**Lab 11 sklearn中的SVM应用实践**

|  |  |  |  |
| --- | --- | --- | --- |
| 学号 | 姓名 | 年级 | 专业 |
| 18353070 | 谭嘉伟 | 2018 | 软件工程 |

1. 实验目的：
2. 进一步熟悉python中的sklearn库应用
3. 学习并实践使用sklearn中模块实现支持向量机
4. 进一步熟练用Python进行程序设计和处理模型的能力
5. 提升对实际问题进行建模分析，用机器学习处理实际问题的能力
6. 进一步熟练提升用Python处理数据集可视化的能力
7. 在实践中熟悉调试Python程序的技巧
8. 进一步巩固学习到的支持向量机理论知识
9. 实验环境：

1、硬件环境：PC 1.60GHz四核处理器 8.00G内存

2、软件环境：Windows10 64位操作系统

Python3.5.6 tensorflow1.4.0 Jupyter Notebook

1. 实验步骤：
2. 准备数据

从sklearn.datasets中导入iris数据集

在jupyter notebook的单元格中输入以下代码：

from sklearn.datasets import load\_iris

iris=load\_iris().data

iris\_target=load\_iris().target

iris\_target

可以得到目标数据数组，然后再输入代码对数据进行预处理

from sklearn.preprocessing import MinMaxScaler

MinMax=MinMaxScaler()

MinMax.fit(iris)

iris\_transf=MinMax.transform(iris)

iris\_transf

然后，再对这里的数据集进行进一步分割，将数据集分为测试集和训练集两部分

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

X\_train,X\_test,y\_train,y\_test=train\_test\_split(iris\_transf,iris\_target,random\_state=14)

1. 构造SVM分类器

利用sklearn中的SVC估计器，用默认参数进行训练

from sklearn.svm import SVC

svc=SVC()

svc.fit(X\_train,y\_train)

1. 模型评估

运行下列代码

print("test\_svc.score: {:.3f}".format(svc.score(X\_test,y\_test)))

print("test\_svc.score: {:.3f}".format(svc.score(X\_train,y\_train)))

1. 模型预测

运行下面的代码对给定的一个数据进行模型预测

predict\_labels=svc.predict([[0.13888889,0.58333333,0.15254237,0.04166667]])

print(predict\_labels)

1. 利用Grid\_searchCV寻找最佳参数

输入以下代码网格搜寻最佳参数（逐一枚举候选值）

from sklearn.model\_selection import GridSearchCV

from sklearn.svm import SVC

svc=SVC()

tuned\_parameters=[{'kernel':['rbf'],'gamma':[1e-3,1e-4],'C':[1,10,100,1000]},{'kernel':['linear'],'C':[1,10,100,1000]}]

clf=GridSearchCV(svc,tuned\_parameters)

clf.fit(X\_train,y\_train)

print("Best parameters: ",clf.best\_params\_)

1. 利用获得的最优参数预测

输入以下代码并运行

from sklearn.svm import SVC

svc=SVC(C=1,kernel='linear')

svc.fit(X\_train,y\_train)

print("test\_svc.score: {:.3f}".format(svc.score(X\_test,y\_test)))

print("test\_svc.score: {:.3f}".format(svc.score(X\_train,y\_train)))

1. 利用matplotlib可视化不同参数模型的准确率变化

输入并运行以下代码

from sklearn.svm import SVC

Cs=[]

score=[]

for c in range(10,100,10):

svc=SVC(C=c,gamma=0.001,kernel='rbf')

svc.fit(X\_train,y\_train)

accuracy=svc.score(X\_train,y\_train)

print("the accuracy of svc model with C = {0} is {1}".format(c,accuracy))

Cs.append(c)

score.append(accuracy)

import matplotlib.pyplot as plt

%matplotlib inline

plt.title("SVC model: argument: C")

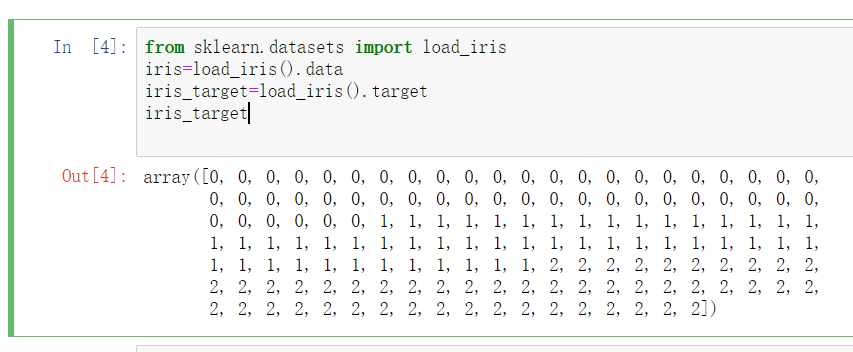
plt.xlabel("C")

plt.ylabel("Accuracy")

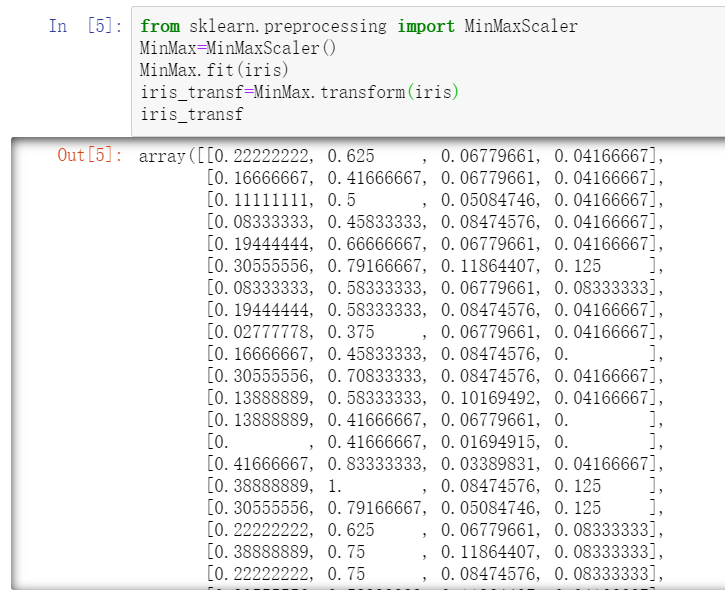
plt.plot(Cs,score,'b')

1. 实验结果与分析：
2. 准备数据

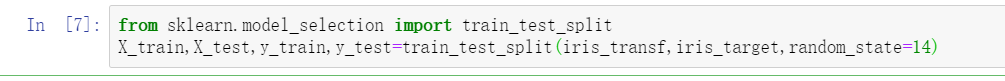
导入iris数据集的输出结果：



预处理的输出结果为:



