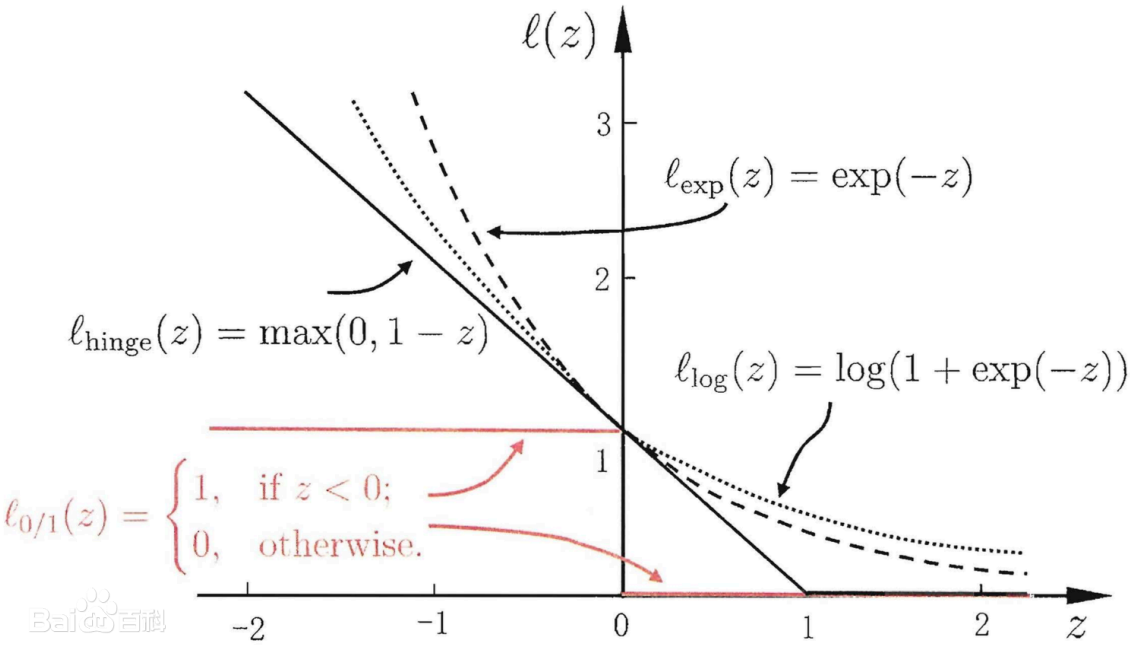
$$L(p)=\begin{cases}0 & p<0\\ 1 & p\geq 0\end{cases}$$

0-1损失函数不是连续函数，不利于优化问题的求解，通常的选择是构造代理损失（surrogate loss）。可用的选择包括铰链损失函数（hinge loss）、logistic损失函数（logistic loss）、和指数损失函数（exponential loss），对应的表达式如下[2]

$$\begin{aligned}&\text{hinge:} && L(p)=\max(0,1-p)\\ &\text{logistic:} && L(p)=\log[1+\exp(-p)]\\ &\text{exponential:} && L(p)=\exp(-p)\end{aligned}$$

其中SVM使用的是铰链损失函数[2]。对替代损失的相合性研究表明，当代理损失是连续凸函数，并在任意取值下是0-1损失函数的上界，则求解代理损失最小化所得结果也是0-1损失最小

化的解[2] [19]。



- 经验风险（**empirical risk**）与正则化（**regularizatgion**）

通过损失函数可以定义经验风险：

$$\epsilon = \sum_{i=1}^{N} L(p_i) = \sum_{i=1}^{N} L[f(\boldsymbol{X}_i, \boldsymbol{w}), y_i]$$

式中的 f 表示分类器，其复杂程度可以定义结构风险（structural risk）：$\Omega(f) = \|\boldsymbol{w}\|^p$。经验风险描述了分类器所给出的分类结果的准确程度；结构风险描述了分类器自身的稳定程度，复杂的分类器容易产生过拟合，因此是不稳定的。若一个分类器通过最小化经验风险和结构风险的线性组合以确定其模型参数：

$$\min_f \|\boldsymbol{w}\|^p + C \sum_{i=1}^{N} L[f(\boldsymbol{X}_i, \boldsymbol{w}), y_i]$$

