**机器学习课程报告**

**SVM&SVR**

**一、任务描述**

1.1 基于 MNIST 数据集，设计和实现分类器，包括：线性和非线性 SVM。

1.2 设计一个回归系统来预测房价。回归算法应包含 SVR。

**二、实验准备**

**（一）导入必要的包**

1.1 SVM导入的包

1. **import** matplotlib.pyplot **as** plt
2. **from** sklearn **import** svm
3. **from** keras.datasets **import** mnist
4. **from** sklearn.metrics **import** classification\_report
5. **from** sklearn.model\_selection **import** GridSearchCV

1.2 SVR导入的包

1. **from** sklearn.datasets **import** load\_boston
2. **import** pandas **as** pd
3. **import** matplotlib.pyplot **as** plt
4. **from** sklearn.svm **import** SVR
5. **import** seaborn **as** sns
6. **from** sklearn.metrics **import** r2\_score
7. **from** sklearn.preprocessing **import** StandardScaler

**三、数据集描述**

**（一）SVM使用的MNIST 数据集**

1.导入MNIST 数据集

1. (train\_images, train\_labels), (test\_images, test\_labels) = mnist.load\_data()

一共统计了来自250个不同的人手写数字图片，其中50%是高中生，50%来自人口普查局的工作人员。该数据集的收集目的是希望通过算法，实现对手写数字的识别。

注意：前5000个比后5000个要规整，这是因为前5000个数据来自于美国人口普查局的员工，而后5000个来自于大学生。

2. MNIST 数据集特征

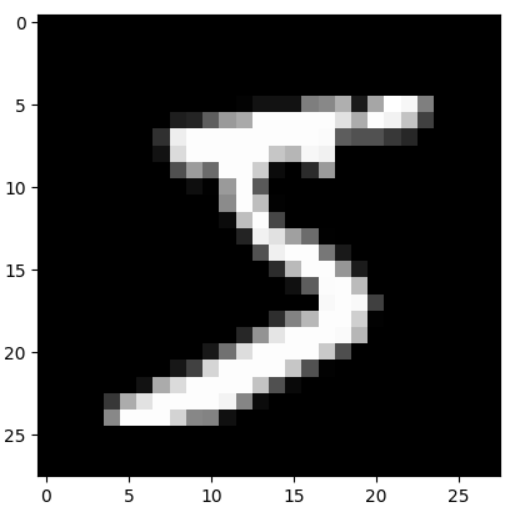
1. **print**(train\_images.shape,test\_images.shape)



训练集train一共包含了 60000 张图像和标签，而测试集一共包含了 10000 张图像和标签。idx3表示3维，ubyte表示是以字节的形式进行存储的，t10k表示10000张测试图片（test10000）。每张图片是一个28\*28像素点的0 ~ 9的手写数字图片，图像像素值为0 ~ 255，越大该点越白。

3. MNIST 数据集内容

1. plt.imshow(train\_images[0],cmap=plt.get\_cmap('gray'))



展示训练集第一张图片

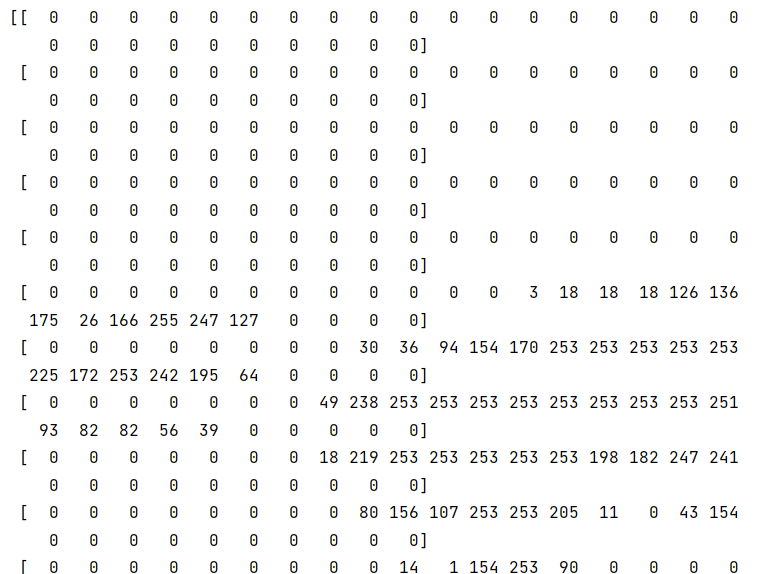
1. **print**(train\_labels[0])