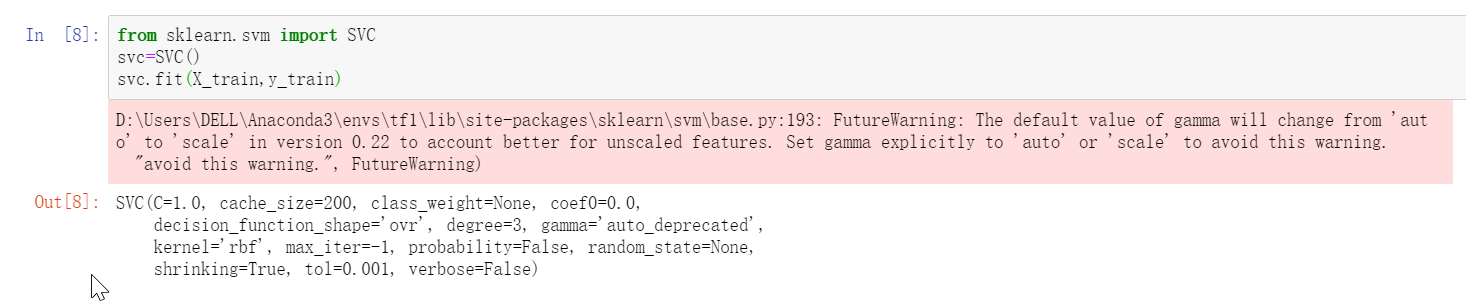
这一部分预处理的目的是将sklearn的iris数据集数据进行标准化，将数据用Min和Max从[Min,Max]区间标准化为[0,1]区间



这里的train\_test\_split起到了将数据集根据实际需要划分为训练集和测试集的作用。参数random\_state=14是随机数种子，用于这个划分数据集的算法产生随机数。

1. 构造SVM分类器

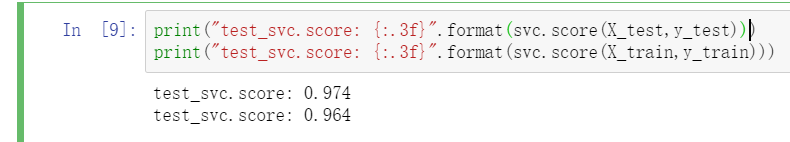
代码运行结果如下



成功构建了需要的SVM分类器

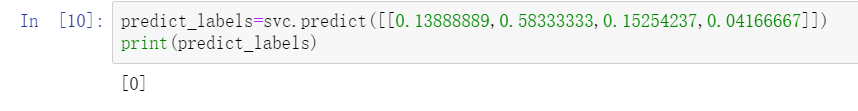
1. 模型评估

代码运行结果如下



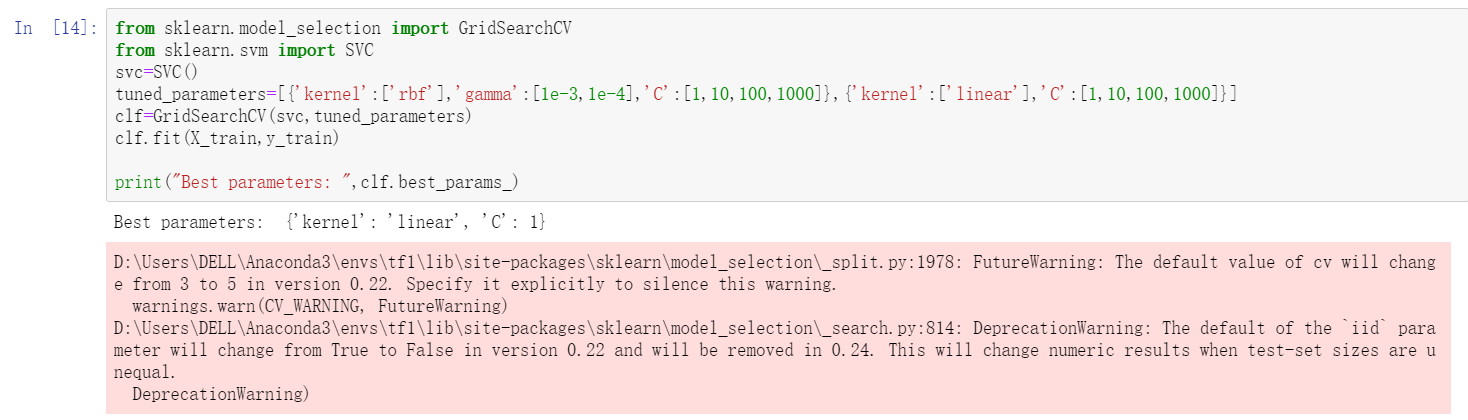
.3f用于python格式控制，让后面的正确率保留三位小数输出。输出结果训练集正确率达到97.4%，测试集正确率达到96.4%，正确率都非常高

1. 模型预测



该数据集被分到了第0类

1. 利用Grid\_searchCV寻找最佳参数



找到的最佳参数为'kernel': 'linear', 'C': 1

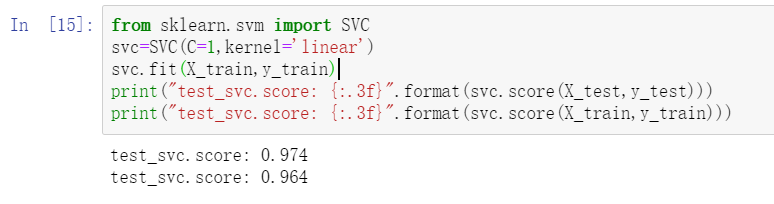
其中，SVM参数的含义为

kernel:核函数类型，linear为线性核函数，rbf为高斯核函数

C为错误项的惩罚因子，默认值为1

gamma为核函数系数

1. 利用获得的最优参数预测

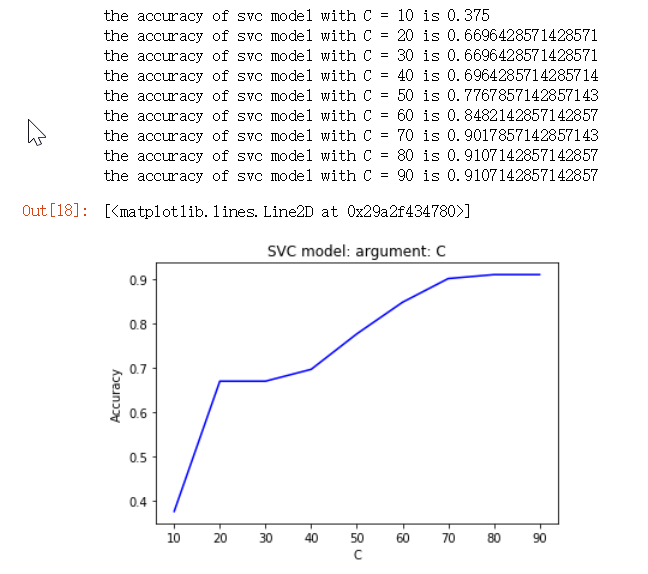


结果如上图

训练集正确率为97.4%,测试集正确率为96.4%，训练效果比较理想

1. 利用matplotlib可视化不同参数模型的准确率变化

运行结果如下：



代码中range(10,100,10)生成了一个[10,100)间隔为10的列表，然后在这个列表中枚举C，求解对应的accuracy，将所有的(C,accuracy)对进行画图，得到上面的结果。