则对该分类器的求解是一个正则化问题，常数C是正则化系数。常见地，$p = 2$时，该式被称为L_2正则化或Tikhonov正则化（Tikhonov regularization）[20]。硬边界SVM是一个完全最小化结构风险的分类器，软边界SVM是一个L_2正则化分类器，同时最小化结构风险和经验风险。

核方法

一些线性不可分的问题可能是非线性可分的，即特征空间存在超曲面（hypersurface）将正类和负类分开。使用非线性函数可以将非线性可分问题从原始的特征空间映射至更高维的希尔伯特空间（Hilbert space）$\mathcal{H}$，从而转化为线性可分问题，此时作为决策边界的超平面表示如下[2] [3]

$$\boldsymbol{w}^{\mathrm{T}} \phi(\boldsymbol{X}) + b = 0$$

式中 $\phi : \chi \mapsto \mathcal{H}$ 为映射函数。由于映射函数是复杂的非线性函数，因此其内积的计算是困难的，此时可使用核方法（kernel method），即定义映射函数的内积为核函数（kernel function）：$\kappa(\boldsymbol{X}_1, \boldsymbol{X}_2) = \phi(\boldsymbol{X}_1)^{\mathrm{T}} \phi(\boldsymbol{X}_2)$ 以回避内积的显式计算 [2] [3]

- **Mercer定理（Mercer's theorem）**

核函数的选择需要一定条件，函数 $\kappa(\boldsymbol{X}_1, \boldsymbol{X}_2) : \chi \times \chi \mapsto \mathbb{R}$ 是核函数的充要条件是，对输入空间的任意向量：$\{\boldsymbol{X}_1, \ldots, \boldsymbol{X}_m\} \in \chi$，其核矩阵（kernel matrix），即如下形式的格拉姆矩阵（Gram matrix）：

$$G(\boldsymbol{X}, \boldsymbol{X}) = \begin{bmatrix} \kappa(\boldsymbol{X}_1, \boldsymbol{X}_1) & \kappa(\boldsymbol{X}_1, \boldsymbol{X}_2) & \cdots & \kappa(\boldsymbol{X}_1, \boldsymbol{X}_m) \\ \kappa(\boldsymbol{X}_2, \boldsymbol{X}_1) & \kappa(\boldsymbol{X}_2, \boldsymbol{X}_2) & \cdots & \kappa(\boldsymbol{X}_2, \boldsymbol{X}_m) \\ \vdots & \vdots & \ddots & \vdots \\ \kappa(\boldsymbol{X}_m, \boldsymbol{X}_1) & \kappa(\boldsymbol{X}_m, \boldsymbol{X}_2) & \cdots & \kappa(\boldsymbol{X}_m, \boldsymbol{X}_m) \end{bmatrix}$$

是半正定矩阵。上述结论被称为Mercer定理 [3] [1]。定理的证明从略，结论性地，作为充分条件：特征空间内两个函数的内积是一个二元函数，在其核矩阵为半正定矩阵时，该二元函数具有可再生性：$\kappa(\boldsymbol{X}_1, \boldsymbol{X}_2) = \kappa(\cdot, \boldsymbol{X}_1)^{\mathrm{T}} \kappa(\cdot, \boldsymbol{X}_2)$，因此其内积空间是一个赋范向量空间（normed vector space），可以完备化得到希尔伯特空间 $\mathcal{H}$，即再生核希尔伯特空间（Reproducing Kernel Hilbert Space, RKHS）。作为必要条件，对核函数构造核矩阵后易知：

$$\sum_{i,j=1}^{m} G\left(\boldsymbol{X}_i, \boldsymbol{X}_j\right) = \left\| \sum_{i=1}^{m} \phi(\boldsymbol{X}_i)^2 \right\| \geq 0$$

