## 模式识别第四次上机大作业

**人工智能32班 20009201208 张佳凡**

###### 问题重述

在Iris和Sonar数据集上验证SVM支持向量机模型的分类效果，并尝试探究用不同的核函数和改变惩罚系数对分类准确率的影响。

###### 二、数据集介绍

、数据集说明

UC 数据库是加州大学欧文分校(University of California, Irvine)提出的用于机器学习的数据库，这个数据库日前共有488个数据集，其数目还在不断增加，UCI 数据集是一个常用的标准测试数据集。

下表为部分Iris及Sonar数据展示：

表 1：Iris数据展示

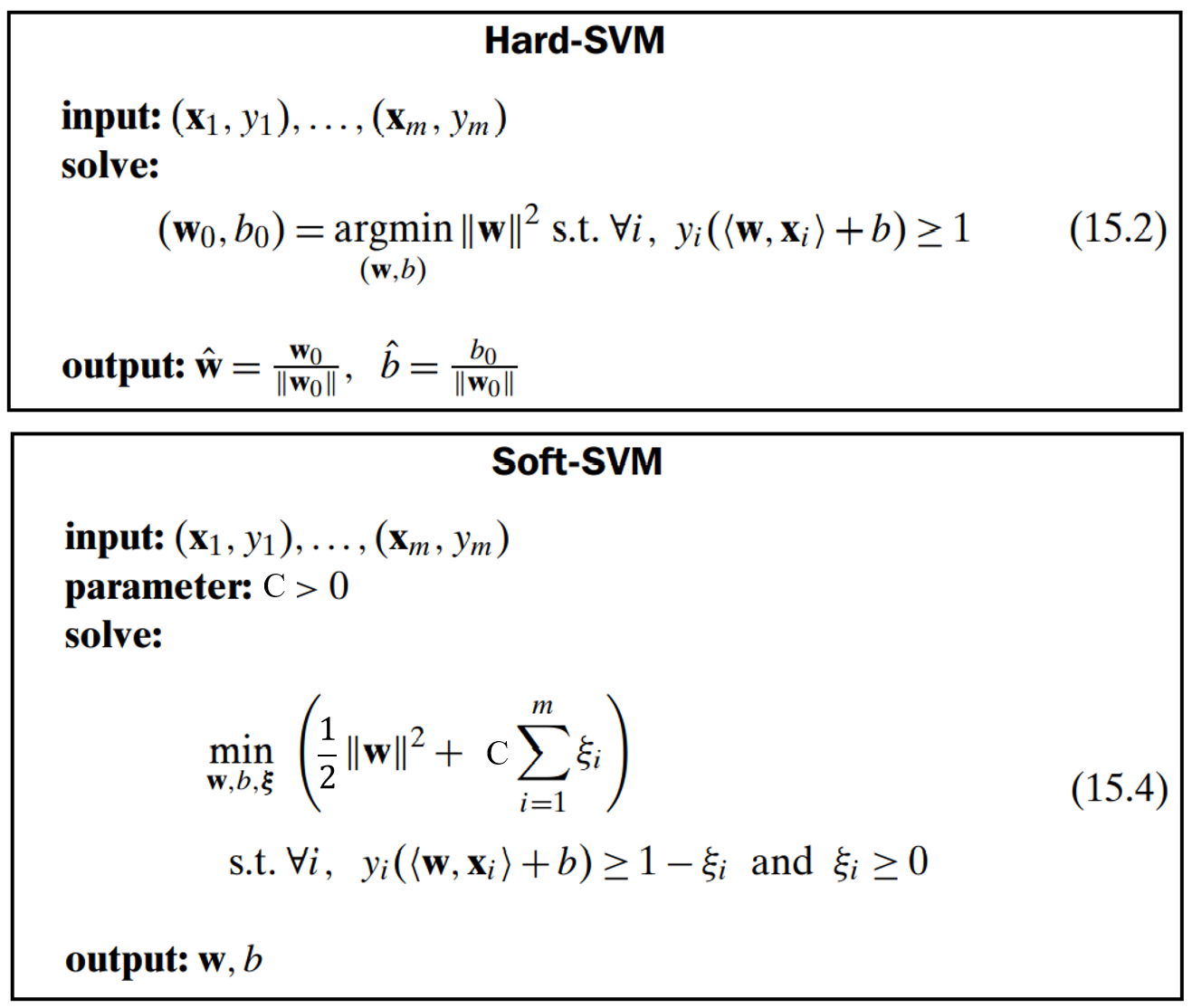
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 5.1 | 3.5 | 1.4 | 0.2 | Iris-setosa |
| 4.9 | 3 | 1.4 | 0.2 | Iris-setosa |
| … |  | … |  | … |
| 7 | 3.2 | 4.7 | 1.4 | Iris-versicolor |
| 6.4 | 3.2 | 4.5 | 1.5 | Iris-versicolor |
| … |  | … |  | … |
| 6.3 | 3.3 | 6 | 2.5 | Iris-virginica |
| 5.8 | 2.7 | 5.1 | 1.9 | Iris-virginica |
| … |  | … |  | … |