标签为5

1. **print**(train\_images[0])

以象素矩阵显示。



第0 ~ 3字节，是32位整型数据，取值为0x00000801(2049)，即用幻数2049记录文件数据格式，这里的格式为文本格式。

第4～7个字节，是32位整型数据，取值为60000（训练集时）或10000（测试集时），用来记录标签数据的个数；

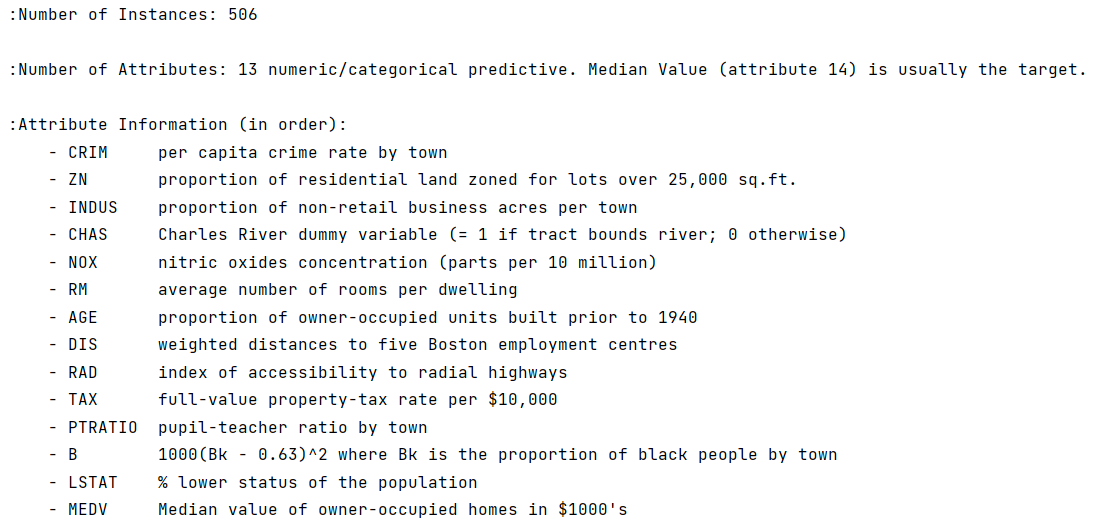
第8个字节 ~ ），是一个无符号型的数，取值为对应0~9 之间的标签数字，用来记录样本的标签。

**（二）SVR使用的Boston 房价数据集**1.导入Boston 房价数据集

1. house = load\_boston()

2. Boston 房价数据集特征

1. **print**(house.DESCR)



特征 说明

CRIM 城镇人均犯罪率

ZN 大于25,000平方英尺的地块划分为住宅用地的比例

INDUS 每个城镇非零售业务的比例

CHAS 查尔斯河虚拟变量（如果 = 1则为河; =0则不为河）

NOX 一氧化氮浓度（每千万）

RM 每间住宅的平均房间数

AGE 自住房屋是在1940年之前建造的比例

DIS 到加州五个就业中心的加权距离

RAD 对径向高速公路的可达性指数

TAX 每10,000美元的全价物业税

PTRATIO 城镇的学生与教师比例

B 1000（Bk-0.63）^ 2其中Bk是城镇的黑人的比例

LSTAT 低社会阶层人口比例％

MEDV 以1000美元为单位的自住房屋的中位数价格

3. Boston 房价数据集内容

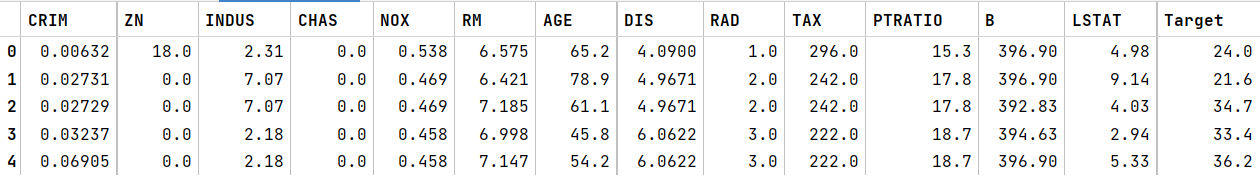
1. df.shape



特征数量： 13个相关属性（即13个指标变量），1个目标变量（房价）。

1. df.head()