[3]。

- **常见的核函数**

在构造核函数后，验证其对输入空间内的任意格拉姆矩阵为半正定矩阵是困难的，因此通常的选择是使用现成的核函数 [3]。以下给出一些核函数的例子，其中未做说明的参数均是该核函数的超参数（hyper-parameter） [2]：

多项式核（polynomial kernel）	$\kappa(\boldsymbol{X}_1, \boldsymbol{X}_2) = \left(\boldsymbol{X}_1^{\mathrm{T}} \boldsymbol{X}_2\right)^n$
径向基函数核（RBF kernel）	$\kappa(\boldsymbol{X}_1, \boldsymbol{X}_2) = \exp\left(-\frac{\|\boldsymbol{X}_1 - \boldsymbol{X}_2\|^2}{2\sigma^2}\right)$
拉普拉斯核（Laplacian kernel）	$\kappa(\boldsymbol{X}_1, \boldsymbol{X}_2) = \exp\left(-\frac{\|\boldsymbol{X}_1 - \boldsymbol{X}_2\|}{\sigma}\right)$
Sigmoid核（Sigmoid kernel）	$\kappa(\boldsymbol{X}_1, \boldsymbol{X}_2) = \tanh\left[a\left(\boldsymbol{X}_1^{\mathrm{T}} \boldsymbol{X}_2\right) - b\right],$

当多项式核的阶为1时，其被称为线性核，对应的非线性分类器退化为线性分类器。RBF核

也被称为高斯核（Gaussian kernel），其对应的映射函数将样本空间映射至无限维空间。核函数的线性组合和笛卡尔积也是核函数，此外对特征空间内的函数 $g(X)$，
$g(X_1)\kappa(X_1,X_2)g(X_2)$
也是核函数 [2] 。

- SVM算法Demo

基于SVM线性原理处理数据。

```
"""
一个  线性可分svm分类器  的demo
"""

import numpy as np
import matplotlib.pyplot as plt
from matplotlib import style
#style.use("ggplot")
from sklearn import svm

"""
# 可以看原始数据
# 这是原始的数据
x = [1, 5, 1.5, 8, 1, 9]
y = [2, 8, 1.8, 8, 0.6, 11]

plt.scatter(x,y)
plt.show()
“""

# 转换成  样本数  x  特征  的格式
X = np.array([[1,2],[5,8],[1.5,1.8],[8,8],[1,0.6],[9,11]])
# y 表示的是label
y = [0,1,0,1,0,1]

# 惩罚因子C取1.0。如果你不知道C的作用也不用着急，姑且看成是对分类器表现的调节参数
# sklearn 文档  ，https://xacecask2.gitbooks.io/scikit-learn-user-guide-chinese-version/content/sec1.4.html
clf = svm.SVC(kernel='linear', C = 1.0)
clf.fit(X,y)

# clf.coef_ 可以理解为  weights * x + bias ,clf.coef_  只存在与线性核里面
# 这是一个二维平面的分类，所以分类直线的方程是  theta0 * x1 + theta1 * x2 + bias = 0
# clf.coef_[0] 里面的两个数，分别是  theta0 和  theta1 ,intercept_[0] 指的是bias
w = clf.coef_[0]
print(w)

xx = np.linspace(0,12)
# a 是斜率
a = -w[0] / w[1]
# clf.intercept_[0] / w[1] 是截距
yy = a * xx - clf.intercept_[0] / w[1]
```

```
h0 = plt.plot(xx, yy, 'k-', label="non weighted div")
plt.scatter(X[:, 0], X[:, 1], c = y)
pred_x1=0.58
pred_y1=0.76
pred_x2=10.58
pred_y2=10.76

# 预测一个点的类别
print ("x1 and y1 的类别是 ：",clf.predict(np.array( [[pred_x1,pred_y1]] )))
print ("x2 and y2 的类别是 ：",clf.predict(np.array( [[pred_x2,pred_y2]] )))
plt.scatter(0.58, 0.76,color="red")
plt.scatter(10.58,10.76,color="red")
plt.legend()
plt.show()


终端输出:
➜  Python_Workplace ipython svm_demo1.py
[0.1380943  0.24462418]
('x1 and y1
\xe7\x9a\x84\xe7\xb1\xbb\xe5\x88\xab\xe6\x98\xaf\xe3\x80\x80\xef\xbc\x9a',
array([0]))
('x2 and y2
\xe7\x9a\x84\xe7\xb1\xbb\xe5\x88\xab\xe6\x98\xaf\xe3\x80\x80\xef\xbc\x9a',
array([1]))
```