表 2：Sonar数据展示

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 0.02 | 0.0371 | 0.0428 |  | 0.0084 | 0.009 | R |
| 0.0453 | 0.0523 | 0.0843 | … | 0.0049 | 0.0052 | R |
| 0.0262 | 0.0582 | 0.1099 |  | 0.0164 | 0.0095 | R |
| 0.01 | 0.0171 | 0.0623 |  | 0.0044 | 0.004 | R |
| … |  | … |  | … | … |  |
| 0.0323 | 0.0101 | 0.0298 |  | 0.0062 | 0.0067 | M |
| 0.0522 | 0.0437 | 0.018 | … | 0.0077 | 0.0031 | M |
| 0.0303 | 0.0353 | 0.049 |  | 0.0036 | 0.0048 | M |
| 0.026 | 0.0363 | 0.0136 |  | 0.0061 | 0.0115 | M |

###### 三、实验原理推导

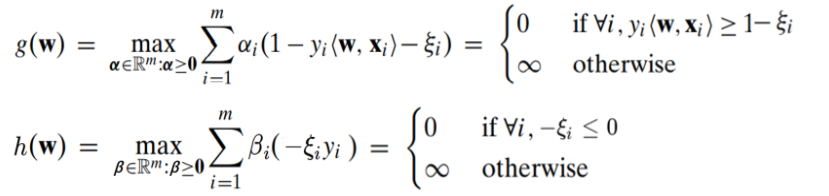
硬间隔和软间隔的目标函数和约束条件，硬间隔就是经典的SVM算法形式，建立在数据线性可分的情况下。而现实生活中，数据常常不可分，为了解决这个问题，允许部分样本出错，因此引入了松弛变量和惩罚因子C，得到了SVM软间隔的算法形式。



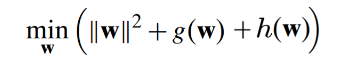
关于惩罚因子C，值越大，表示后者的权重越大，即越不希望出现错误分类的情况；C越小，表示前者的权重越大，允许样本出错，争取omiga的二范数越小。

remark：当C取无穷时，表示后者占据主要地位，不允许出现错误分类，软间隔转化为硬间隔。

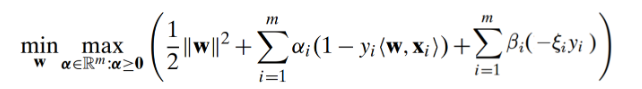
对于SVM软间隔算法形式，我们可以用lagrange乘数法进行化简。两个约束条件转化为如下形式。