展示前五行数据



**四、数据预处理**

**（一）SVM--MNIST 数据集**

1.MNIST 数据集形状变换

1. train\_images = train\_images.reshape(60000, 28 \* 28).astype('float32') / 255
2. test\_images = test\_images.reshape(10000, 28 \* 28).astype('float32') / 255

astype方法：通用函数，可以用于把dataframe中的任何列转换成其他类型。需要将MNIST 数据集变换为一个形状为(60000, 28 \* 28)，取值范围为 0~1 的数组。

2.MNIST 数据集选取部分数据

1. x\_train, y\_train = train\_images[:7500], train\_labels[:7500]
2. x\_test, y\_test = test\_images[:2500], test\_labels[:2500]

选取7500个训练数据，2500个测试数据，

3.MNIST 数据集PCA降维

1. **from** sklearn.decomposition **import** PCA
3. pca = PCA(n\_components=100)
4. pca\_x\_train = pca.fit\_transform(x\_train)
5. pca\_y\_train = y\_train
6. pca\_x\_test = pca.transform(x\_test)
7. pca\_y\_test = y\_test

主成分分析，提取数据的主要成分，剔除数据中相对次要的成分，换句话说PCA的目标是降维，就是剔除数据次要成分的维度。剔除数据中相对次要的成分，一次来压缩数据量的大小、降低数据变量的复杂度。

高维数据中变量之间的关系是不可见的。一个合理的方法，在降低维度的同时，尽量的减少数据信息的损失，这样对于数据的处理是可以接受的。

1. **print**(pca\_x\_train.shape,pca\_x\_test.shape)



把原来的像素点压缩为100个维度

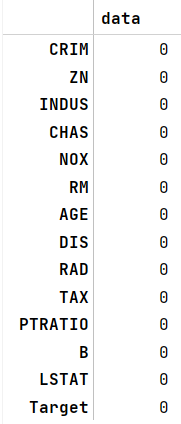
**（二）SVR--Boston 房价数据集**

1.Boston 房价数据集转化

1. x = house.data
2. y = house.target
3. df = pd.DataFrame(x, columns=house.feature\_names)
4. df['Target'] = pd.DataFrame(y, columns=['Target'])

2.检查Boston 房价数据集是否缺项

1. df.isnull().sum()



isnull()函数，它可以⽤来判断缺失值。isnull().sum(): 生成每一列缺失值的总数

3.Boston 房价数据集画出热力图

1. plt.figure(figsize=(15,15))
2. p=sns.heatmap(df.corr(), annot=True, square=True)



根据热力图中不同方块颜色对应的相关系数的大小，可以判断出变量之间相关性的大小。两个变量之间相关系数的计算公式为:



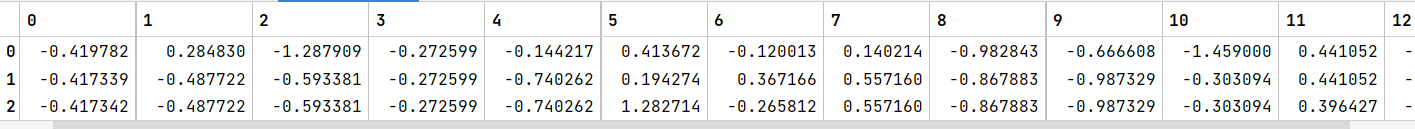
该相关系数只能度量出变量之间的线性相关关系；也就是说，相关系数越高，则变量间的线性相关程度越高。对于相关系数小的两个变量，只能说明变量间的线性相关程度弱，但不能说明变量之间不存在其它的相关关系，如曲线关系等。

4.Boston 房价数据集数据标准化

1. ss = StandardScaler()
2. x = ss.fit\_transform(x)

标准化：将数据变换为均值为0，标准差为1的分布切记，并非一定是正态的；

1. x[:3]