可以看出，在这个数据集中，错误因子C越大，正确率就越高。可以在之后进行更大范围的枚举，以及更高精度的枚举，找到最优的参数C。

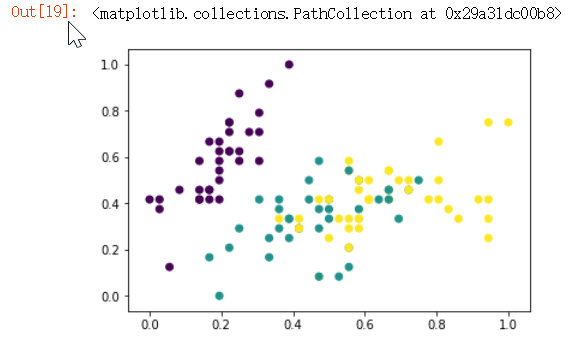
1. 作业：
2. 参照PPT，可视化基于iris数据集的SVM分类器的决策边界，并操作说明decision\_function()方法的功能和用法，以及matplotlib.pyplot.contour()的功能和用法

输入以下代码对数据点可视化(iris数据集有四个维度，这里用前两个维度做SVM并进行画图，这个数据集有三类标签，所以在后面的代码用两个维度数据来分三类)：

clf=SVC()

clf.fit(X\_train,y\_train)

plt.scatter(X\_train[:,0],X\_train[:,1],c=y\_train)



输入以下代码可以画出SVM分类器的决策边界：

import numpy as np

def plot\_hyperplane(clf, X, y):

plt.scatter(X\_train[:,0],X\_train[:,1],c=y\_train)

ax=plt.gca()

xlim=ax.get\_xlim()

ylim=ax.get\_ylim()

xx=np.linspace(xlim[0],xlim[1],30)

yy=np.linspace(ylim[0],ylim[1],30)

plt.xlim(xx.min(), xx.max())

plt.ylim(yy.min(), yy.max())

YY,XX=np.meshgrid(yy,xx)

#print(len(XX[0]))

#print(len(YY[0]))

xy=np.vstack([XX.ravel(),YY.ravel()]).T

#print(clf.decision\_function(xy))

Z0=clf.decision\_function(xy)[:,0].reshape(XX.shape)

#print(clf.decision\_function(xy))

ax.contour(XX,YY,Z0,colors='r',levels=[-1,0,1],alpha=0.8,linestyles=['--','-','--'])

Z1=clf.decision\_function(xy)[:,1].reshape(XX.shape)

ax.contour(XX,YY,Z1,colors='k',levels=[-1,0,1],alpha=0.8,linestyles=['--','-','--'])

Z2=clf.decision\_function(xy)[:,2].reshape(XX.shape)

ax.contour(XX,YY,Z2,colors='g',levels=[-1,0,1],alpha=0.8,linestyles=['--','-','--'])

#ax.scatter(clf.support\_vectors\_[:,0],clf.support\_vectors\_[:,1],s=100,linewidth=1,c='g')

plt.show()

from sklearn import svm

plt.figure(figsize=(12, 4), dpi=144)

#print(type(X\_train[1]))

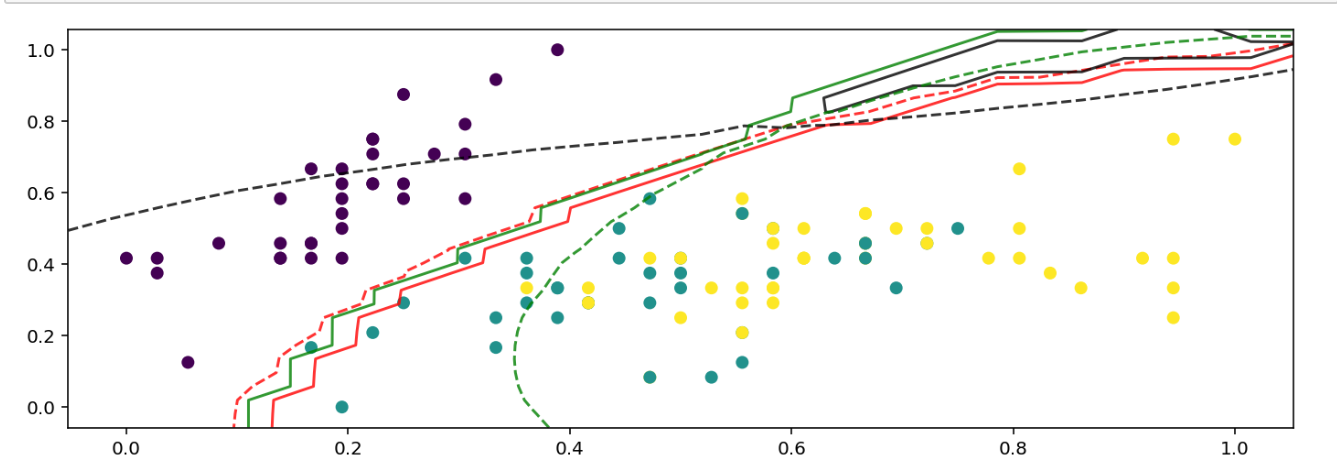
x2train=X\_train.T[range(0,2)].T

#print(x2train)

clf = svm.SVC(C=50.0, kernel='rbf',gamma=1)

clf.fit(x2train, y\_train)

plot\_hyperplane(clf, x2train, y\_train)



上图为clf = svm.SVC(C=50.0, kernel='rbf',gamma=1)的画图结果

尝试用另一个参数的SVC

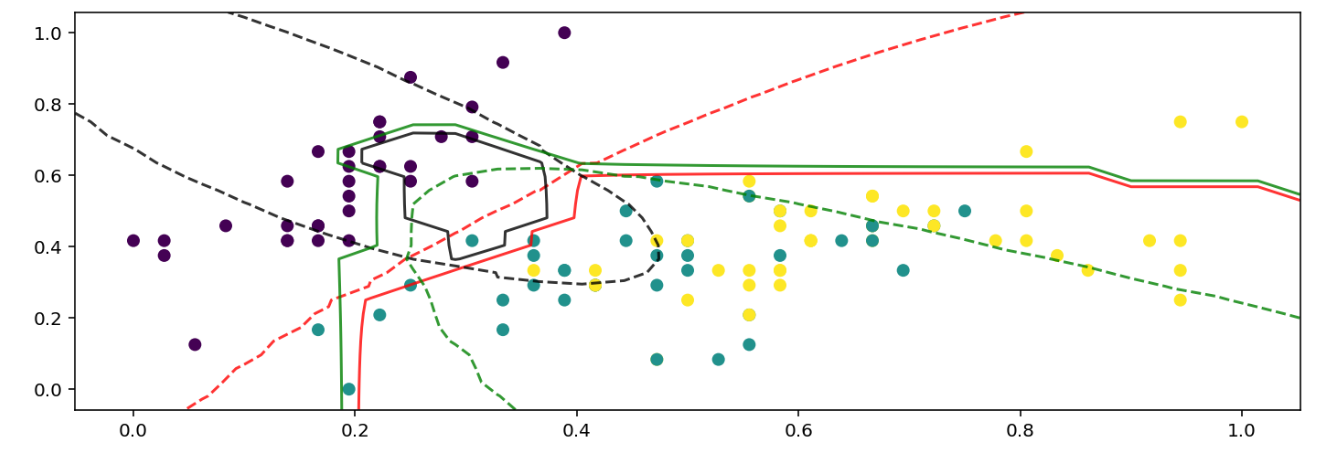
plt.figure(figsize=(12, 4), dpi=144)

clf = svm.SVC(C=5.0, kernel='sigmoid',gamma=2)

clf.fit(x2train, y\_train)

plot\_hyperplane(clf, x2train, y\_train)