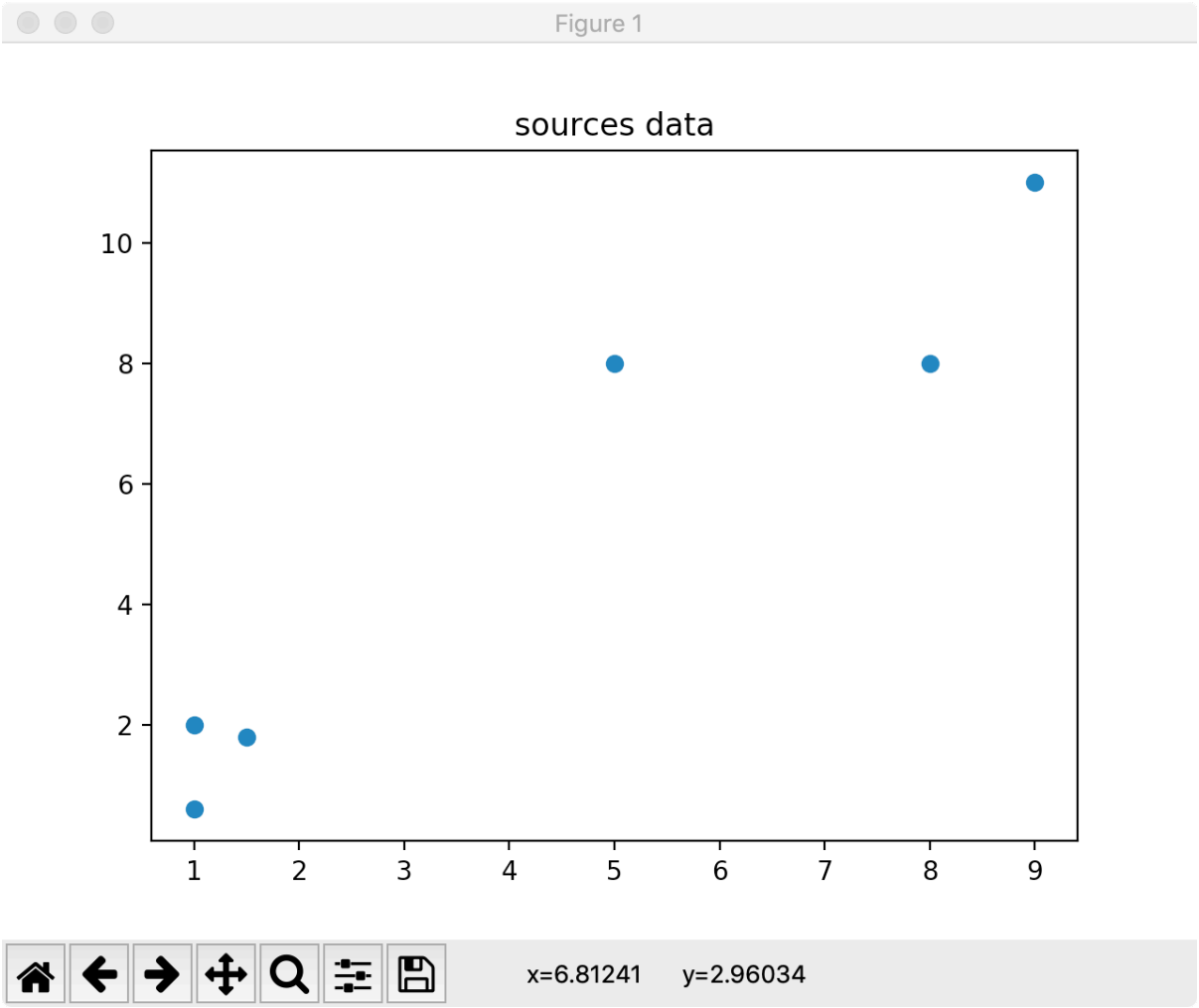