这是标准化后的前三行数据。

数据的标准化（normalization）是将数据按比例缩放，使之落入一个小的特定区间。在某些比较和评价的指标处理中经常会用到，去除数据的单位限制，将其转化为无量纲的纯数值，便于不同单位或量级的指标能够进行比较和加权。

5.Boston 房价数据集切分

1. **from** sklearn.model\_selection **import** train\_test\_split
2. x\_train,x\_test,y\_train,y\_test = train\_test\_split(x,y,test\_size=0.3)

train\_test\_split函数可按照用户设定的比例，随机将样本集合划分为训练集 和测试集，并返回划分好的训练集和测试集数据。

训练集（Test Data）：用户构建模型

测试集（Test Data）:用于检验模型构建，此数据只模型检验时使用，用于评估模型的准确率。绝对不允许用于模型构建过程，否者会导致过度拟合

**五、算法描述**

**（一）SVM概述**

支持向量机是在分类与回归分析中分析数据的监督式学习模型与相关的学习算法。给定一组训练实例，每个训练实例被标记为属于两个类别中的一个或另一个，SVM训练算法创建一个将新的实例分配给两个类别之一的模型，使其成为非概率二元线性分类器。SVM模型是将实例表示为空间中的点，这样映射就使得单独类别的实例被尽可能宽的明显的间隔分开。然后，将新的实例映射到同一空间，并基于它们落在间隔的哪一侧来预测所属类别。

**（二）SVM算法**

1.线性SVM

我们考虑右边形式的n点测试集：

其中是1或者−1，表明点所属的类。中每个都是一个p维实向量。我们要求将=1的点集与=-1的点集分开的 “最大间隔超平面”，使得超平面与最近的点之间的距离最大化。

任何超平面都可以写作满足下面方程的点集



其中（不必是归一化的）是该法向量。参数决定从原点沿法向量到超平面的偏移量。

2.硬间隔

如果这些训练数据是线性可分的，可以选择分离两类数据的两个平行超平面，使得它们之间的距离尽可能大。在这两个超平面范围内的区域称为“间隔”，最大间隔超平面是位于它们正中间的超平面。

这些超平面可以由方程：



或是



来表示。

通过几何不难得到这两个超平面之间的距离是，因此要使两平面间的距离最大，我们需要最小化。同时为了使得样本数据点都在超平面的间隔区以外，我们需要保证对于所有的i满足其中的一个条件：



或是



这些约束表明每个数据点都必须位于间隔的正确一侧。

这两个式子可以写作：



可以用这个式子一起来得到优化问题：

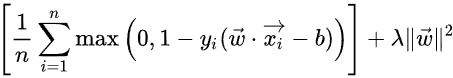


3.软间隔

为了将SVM扩展到数据线性不可分的情况，我们引入铰链损失函数，



当约束条件 (1) 满足时（也就是如果位于边距的正确一侧）此函数为零。对于间隔的错误一侧的数据，该函数的值与距间隔的距离成正比。 然后我们希望最小化



其中参数用来权衡增加间隔大小与确保位于间隔的正确一侧之间的关系。因此，对于足够小的值，如果输入数据是可以线性分类的，则软间隔SVM与硬间隔SVM将表现相同，但即使不可线性分类，仍能学习出可行的分类规则。

**（三）核函数概述**

线性核函数，就是简单原始空间中的内积。优点： 简单、运算效率高；缺点： 对线性不可分的数据集没有很好的效果。

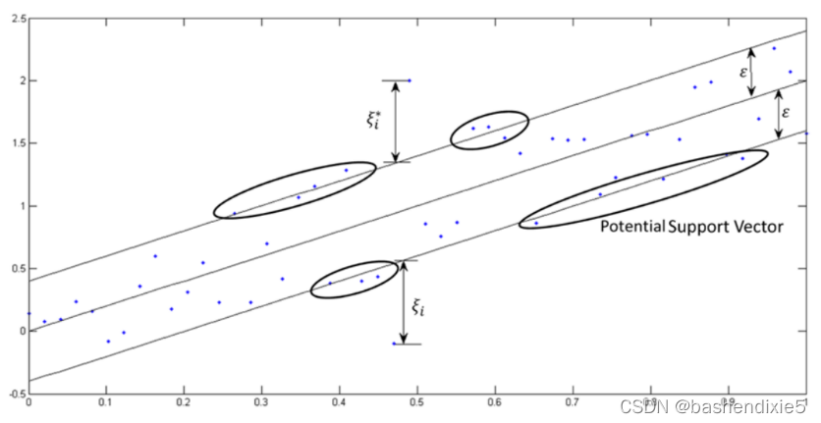
多项式核函数，可根据R和d的取值不同，而有不同的计算式。优点： 可以拟合出复杂的分割超平面；缺点： 有三个参数，调参困难，且当n过大时，模型拟合时间会很长 ；