核函数采用sigmoid函数，gamma值为2，画图结果为



其中，numpy.meshgrid()的功能是生成网格坐标矩阵

X,Y=np.meshgrid(x,y)将会对于每一个x[i],y[j]都生成出一个对应(i,j)对的点(X[i],Y[j])

Np.linspace()的功能是生成等差数列

numpy.linspace(start, stop, num=50)生成一个值域在start~stop的等差数列，num为项数，默认为50

ravel()有将多维数组转换为一维数组的功能

np.vstack:可以按垂直方向（行的方向）堆叠几个数组构成一个新的数组

decision\_function()的功能：计算样本点到分割超平面的函数距离(归一化后的距离)

如果样本有k类，参数X有n项，那么函数decision\_function(X)将返回一个k\*(k-1)/2列，n行的 ndarray

因为样本有k类，所以sklearn的SVC对每两类两两做了一个超平面那分割，得到k\*(k-1)/2个超平面，decision\_function(X)[:,i]就返回了第i个超平面的函数距离

decision\_function(X)[j,i]表示第j个点到第i个超平面的函数距离(有正有负)，根据这个函数距离的正负性，可以判断出这个点处于超平面的哪一侧。

由于数据集有3个类别，所以对应于3个超平面，上面的图将三个超平面都画在了一个图中

matplotlib.pyplot.contour()用于绘制函数的轮廓，可以画出等高线图

函数原型为contour([X, Y,] Z, [levels], \*\* kwargs)，

X与Y是可选的，X与Y是2-d时，形状(shape)必须与Z一样；如果X余Y是1-d时，则len(X)与Z行数相同，len(Y)与Z列数相同

Z是函数值

levels确定了轮廓线(等高线)的数量和高度，对于对应于levels[i]的每一条等高线，都绘制出对应于这一条高度的曲线

下面的代码对于每一个超平面分别画出对应的图

def plot\_ahyperplane(clf, X, y,na,c):

plt.scatter(X\_train[:,0],X\_train[:,1],c=y\_train)

ax=plt.gca()

xlim=ax.get\_xlim()

ylim=ax.get\_ylim()

xx=np.linspace(xlim[0],xlim[1],30)

yy=np.linspace(ylim[0],ylim[1],30)

plt.xlim(xx.min(), xx.max())

plt.ylim(yy.min(), yy.max())

YY,XX=np.meshgrid(yy,xx)

#print(len(XX[0]))

#print(len(YY[0]))

xy=np.vstack([XX.ravel(),YY.ravel()]).T

#print(clf.decision\_function(xy))

Z=clf.decision\_function(xy)[:,na].reshape(XX.shape)

#print(clf.decision\_function(xy))

ax.contour(XX,YY,Z,colors=c,levels=[-0.3,0,0.3],alpha=0.8,linestyles=['--','-','--'])

Z1=clf.decision\_function(xy)[:,1].reshape(XX.shape)

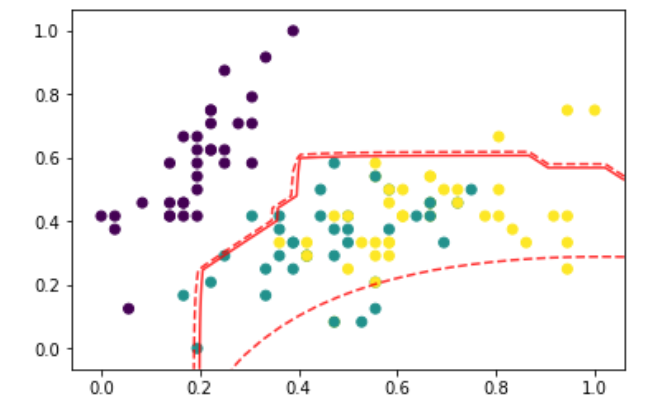
#ax.scatter(clf.support\_vectors\_[:,0],clf.support\_vectors\_[:,1],s=100,linewidth=1,c='g')

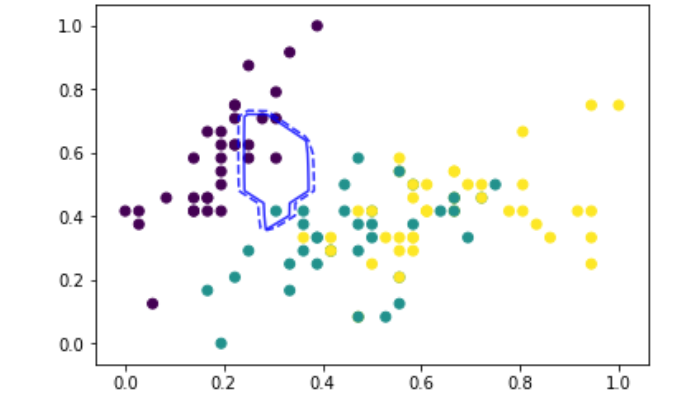
plt.show()

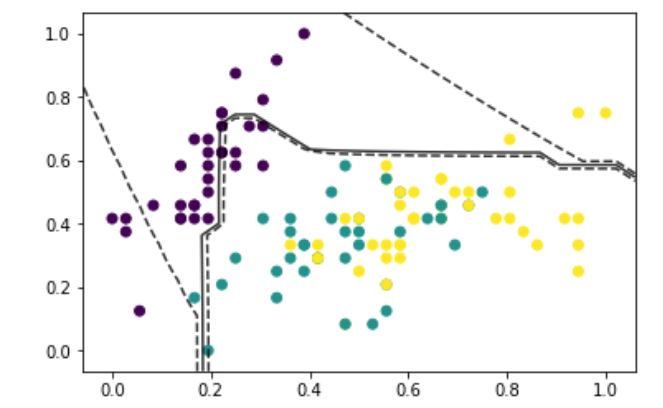
plot\_ahyperplane(clf, x2train, y\_train,0,'r')

plot\_ahyperplane(clf, x2train, y\_train,1,'b')

plot\_ahyperplane(clf, x2train, y\_train,2,'k')