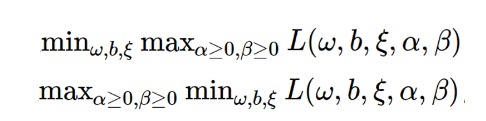
目标函数转化为如下形式



将两个约束条件代入得

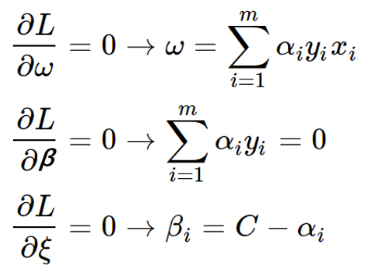


满足KKT条件的话，下面这两个式子可以相互转化

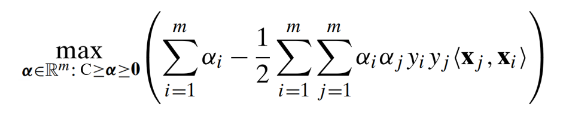


转化为对偶问

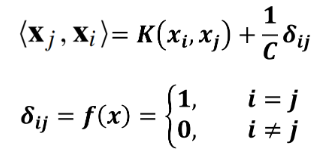
题后，分别对三个变量求导并令等式为0。



代入可求得里面那层的最小值，得到如下式子。



用核方法只需把内积以如下式子变化



求解该凸优化问题可得分类面方程。

以上是线性可分的情况，对于线性不可分问题，可以在错分样本上增加一个惩罚因子来干预最优分类面的确定。这样一来，最优分类面不仅由离分类面最近的样本决定，还要由错分的样本决定。这种情况下的支持向量就由两部分组成：一部分是边界支持向量；另一部分是错分支持向量。

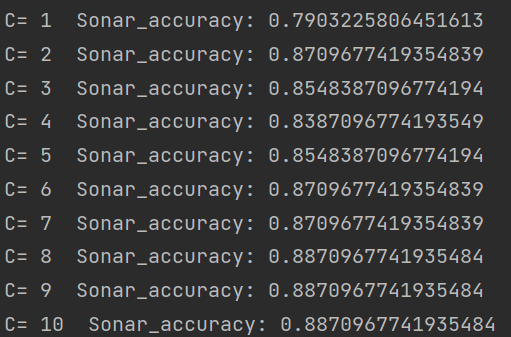
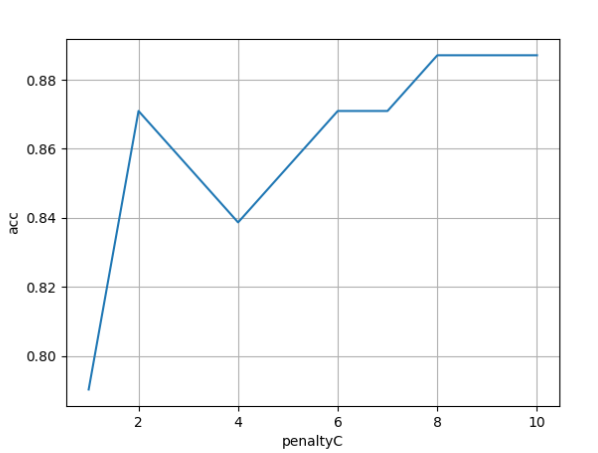
对于非线性的分类问题，可以通过特征变换将非线性问题转化为新空间中的线性问题。但是这样做的代价是会造成样本维数增加，进而导致计算量急剧增加，这就是所谓的“维度灾难”。为了避免高维空间中的计算，可以引入核函数的概念。这样一来，无论变换后空间的维数有多高，这个新空间中的线性支持向量机求解都可以在原空间通过核函数来进行。常用的核函数有多项式核、高斯核（径向基核）、Sigmoid函数。