高斯核函数，可根据实际需要灵活选取参数σ，甚至还可以将原始维度空间映射到无穷维度空间。不过，如果σ取值很大，会导致高次特征上的权重衰减快；如果σ取值很小，其好处是可以将任意的数据映射成为线性可分，但容易造成过拟合现象。 高斯核函数是非常经典，也是使用最广泛的核函数之一。

**（四）SVR**

SVR 模型在回归线周围设置了一个阈值ε，这样ε内的所有数据点都不会因其误差而受到惩罚。

所以说SVM 对 SVR 的泛化是通过在函数周围引入一个ε不敏感区域来完成的，称为ε-管子。



SVR 模型设置了阈值误差容限ε围绕回归线，使得所有数据点在ε不会因为他们的错误而受到惩罚。

支持向量机通过将二元分类问题表述为凸优化问题来解决二元分类问题。优化问题需要找到分离超平面的最大边距，同时正确分类尽可能多的训练点。SVM 用支持向量表示这个最优超平面。SVM 的稀疏解决方案和良好的泛化能力有助于适应回归问题。

通过首先定义一个凸ε，将 SVR 表述为一个优化问题-最小化不敏感损失函数并找到包含大多数训练实例的最扁平管。因此，多目标函数由损失函数和管的几何特性构成。然后，使用适当的数值优化算法求解具有唯一解的凸优化。超平面用支持向量表示，支持向量是位于管边界之外的训练样本。

和 SVM 一样，SVR 中的支持向量是影响ε管子形状的最有影响力的实例，并且假设训练和测试数据是独立同分布(iid)，取自相同的固定但未知的概率分布函数在监督学习环境中。

**六、sklearn中的SVM和SVR**

**（一）sklearn中的SVM**

1. sklearn.svm.SVR（kernel ='rbf'，degree = 3，gamma ='auto\_deprecated'，coef0 = 0.0，tol = 0.001，C = 1.0，epsilon = 0.1，shrinking = True，cache\_size = 200，verbose = False，max\_iter = -1 ）

1.kernel：

string，optional（default ='rbf'）指定要在算法中使用的内核类型。

2.degree：

int，可选（默认= 3）多项式核函数的次数（'poly'）。被所有其他内核忽略。

3.gamma：

float，optional（默认='auto'）

4.C：

float，可选（默认= 1.0）错误术语的惩罚参数C.

5.epsilon：

float，optional（默认值= 0.1）

Epsilon在epsilon-SVR模型中。它指定了epsilon-tube，其中训练损失函数中没有惩罚与在实际值的距离epsilon内预测的点。

**（二）sklearn中的SVR**

1. sklearn.svm.SVR(\*, kernel='rbf', degree=3, gamma='scale', coef0=0.0, tol=0.001, C=1.0, epsilon=0.1, shrinking=True, cache\_size=200, verbose=False, max\_iter=- 1)

1. kernel

用来选择映射到高维线性可分的核函数

2.degree

多项式核函数的维度，

3. gamma

‘rbf’, ‘poly’ 和‘sigmoid’ 核函数的系数

4. C

正则化系数，正则化的强度与C的大小成反比，且必须为正。C值越大，对模型的惩罚越高，泛化能力越弱（过拟合）。

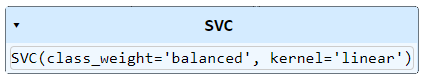
5. epsilon

用来定义模型对于错误分类的容忍度，即错误分类而不受到惩罚。ϵ 的值越大，模型允许错误分类的容忍度越高，反之，容忍度越小。

**七.实验过程**

**（一）线性SVM**

1. linear\_model=svm.SVC(kernel='linear',class\_weight='balanced')
2. linear\_model.fit(pca\_x\_train,pca\_y\_train)