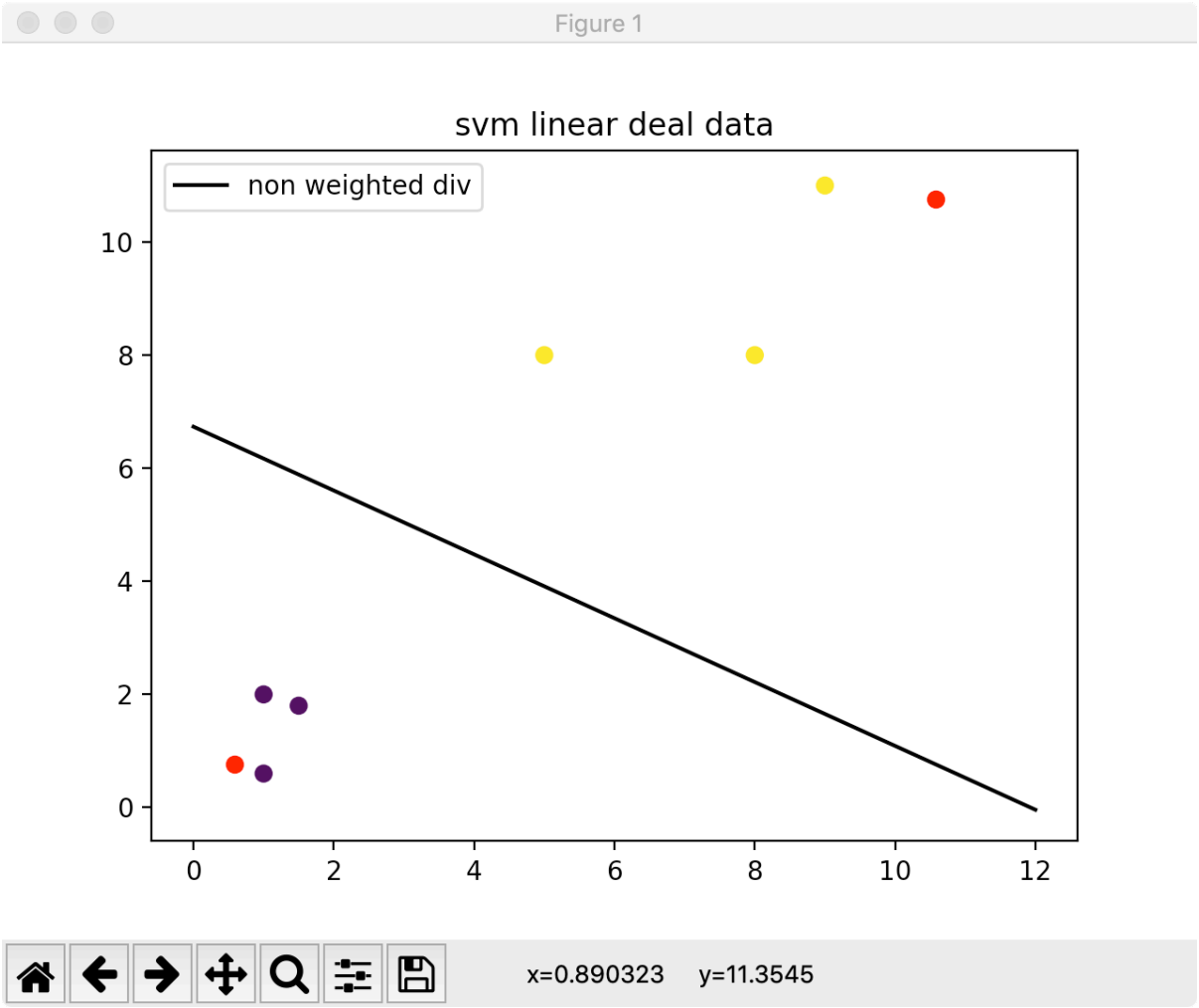