1. 按照上述步骤，对手写体数字集识别，并利用matplotlib可视化测试集中前10个数据的灰度图像

from sklearn.datasets import load\_digits

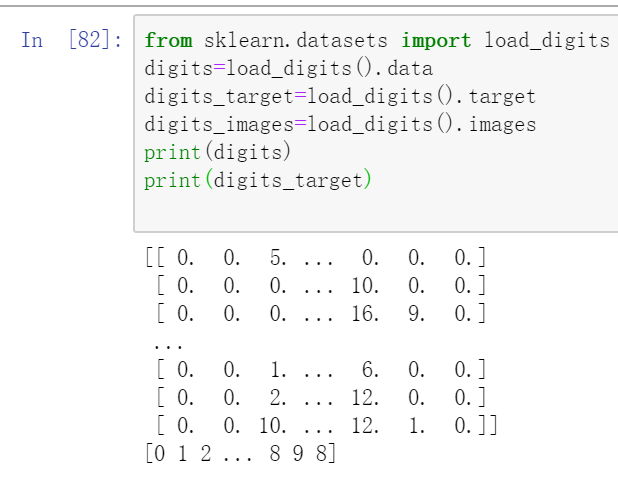
digits=load\_digits().data

digits\_target=load\_digits().target

digits\_images=load\_digits().images

print(digits)

print(digits\_target)



导入结果如上图

对数据集进行预处理如下

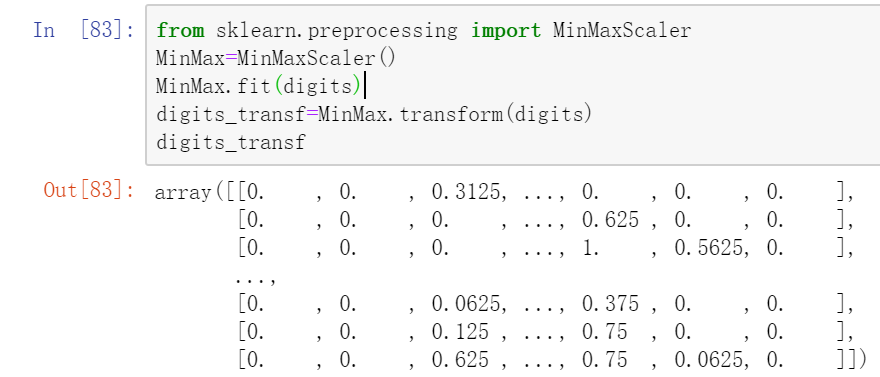
from sklearn.preprocessing import MinMaxScaler

MinMax=MinMaxScaler()

MinMax.fit(digits)

digits\_transf=MinMax.transform(digits)

digits\_transf



进行数据集分割：

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

X\_train,X\_test,y\_train,y\_test=train\_test\_split(digits\_transf,digits\_target,random\_state=18)

建立支持向量机并输出正确率：

from sklearn.svm import SVC

svc=SVC()

svc.fit(X\_train,y\_train)

print("test\_svc.score: {:.3f}".format(svc.score(X\_test,y\_test)))

print("test\_svc.score: {:.3f}".format(svc.score(X\_train,y\_train)))

正确率达到96.9%，训练效果很好

#手写数字搜寻最佳参数

from sklearn.model\_selection import GridSearchCV

from sklearn.svm import SVC

svc=SVC()

tuned\_parameters=[{'kernel':['rbf'],'gamma':[1e-3,1e-4],'C':[1,10,100,1000]},{'kernel':['linear'],'C':[1,10,100,1000]}]

clf=GridSearchCV(svc,tuned\_parameters)

clf.fit(X\_train,y\_train)

print("Best parameters: ",clf.best\_params\_)

搜索结果为

Best parameters: {'C': 1, 'kernel': 'linear'}

观察C对训练效果影响并可视化

from sklearn.svm import SVC

Cs=[]

score=[]

for c in range(10,400,10):

svc=SVC(C=c,gamma=0.001,kernel='rbf')

svc.fit(X\_train,y\_train)

accuracy=svc.score(X\_train,y\_train)

print("the accuracy of svc model with C = {0} is {1}".format(c,accuracy))

Cs.append(c)

score.append(accuracy)

import matplotlib.pyplot as plt

%matplotlib inline

plt.title("SVC model: argument: C")

plt.xlabel("C")

plt.ylabel("Accuracy")

plt.plot(Cs,score,'b')

the accuracy of svc model with C = 10 is 0.9547141796585004

the accuracy of svc model with C = 20 is 0.9710467706013363

the accuracy of svc model with C = 30 is 0.9762435040831478

the accuracy of svc model with C = 40 is 0.9806978470675576

the accuracy of svc model with C = 50 is 0.9829250185597624

the accuracy of svc model with C = 60 is 0.9873793615441723

the accuracy of svc model with C = 70 is 0.9881217520415738

the accuracy of svc model with C = 80 is 0.9888641425389755

the accuracy of svc model with C = 90 is 0.9896065330363771

the accuracy of svc model with C = 100 is 0.9903489235337788

the accuracy of svc model with C = 110 is 0.9910913140311804

the accuracy of svc model with C = 120 is 0.991833704528582

the accuracy of svc model with C = 130 is 0.991833704528582

the accuracy of svc model with C = 140 is 0.991833704528582

the accuracy of svc model with C = 150 is 0.991833704528582

the accuracy of svc model with C = 160 is 0.9925760950259837

the accuracy of svc model with C = 170 is 0.9925760950259837

the accuracy of svc model with C = 180 is 0.991833704528582

the accuracy of svc model with C = 190 is 0.991833704528582

the accuracy of svc model with C = 200 is 0.991833704528582

the accuracy of svc model with C = 210 is 0.991833704528582

the accuracy of svc model with C = 220 is 0.991833704528582

the accuracy of svc model with C = 230 is 0.9925760950259837

the accuracy of svc model with C = 240 is 0.9933184855233853

the accuracy of svc model with C = 250 is 0.9933184855233853

the accuracy of svc model with C = 260 is 0.9933184855233853

the accuracy of svc model with C = 270 is 0.9933184855233853

the accuracy of svc model with C = 280 is 0.9933184855233853

the accuracy of svc model with C = 290 is 0.994060876020787

the accuracy of svc model with C = 300 is 0.9955456570155902

the accuracy of svc model with C = 310 is 0.9955456570155902

the accuracy of svc model with C = 320 is 0.9955456570155902

the accuracy of svc model with C = 330 is 0.9955456570155902

the accuracy of svc model with C = 340 is 0.9955456570155902

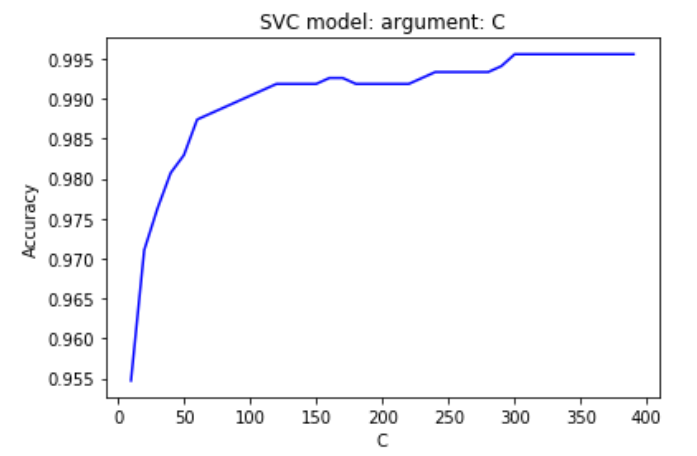
the accuracy of svc model with C = 350 is 0.9955456570155902

the accuracy of svc model with C = 360 is 0.9955456570155902

the accuracy of svc model with C = 370 is 0.9955456570155902

the accuracy of svc model with C = 380 is 0.9955456570155902

the accuracy of svc model with C = 390 is 0.9955456570155902



利用matplotlib可视化测试集中前10个灰度图像代码：

print(digits\_images.shape)

print(digits\_images)

print(digits\_images[0,0,3])

输出结果为：

(1797, 8, 8)

[[[ 0. 0. 5. ... 1. 0. 0.]