让转换函数，则得到线性核函数，则两个向量的点积为：k(x,{x}')=x^T{x}'

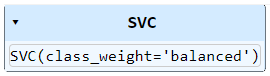
线性核函数的特征空间F的维度与输入控件\chi，的维度一样，每个向量的特征数也一样(x=(x_1,x_2,....)，x_1叫特征，x代表一个样本）。

当不需要在特征空间进行运算时，可以用线性核函数。如原始数据已经是高维的、可比较的，并且在输入空间线性可分。

**（二）非线性SVM**

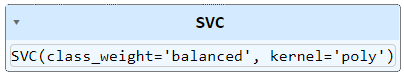
1.SVM rbf

1. rbf\_model = svm.SVC(kernel='rbf', class\_weight='balanced')
2. rbf\_model.fit(pca\_x\_train, pca\_y\_train)



2.SVM poly

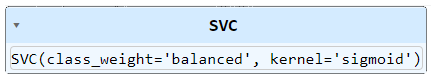
1. poly\_model = svm.SVC(kernel='poly', class\_weight='balanced')
2. poly\_model.fit(pca\_x\_train, pca\_y\_train)



多项式核(poly)：

3.SVM sigmoid

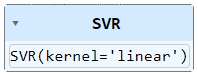
1. sigmoid\_model = svm.SVC(kernel='sigmoid', class\_weight='balanced')
2. sigmoid\_model.fit(pca\_x\_train, pca\_y\_train)



**（三）SVR**

1.SVR linear

1. linear\_model=SVR(kernel='linear')
2. linear\_model.fit(x,y)



2.SVR rbf

1. rbf\_model=SVR(kernel='rbf')
2. rbf\_model.fit(x,y)



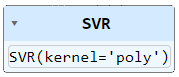
高斯核k(x,x')=exp(-\frac{1}{2}(x-x')^T\Sigma ^{-1}(x-x')),

\Sigma是观测中每个特征的协方差，p维矩阵。当\Sigma是对角线矩阵时，可以写为：

k(x,x')=exp(-\frac{1}{2}\sum_{j=1}^p\frac{(x_j-x'_j)^2}{\sigma _j^2})

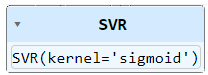
3.SVR poly

1. poly\_model=SVR(kernel='poly')
2. poly\_model.fit(x,y)



4.SVR sigmoid

1. sigmoid\_model=SVR(kernel='sigmoid')
2. sigmoid\_model.fit(x,y)



sigmoid函数也叫Logistic函数，用于隐层神经元输出，取值范围为(0,1)，它可以将一个实数映射到(0,1)的区间，可以用来做二分类。在特征相差比较复杂或是相差不是特别大时效果比较好。Sigmoid作为激活函数有以下优缺点：

优点：平滑、易于求导

缺点：激活函数计算量大，反向传播求误差梯度时，求导涉及除法；反向传播时，很容易就会出现梯度消失的情况，从而无法完成深层网络的训练。

Sigmoid函数由下列公式定义

在这里插入图片描述

其对x的导数可以用自身表示：

在这里插入图片描述

**八、实验结果分析**

**（一）评价指标**

1.SVM评价指标

（1）准确率（Accuracy）

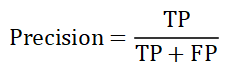
所有预测正确的（包括正类和负类）占总的的比例。



（2）精确率（Precision）

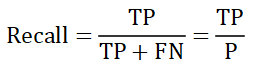
查准率。即正确预测为正类的占全部预测为正类的的比例。

精确率是针对我们预测结果而言的，它表示的是预测为正类的样本中有多少是真正的正类样本。那么预测为正类就有两种可能了，一种就是把正类预测为正类(TP)，另一种就是把负类预测为正类(FP)。