###### 实验结果

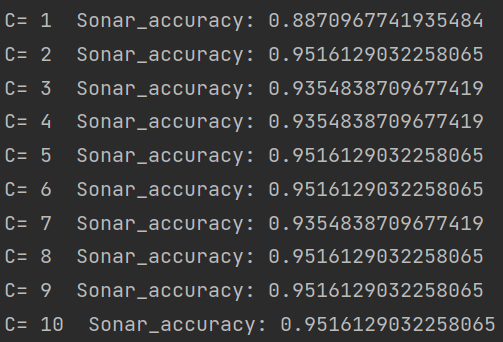
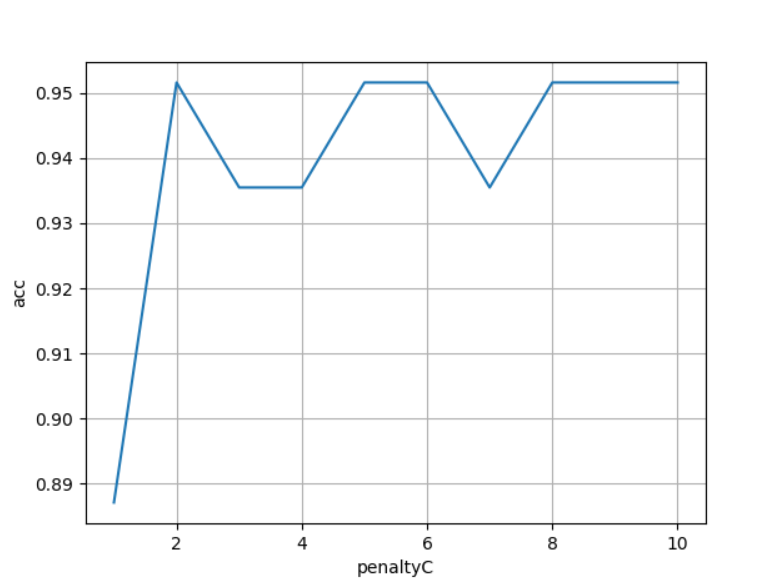
4.1 Sonar数据集

使用Sonar数据集验证SVM算法，在不同核函数下，改变软分类惩罚系数的值，分别测试SVM的分类准确率，并作可视化分析如下：

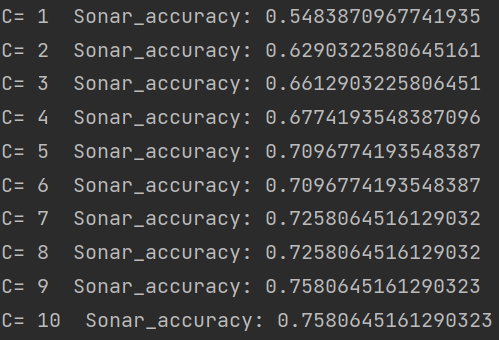
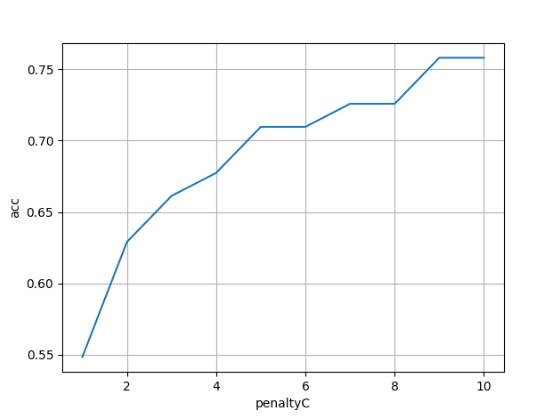
线性核函数 惩罚系数为1-10