[ 0. 0. 13. ... 15. 5. 0.]

[ 0. 3. 15. ... 11. 8. 0.]

...

[ 0. 4. 11. ... 12. 7. 0.]

[ 0. 2. 14. ... 12. 0. 0.]

[ 0. 0. 6. ... 0. 0. 0.]]

[[ 0. 0. 0. ... 5. 0. 0.]

[ 0. 0. 0. ... 9. 0. 0.]

[ 0. 0. 3. ... 6. 0. 0.]

...

[ 0. 0. 1. ... 6. 0. 0.]

[ 0. 0. 1. ... 6. 0. 0.]

[ 0. 0. 0. ... 10. 0. 0.]]

[[ 0. 0. 0. ... 12. 0. 0.]

[ 0. 0. 3. ... 14. 0. 0.]

[ 0. 0. 8. ... 16. 0. 0.]

...

[ 0. 9. 16. ... 0. 0. 0.]

[ 0. 3. 13. ... 11. 5. 0.]

[ 0. 0. 0. ... 16. 9. 0.]]

...

[[ 0. 0. 1. ... 1. 0. 0.]

[ 0. 0. 13. ... 2. 1. 0.]

[ 0. 0. 16. ... 16. 5. 0.]

...

[ 0. 0. 16. ... 15. 0. 0.]

[ 0. 0. 15. ... 16. 0. 0.]

[ 0. 0. 2. ... 6. 0. 0.]]

[[ 0. 0. 2. ... 0. 0. 0.]

[ 0. 0. 14. ... 15. 1. 0.]

[ 0. 4. 16. ... 16. 7. 0.]

...

[ 0. 0. 0. ... 16. 2. 0.]

[ 0. 0. 4. ... 16. 2. 0.]

[ 0. 0. 5. ... 12. 0. 0.]]

[[ 0. 0. 10. ... 1. 0. 0.]

[ 0. 2. 16. ... 1. 0. 0.]

[ 0. 0. 15. ... 15. 0. 0.]

...

[ 0. 4. 16. ... 16. 6. 0.]

[ 0. 8. 16. ... 16. 8. 0.]

[ 0. 1. 8. ... 12. 1. 0.]]]

13.0

可以看出图像数据为1797\*8\*8的ndarray

8\*8的ndarray对应一个手写数字

print(digits\_images.max())

print(digits\_images.min())

可以看出数据的最小值为0，最大值为16

用以下代码可以画图

din=digits\_images

fig, ax = plt.subplots(nrows=2, ncols=5, sharex='all', sharey='all')#2行5列排布图片 不显示坐标轴

ax = ax.flatten()#将对应下标转换成一维，第i个子图即可用ax[i]访问

for i in range(10):

ax[i].imshow(din[i], cmap='Greys', interpolation='nearest')#画出灰度图像

ax[i].set\_title(digits\_target[i])#真实标签

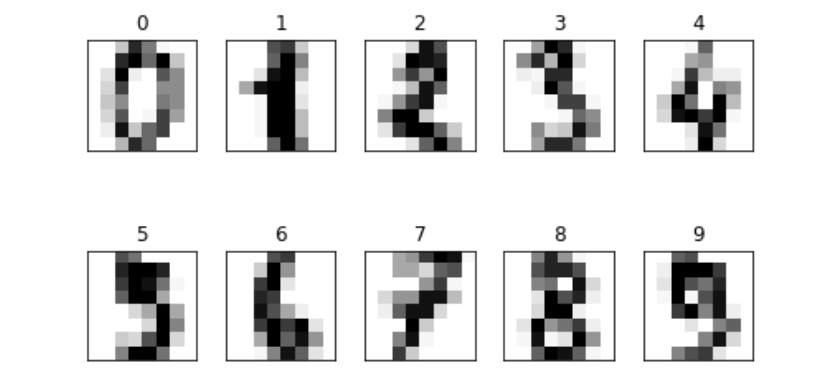
ax[0].set\_xticks([])#不显示坐标轴

ax[0].set\_yticks([])#

plt.tight\_layout()

plt.show()

显示灰度图像结果如下



1. 利用SVR预测患有疝病的马的死亡率，并分别利用网格搜索和随机搜索寻找最佳组合参数

SVR是支持向量回归机(Support Vactor Regerssion)。  支持向量机(SVM)是针对二分类问题提出的，而SVR（支持向量回归）是SVM（支持向量机）中的一个重要的应用分支。SVR回归与SVM分类的区别在于，SVR的样本点最终只有一类，SVR寻求的最优超平面不是SVM那样使两类或多类样本点划分地更泾渭分明(使到超平面最近的样本点的“距离”(margin)最大)，而是使所有的样本点离着超平面的总偏差最小。

用以下代码得到疝病马的数据集

frTrain = open('D://PUB128//horseColic//horseColicTraining.txt')#打开训练集文件

frTest = open('D://PUB128//horseColic//horseColicTest.txt') #打开测试集文件

#将训练集数据和对应标签存放到trainingSet和trainingLabels中

trainingSet = []; trainingLabels = []

for line in frTrain.readlines():

currLine = line.strip().split('\t')#用tab符号分割

lineArr =[]

for i in range(21):

lineArr.append(float(currLine[i]))

trainingSet.append(lineArr)

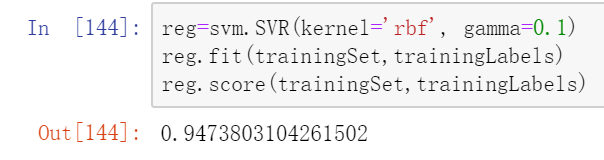
trainingLabels.append(float(currLine[21]))

在该数据集上建立SVR

reg=svm.SVR(kernel='rbf', gamma=0.1)

reg.fit(trainingSet,trainingLabels)

reg.score(trainingSet,trainingLabels)



得分为0.94738，比较理想

利用网格搜索寻找最佳参数：

from sklearn.model\_selection import GridSearchCV

from sklearn.svm import SVC

gammalist=[i\*1e-2 for i in range(25,100,25)]

tuned\_parameters=[{'kernel':['rbf'],'gamma':gammalist,'C':[i for i in range(10,100,10)]},{'kernel':['linear'],'C':[i for i in range(10,100,10)]}]

svr=svm.SVR()