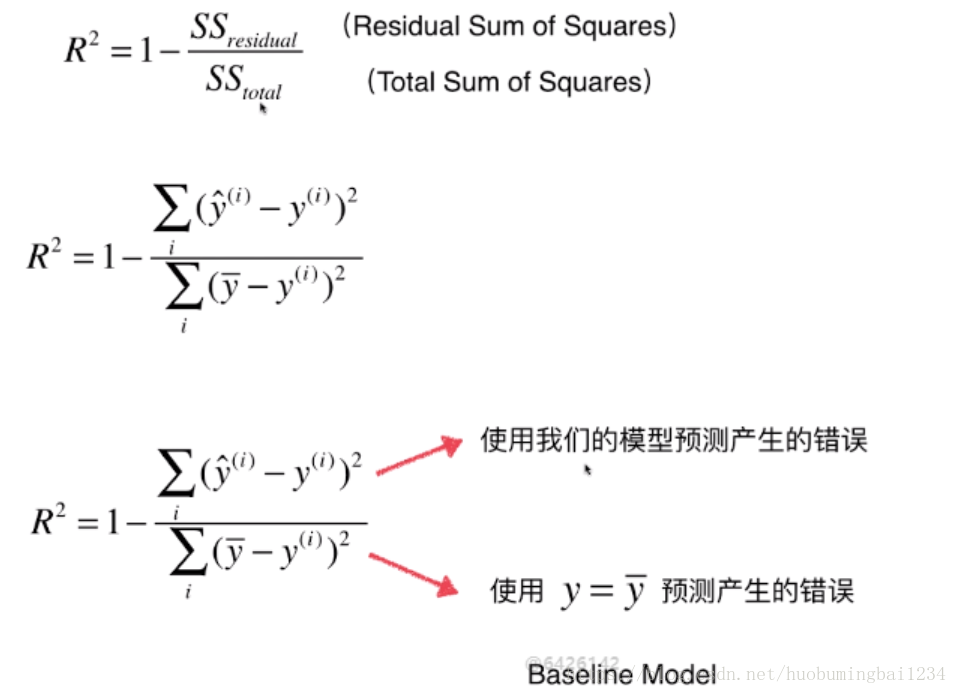
（3）召回率（Recall）

正确预测为正类的占全部实际为正类的的比例。召回率是针对我们原始样本而言的，它表示的是全体样本中的所有正类样本有多少被预测正确了。



2.SVR评价指标

（1）R2



分子是我们的模型预测产生的错误，分母是使用y等于y的均值这个模型所产生的错误。

自己的模型预测产生的错误/基础模型预测生产的错误，表示自己的模型没有拟合住的数据，因此R2可以理解为，自己的模型拟合住的数据。

R2 <= 1。

R2越大越好，当自己的预测模型不犯任何错误时：R2 = 1。

当我们的模型等于基准模型时：R2 = 0。

如果R2 < 0，说明学习到的模型还不如基准模型。

（4）混淆矩阵

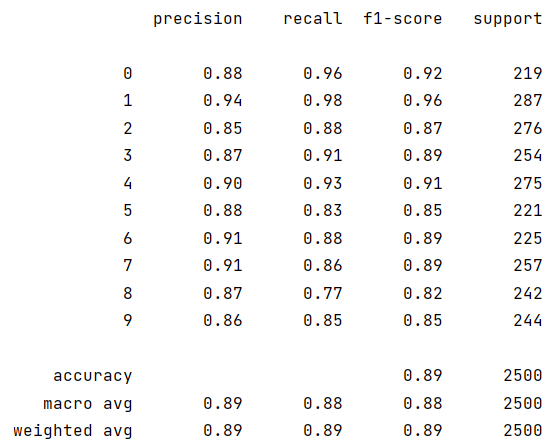
混淆矩阵是对分类问题的预测结果的总结。使用计数值汇总正确和不正确预测的数量，并按照每个类别进行细分，这是混淆矩阵的关键所在。混淆矩阵显示了额分类模型在进行预测时会对那一部分产生混淆，不仅能了解分类模型所犯的错误，更重要的是可以了解发生错误的类型。这是这种对结果的分解客服了仅使用分类准确率所带来的局限性。

**（二）定量评价结果**

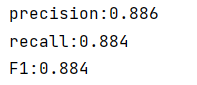
1.线性SVM

（1）准确率，召回率，F1

1. linear\_predictions = linear\_model.predict(pca\_x\_test)
2. **print**(classification\_report(pca\_y\_test, linear\_predictions))