Rbf核函数 惩罚系数为1-10

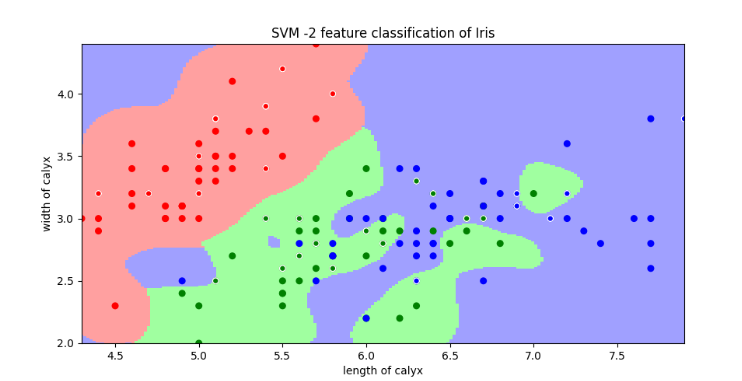
 

多项式核函数 惩罚系数为1-10

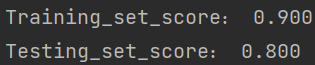
 

4.2 Iris数据集

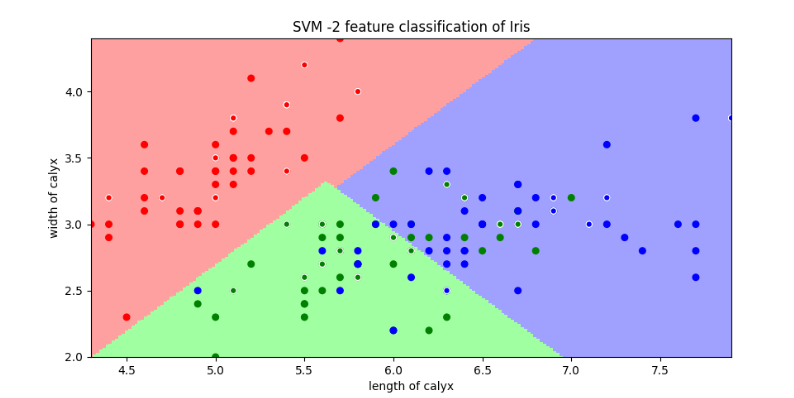
选用rbf核函数，惩罚系数为5的Iris分类结果可视化：



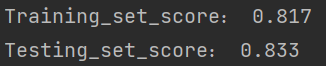
准确率：



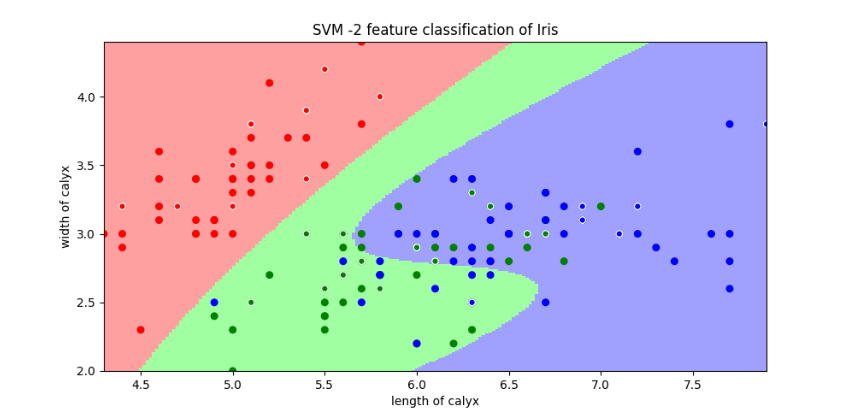
选用线性核函数，分类结果可视化：



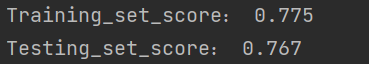
准确率：



选用多项式核函数，惩罚系数为5的Iris分类结果可视化：



准确率：



1. 结果分析

经过上面SVM在Iris和Sonar数据集上的验证，可知选择不同的核函数、调整不同的惩罚系数对于SVM分类结果准确率的影响很大，因此针对不同的情况，应该选择合适的参数可以达到更高的分类准确率。

###### 附录

**6.1 SVM-Sonar.py**

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import cvxopt

import random

from sklearn.metrics import accuracy\_score

# 数据标准化

def generalization(x):

num\_feature = x.shape[1]

for i in range(num\_feature):

x[:, i] = (x[:, i] - min(x[:, i])) / (max(x[:, 1]) - min(x[:, i]))

return x

# 训练集测试集分割

def train\_test\_split(x, y, test\_rate):

num\_x = x.shape[0]

# 打乱

index = list(range(num\_x))

random.shuffle(index)

x = x[index]

y = y[index]

# 分割

split = round(num\_x \* test\_rate)

x\_test = x[:split, :]

x\_train = x[split:, :]

y\_test = y[:split]

y\_train = y[split:]

return x\_train, x\_test, y\_train, y\_test

# Sonar数据集读取

# 208个样本，60维特征（不同角度返回的值），二分类结果为岩石/金属

data2 = pd.read\_csv('.\data\sonar.all-data', header=None)

data2 = np.array(data2)

x2 = np.float64(data2[:, 0:60])

y2 = data2[:, 60]

for i in range(len(y2)):

if y2[i] == 'R':

y2[i] = -1

else:

y2[i] = 1

y2 = np.int64(y2)

x2\_train, x2\_test, y2\_train, y2\_test = train\_test\_split(x2, y2, test\_rate=0.3)