Figure 1
sources data
10
8
6
4
2
1
2
3
4
5
6
7
8
9
x=6.81241
y=2.96034



换一个基于kernel的算法历程：

```
"""
一个  非线性svm分类器  的demo
"""

import numpy as np
import matplotlib.pyplot as plt
from matplotlib import style
from sklearn import svm
#style.use("ggplot")


# 构造了一个非线性的数据
X = np.array([[1,1],[5,5],[5,1],[1,5],[2.5,2.5],[2,3],[3,2],[2,2],[3,3]])
y = [1,1,1,1,0,0,0,0,0]
plt.scatter(X[:, 0], X[:, 1], c = y)
#plt.show()

#svc = svm.SVC(kernel='linear',C=1.0) # 线性核
#svc = svm.SVC(kernel='rbf',C=1.0,gamma=1)  #如果是非线性核  可以还不同的gamma值
```

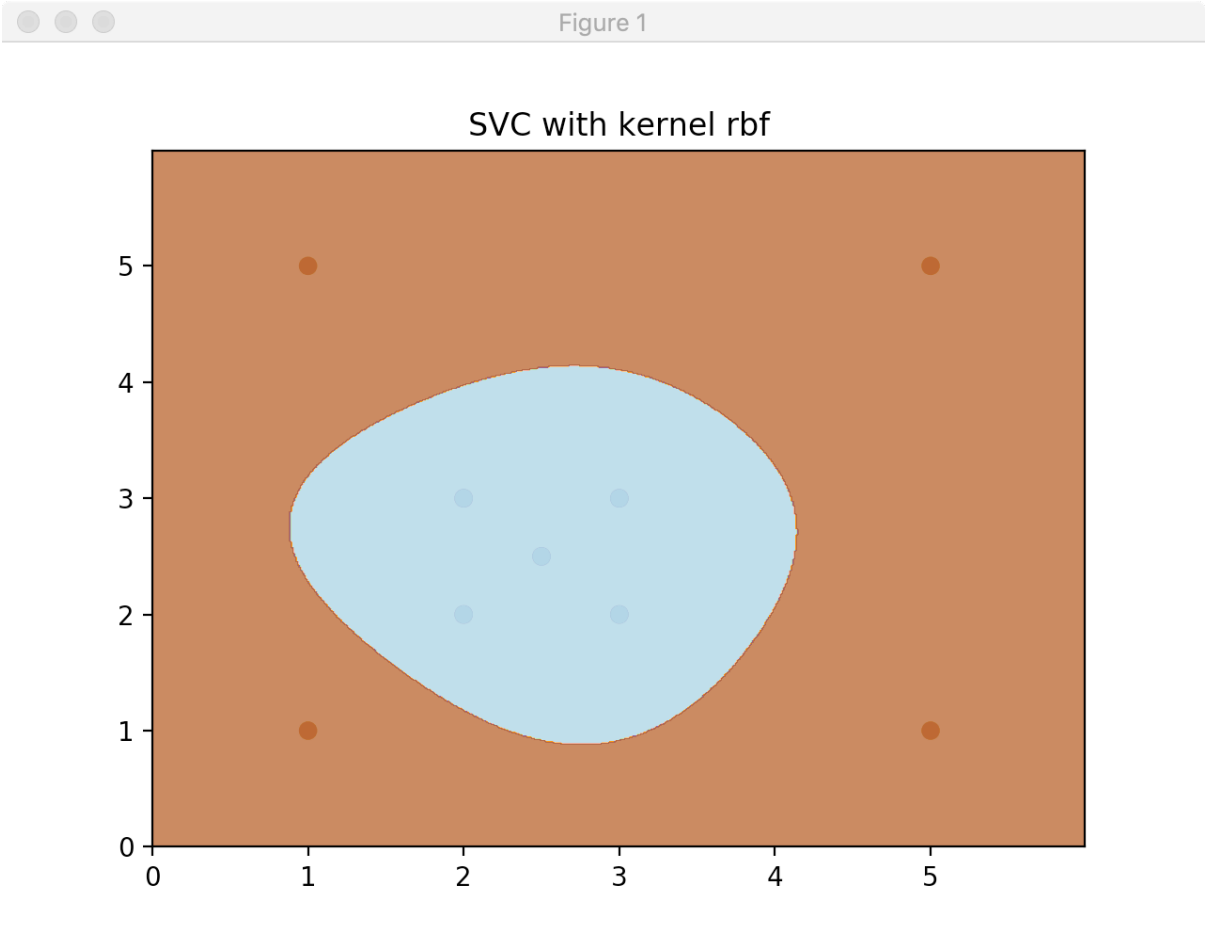
```
svc = svm.SVC(kernel='rbf',C=1.0,gamma='auto')
svc.fit(X,y)

h = 0.01
x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
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.xlim(xx.min(), xx.max())
plt.title("SVC with kernel "+svc.kernel)
plt.show()
```