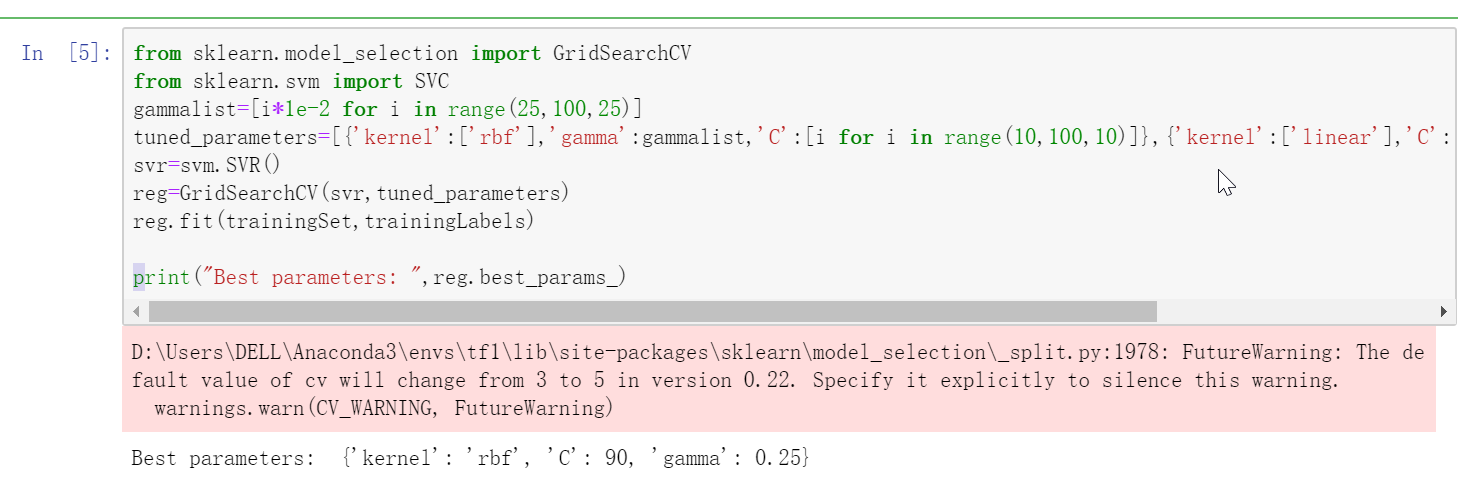
reg=GridSearchCV(svr,tuned\_parameters)

reg.fit(trainingSet,trainingLabels)

print("Best parameters: ",reg.best\_params\_)

这里用列表推导式生成参数枚举需要的列表

这个搜索过程需要运行很长很长时间，运行三四个小时后，可以得到运行结果为：



Best parameters: {'kernel': 'rbf', 'C': 90, 'gamma': 0.25}

用随机搜寻来找最优参数：

RandomizedSearchCV类实现了在参数空间上进行随机搜索，参数的取值是从某种概率分布中抽取的，这种搜索相比网格搜索，可以从候选的参数集合里选一些有代表性的来搜寻，可以让搜索量相对少些，在添加新的候选参数时也不会降低准确性和速度

import scipy

from sklearn.model\_selection import RandomizedSearchCV

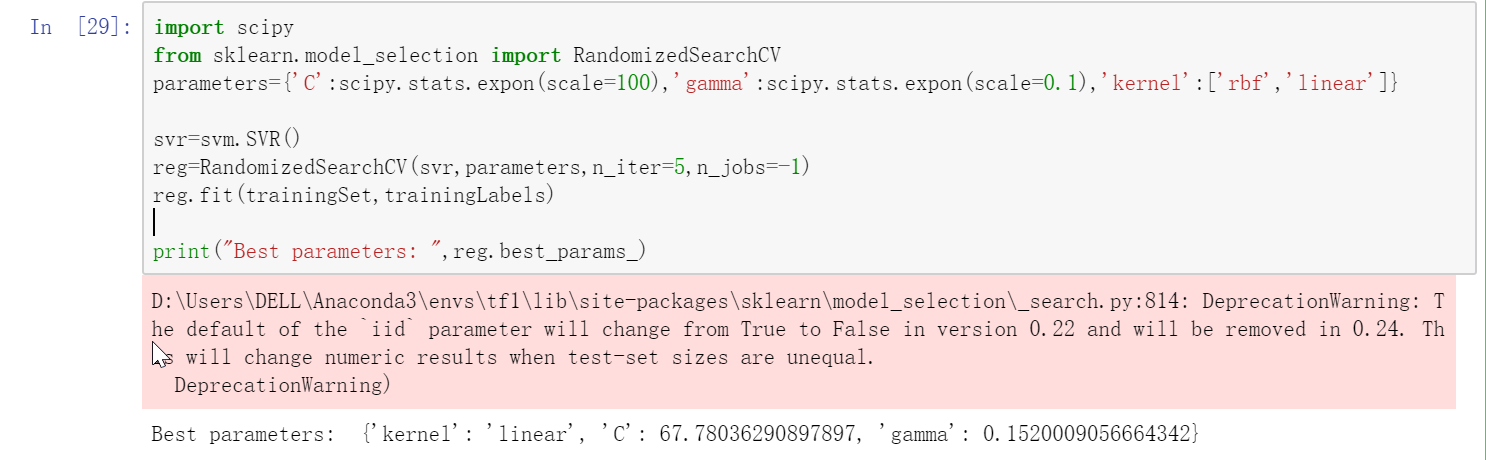
parameters={'C':scipy.stats.expon(scale=100),'gamma':scipy.stats.expon(scale=0.1),'kernel':['rbf','linear']}

svr=svm.SVR()

reg=RandomizedSearchCV(svr,parameters,n\_iter=5,n\_jobs=-1)

reg.fit(trainingSet,trainingLabels)

print("Best parameters: ",reg.best\_params\_)



Best parameters: {'kernel': 'linear', 'C': 67.78036290897897, 'gamma': 0.1520009056664342}

用找到的两组最优参数来训练和测试：

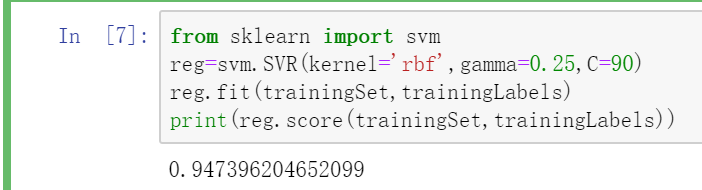
网格搜寻的最优参数

from sklearn import svm

reg=svm.SVR(kernel='rbf',gamma=0.25,C=90)

reg.fit(trainingSet,trainingLabels)

print(reg.score(trainingSet,trainingLabels))