# 定义三种常见的核函数

# 线性核

def linear\_kernal(\*\*kwargs):

def K(x1, x2):

return np.inner(x1, x2)

return K

# 多项式核

def poly\_kernal(power, coef, \*\*kwargs):

def K(x1, x2):

return (np.inner(x1, x2) + coef) \*\* power

return K

# 径向基核

def rbf\_kernal(gamma, \*\*kwargs):

def K(x1, x2):

distance = np.linalg.norm(x1 - x2) \*\* 2

return np.exp(-gamma \* distance)

return K

class svm:

# 输入参数

# param kernal:核函数

# param penaltyC:软间隔惩罚项C

# param power, gamma,coef为核函数的一些参数

def \_\_init\_\_(self, kernal=linear\_kernal, penaltyC=1, power=1, gamma=1, coef=1):

self.kernal = kernal

self.penaltyC = penaltyC

self.power = power

self.gamma = gamma

self.coef = coef

self.kernal = self.kernal(

power=self.power,

gamma=self.gamma,

coef=self.coef)

def train(self, x, y):

x\_num = x.shape[0]

kernal\_matrix = self.kernal(x, x) + (1 / self.penaltyC) \* np.eye(x\_num)

# 计算标准凸二次规划的几个参数

p = cvxopt.matrix(kernal\_matrix \* np.outer(y, y))

q = cvxopt.matrix(-np.ones([x\_num, 1], np.float64))

g = cvxopt.matrix(-np.eye(x\_num))

h = cvxopt.matrix(np.zeros([x\_num, 1], np.float64))

y = np.float64(y)

a = cvxopt.matrix(y, (1, x\_num))

b = cvxopt.matrix(0.)

# 使用凸规划工具包cvxopt求解SVM目标函数（算lagrange乘子）

cvxopt.solvers.options['show\_progress'] = False

solution = cvxopt.solvers.qp(p, q, g, h, a, b)

alpha = np.float32(np.array(solution['x']))

alpha[alpha <= 1e-4] = 0

# 求权重w和截距b

w = np.sum(np.reshape(y, [-1, 1]) \* alpha \* x, axis=0)

b = np.mean(np.reshape(y, [-1, 1]) - np.reshape(np.dot(w, np.transpose(x)), [-1, 1]))

self.w = w

self.b = b

return w, b, alpha

def predict(self, x\_test):

y\_predict = []

for sample in x\_test:

predict1 = self.kernal(self.w, sample) + self.b

predict1 = np.int64(np.sign(predict1))

y\_predict.append(predict1)

# y\_predict.tolist()

return y\_predict

if \_\_name\_\_ == '\_\_main\_\_':

a = []

for i in range(1,11):

svm2 = svm(poly\_kernal, penaltyC=i\*i)

w2, b2, alpha2 = svm2.train(x2, y2)

y2\_predict = svm2.predict(x2\_test)

print('C=',i,' Sonar\_accuracy:', accuracy\_score(y2\_test, y2\_predict))

a.append(accuracy\_score(y2\_test, y2\_predict))

plt.figure(1)

plt.plot(range(1,11),a)

plt.grid()

plt.xlabel('penaltyC')

plt.ylabel('acc')

plt.show()

**6.2 SVM-Iris.py**

from sklearn import svm

import numpy as np

import matplotlib.pyplot as plt

import matplotlib

from sklearn.model\_selection import train\_test\_split

# 函数定义部分

# 定义转换器字典用于转换对应兰花数字（相当于loadtxt函数转换器参数converters的字典）

def Sort\_dic(type):

it = {b'Iris-setosa': 0, b'Iris-versicolor': 1, b'Iris-virginica': 2}

return it[type]

# 具体实现部分

# 读取数据集的数据并进行简单清洗

path = 'data/Iris.data'

# converters是数据转换器定义，将第5列的花名格式str转化为0,1,2三种数字分别代表不同的类别兰花类别（这是一步数据清洗过程）

data = np.loadtxt(path, dtype=float, delimiter=',', converters={4: Sort\_dic})