输出的SVC Kernel rbf:



官方网站也提供了不少学习的例子，提供了SKLearn中的四种方法：

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


def make_meshgrid(x, y, h=.02):
    """Create a mesh of points to plot in

    Parameters
    ----------
    x: data to base x-axis meshgrid on
    y: data to base y-axis meshgrid on
    h: stepsize for meshgrid, optional

    Returns
    -------
    xx, yy : ndarray
    """
    x_min, x_max = x.min() - 1, x.max() + 1
    y_min, y_max = y.min() - 1, y.max() + 1
    xx, yy = np.meshgrid(np.arange(x_min, x_max, h),
                         np.arange(y_min, y_max, h))
    return xx, yy


def plot_contours(ax, clf, xx, yy, **params):
    """Plot the decision boundaries for a classifier.

    Parameters
    ----------
    ax: matplotlib axes object
    clf: a classifier
    xx: meshgrid ndarray
    yy: meshgrid ndarray
    params: dictionary of params to pass to contourf, optional
    """
    Z = clf.predict(np.c_[xx.ravel(), yy.ravel()])
    Z = Z.reshape(xx.shape)
    out = ax.contourf(xx, yy, Z, **params)
    return out


# import some data to play with
iris = datasets.load_iris()
# Take the first two features. We could avoid this by using a two-dim dataset
X = iris.data[:, :2]
y = iris.target

# we create an instance of SVM and fit out data. We do not scale our
# data since we want to plot the support vectors
C = 1.0  # SVM regularization parameter
models = (svm.SVC(kernel='linear', C=C),
          svm.LinearSVC(C=C),
          svm.SVC(kernel='rbf', gamma=0.7, C=C),
```

```
          svm.SVC(kernel='poly', degree=3, C=C))
models = (clf.fit(X, y) for clf in models)

# title for the plots
titles = ('SVC with linear kernel',
          'LinearSVC (linear kernel)',
          'SVC with RBF kernel',
          'SVC with polynomial (degree 3) kernel')

# Set-up 2x2 grid for plotting.
fig, sub = plt.subplots(2, 2)
plt.subplots_adjust(wspace=0.4, hspace=0.4)

X0, X1 = X[:, 0], X[:, 1]
xx, yy = make_meshgrid(X0, X1)

for clf, title, ax in zip(models, titles, sub.flatten()):
    plot_contours(ax, clf, xx, yy,
                  cmap=plt.cm.coolwarm, alpha=0.8)
    ax.scatter(X0, X1, c=y, cmap=plt.cm.coolwarm, s=20, edgecolors='k')
    ax.set_xlim(xx.min(), xx.max())
    ax.set_ylim(yy.min(), yy.max())
    ax.set_xlabel('Sepal length')
    ax.set_ylabel('Sepal width')
    ax.set_xticks(())
    ax.set_yticks(())
    ax.set_title(title)

plt.show()
```

- 在iris数据集中试验不同的SVM分类器，并作图比较
- SVM：最大化间隔分离超平面
- SVM：不平衡类别的分离超平面
- SVM-Anova：带有单变量特征选择的SVM
- 非线性SVM
- SVM：带权重问题的例子