正确率为94.73%，比之前效果更好

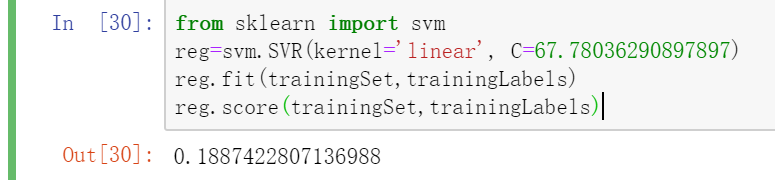
随机搜寻的最优参数

from sklearn import svm

reg=svm.SVR(kernel='linear', C=67.78036290897897)

reg.fit(trainingSet,trainingLabels)

reg.score(trainingSet,trainingLabels)



0.1887422807136988

因为随机搜寻过程中采用的迭代次数较少，只搜寻了5次，所以找到的参数并不是很优秀。

1. 实验总结：
2. 使用机器学习算法，从数据大量统计结果中挖掘信息时，常需要在对数据用算法进行处理前进行标准化，标准化可以用sklearn.preprocessing里的一些内置函数实现，比如MinMaxScaler，StandarScaler，RobustScaler等进行标准化处理。
3. sklearn.preprocessing不仅提供了对数据标准化预处理的方法，还有对数据正则化预处理的类Normalization，可以用这个类提供的方法实现正则化。正则化常用于处理向量数据，进行分类、聚类。
4. sklearn.datasets里面有很多常用的数据集，可以用里面内置的数据集方便地测试自己编写的机器学习算法，这里面的数据是一些比较适合进行机器学习的数据，可以方便对机器学习算法进行进一步深入的研究，对刚接触机器学习的人来说可以减少额外找数据花的时间和精力。
5. sklearn.model\_selection里的train\_test\_split函数可以方便地实行数据集的划分，可以直接用sklearn的这个根据自己机器学习算法的需求，划分出合适的训练集和测试集
6. sklearn.svm提供了SVC，可以方便地用svc=SVC()，svc.fit()直接快速构建出SVM分类器。而svc.score()则非常方便地评价出训练好的svc在某个测试集上的正确率
7. 利用sklearn里的Grid\_searchCV可以很方便地实现网格搜寻最优参数，程序会自动在候选参数集合里逐一枚举并测试，最后返回出最优的模型参数。进行网格搜寻的时候可以用循环产生等间距或指数等间距的参数列表，用参数列表进行网格搜索。相对于自己写代码来进行搜索最优参数，这种方法避免了写更多代码，避免了可能在多次测试中初始化中可能蕴含的bug，减少了代码量和编程时间，也更具有可靠性。这种方法也能处理多类参数类型，不是浮点数和整数的参数，如字符串型参数，非常地方便
8. plt.title()可以给画图指定图表名，plt.xlabel()可以指定x轴标注，plt.ylabel()可以指定y轴标注，可以让画出的图包含更多信息，更科学更美观。
9. 支持向量机是一种支持二分类的机器学习算法。如果样本有k类，参数X有n项，那么函数decision\_function(X)将返回一个k\*(k-1)/2列的ndarray，每一个ndarray对应于分割两个类别的超平面，decision\_function(X)[:,i]就返回了第i个超平面的函数距离，decision\_function(X)[j,i]表示第j个点到第i个超平面的函数距离(有正有负)，根据这个函数距离的正负性，可以判断出这个点处于超平面的哪一侧
10. matplotlib.pyplot.contour()用于绘制函数的轮廓图，等高线图，可以用于二元函数图像的可视化，meshgrid()可以生成用来可视化二元函数的矩阵，表示这个二元函数的一些采样点。
11. flatten可以将操纵子图的下标转换成一维下标，方便对子图的编号与处理
12. imshow(din[i], cmap='Greys', interpolation='nearest')通过参数cmap=’Greys’可以画出对应于一个二维ndarraydin[i]的灰度图像，可以很方便地处理图像可视化
13. line.strip()可以删除一行头尾的指定字符，line.strip().split('\t')可以将文件里一行的头尾多余字符删掉，然后将剩下的用制表符tab分隔的数据转换成列表数据
14. 在数据集大的时候，用网格搜寻svr最优参数时可能需要花很长时间，需要耐心等待，需要关掉多余的应用来给机器学习算法提供更多的内存和CPU使用率来运行。如果运行很久还是得不到结果，需要仔细认真分析计算复杂度，如果确实运行得不到结果，那么需要用任务管理器或者Jupyter管理running的笔记本页中断笔记本执行，然后调整网格搜寻的工作量和搜寻的参数列表。在实际操作中用含有十几组参数的参数列表进行网格搜索处理马的疝病数据时，确实需要运行很长很长时间，原因可能一方面是马疝病数据集比较大，另一方面是网格搜索时包装起方法和类时需要额外的时间开销
15. 可以用列表推导式来方便地给出网格搜索枚举参数的列表
16. SVR为支持向量回归，其与支持向量机的基本思想差不多，但在最优化目标上不同。支持向量机的最优化目标为margin(支持向量的间隔)最大，支持向量回归的优化目标为让所有点到超平面的距离平均值最小。
17. 在参数调优进行网格搜索或随机搜索时，可以指定参数n\_jobs=-1，这种情况下，如果计算机有多个CPU内核，这样多个CPU内核会同时进行并行搜索最优参数。利用计算机多核的并行性，可以加快搜索最优参数的速度
18. 一开始用列表套集合的方式进行RandomizedSearchCV，也就是类似para=[{'C':scipy.stats.expon(scale=100),'gamma':scipy.stats.expon(scale=0.1),'kernel':['rbf','linear']}]的与GridSearchCV同样的传参数表的方式，但这样运行就会报错。显示了list没有value的错误。

这是因为虽然网格搜索和随机搜索都是同一个库里的搜索方法，但是其需要的参数是不同格式的。GridSearchCV需要的是一个集合列表，将集合列表进行打包成一个新的集合来逐一搜索。而RandomizedSearchCV需要的直接是一个集合，因为随机搜索需要对候选参数进行采样，提供的参数列表需要是可以采样的。如果某个参数的候选集合是列表，那么直接就是均匀采样。如果用scipy.stats库，那么可以对参数按不同的分布来采样，如指数分布、高斯分布、均匀分布，或者可以用自定义的随机分布来采样，不过需要自己定义rvs采样函数。正确的参数形式为：

parameters={'C':scipy.stats.expon(scale=100),'gamma':scipy.stats.expon(scale=0.1),'kernel':['rbf','linear']}

除此之外，还可以指定随机搜索的检索次数，参数n\_iter可以指定出随机搜索查找几次。

1. SVR在核函数为linear的时候，似乎要运行很长时间，在参数搜索的过程中会耗费很久。而和函数为高斯核时，似乎运行相对快一些。单独运行核函数为linear和核函数为rbf的两个SVR可以明显发现linear对应的SVR会比rbf对应的SVR慢很多
2. .grid\_scores\_可以直接返回网格搜索后找到的最优得分
3. scipy.stats包含很多常用分布如expon指数分布，randint()随机数等，可以用于RandomizedSearchCV的采样