# 将数据和标签列划分开来

# split函数的参数意义(数据，分割位置（这里用了一种不常见的写法表示前四列为一组记作x，后面剩余部分为一组记作y），

# axis = 1（代表水平分割，以每一个行记录为切割对象） 或 0（代表垂直分割，以属性为切割对象）)。

x, y = np.split(data, indices\_or\_sections=(4,), axis=1) # x为数据，y为标签

# 为便于后边画图显示，只选取前两维度。若不用画图，可选取前四列x[:,0:4]就选中所有特征了

x = x[:, 0:2] # 标记一下，这个切片的意思是提取前两列的每一行[每一行,0,1两列]

# Sklearn库函数train\_test\_split可以实现将数据集按比例划分为训练集和测试集

train\_data, test\_data, train\_label, test\_label = train\_test\_split(x, y, random\_state=1, train\_size=0.8, test\_size=0.2)

# 目前x为数据y为标签（即标注样本属于哪一类）

Training\_set\_score=1

Testing\_set\_score=5

# C越大分类效果越好，但有可能会过拟合，gamma是高斯rbf核参数，而后面的dfs制定了类别划分方式，ovr是一对多方式。

classifier = svm.SVC(C=5, kernel='poly', gamma=20, decision\_function\_shape='ovr')

classifier.fit(train\_data, train\_label.ravel()) # 用训练集数据来训练模型。（ravel函数在降维时默认是行序优先）

# 计算svc分类器的准确率

print("Training\_set\_score：", format(classifier.score(train\_data, train\_label), '.3f'))

print("Testing\_set\_score：", format(classifier.score(test\_data, test\_label), '.3f'))

# 绘制图形将实验结果可视化

# 首先确定坐标轴范围，通过二维坐标最大最小值来确定范围

# 第1维特征的范围（花萼长度）

x1\_min = x[:, 0].min()

x1\_max = x[:, 0].max()

# 第2维特征的范围（花萼宽度）

x2\_min = x[:, 1].min()

x2\_max = x[:, 1].max()

# mgrid方法用来生成网格矩阵形式的图框架

x1, x2 = np.mgrid[x1\_min:x1\_max:200j, x2\_min:x2\_max:200j] # 生成网络采样点（其实是颜色区域），先沿着x1向右扩展，再沿着x2向下扩展

grid\_test = np.stack((x1.flat, x2.flat), axis=1) # 再通过stack()函数，axis=1，生成测试点，其实就是合并横与纵等于计算x1+x2

grid\_value = classifier.predict(grid\_test) # 用训练好的分类器去预测这一片面积内的所有点，为了画出不同类别区域

grid\_value = grid\_value.reshape(x1.shape) # （大坑）使刚刚构建的区域与输入的形状相同（裁减掉过多的冗余点，必须写不然会导致越界读取报错，这个点的bug非常难debug）

# 设置两组颜色（高亮色为预测区域，样本点为深色）

light\_camp = matplotlib.colors.ListedColormap(['#FFA0A0', '#A0FFA0', '#A0A0FF'])

dark\_camp = matplotlib.colors.ListedColormap(['r', 'g', 'b'])

fig = plt.figure(figsize=(10, 5)) # 设置窗体大小

fig.canvas.set\_window\_title('SVM -2 feature classification of Iris') # 设置窗体title

# 使用pcolormesh()将预测值（区域）显示出来

plt.pcolormesh(x1, x2, grid\_value, cmap=light\_camp)

plt.scatter(x[:, 0], x[:, 1], c=y[:, 0], s=30, cmap=dark\_camp) # 加入所有样本点，以深色显示

plt.scatter(test\_data[:, 0], test\_data[:, 1], c=test\_label[:, 0], s=30, edgecolors='white', zorder=2, cmap=dark\_camp)

# 单独再把测试集样本点加一个圈,更加直观的查看命中效果

# 设置图表的标题以及x1,x2坐标轴含义

plt.title('SVM -2 feature classification of Iris')

plt.xlabel('length of calyx')

plt.ylabel('width of calyx')

# 设置坐标轴的边界

plt.xlim(x1\_min, x1\_max)

plt.ylim(x2\_min, x2\_max)

plt.show()