1. **print**('precision:%.3f' % precision\_score(y\_true=y\_test, y\_pred=linear\_predictions,average= 'macro'))
2. **print**('recall:%.3f' % recall\_score(y\_true=y\_test, y\_pred=linear\_predictions,average= 'macro'))
3. **print**('F1:%.3f' % f1\_score(y\_true=y\_test, y\_pred=linear\_predictions,average= 'macro'))



准确率（Accuracy）就是所有的预测正确（正类负类）的占总的比重。

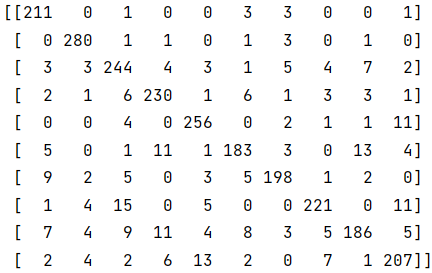
精确率（Precision），查准率。即正确预测为正的占全部预测为正的比例。

召回率（Recall），查全率。即正确预测为正的占全部实际为正的比例。

F1值（H-mean值）。F1值为算数平均数除以几何平均数，且越大越好，将Precision和Recall的上述公式带入会发现，当F1值小时，True Positive相对增加，而false相对减少，即Precision和Recall都相对增加，即F1对Precision和Recall都进行了加权。

（2）混淆矩阵

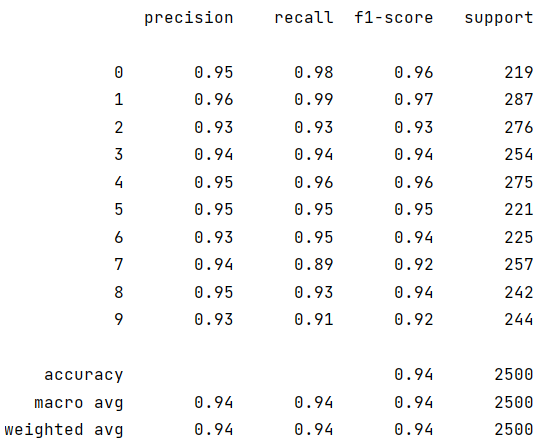
1. confmat = confusion\_matrix(y\_true=pca\_y\_test, y\_pred=linear\_predictions) *# 输出混淆矩阵*
2. **print**(confmat)



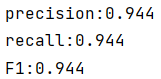
2.非线性SVM

（1）SVM rbf

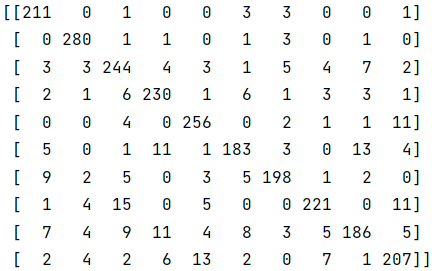
1. rbf\_predictions = rbf\_model.predict(pca\_x\_test)
2. **print**(classification\_report(pca\_y\_test, rbf\_predictions))



1. **print**('precision:%.3f' % precision\_score(y\_true=y\_test, y\_pred=rbf\_predictions,average= 'macro'))
2. **print**('recall:%.3f' % recall\_score(y\_true=y\_test, y\_pred=rbf\_predictions,average= 'macro'))
3. **print**('F1:%.3f' % f1\_score(y\_true=y\_test, y\_pred=rbf\_predictions,average= 'macro'))



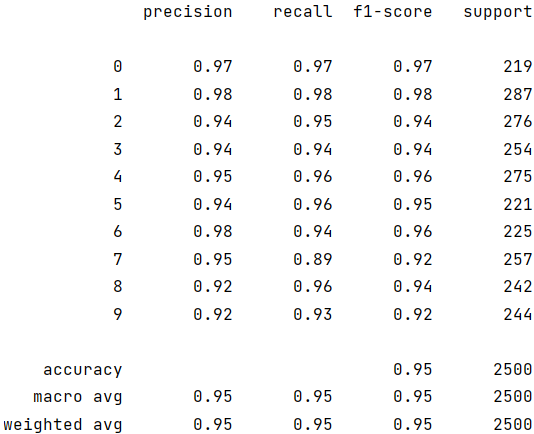
1. rbf\_confmat = confusion\_matrix(y\_true=pca\_y\_test, y\_pred=rbf\_predictions) *# 输出混淆矩阵*
2. **print**(rbf\_confmat)



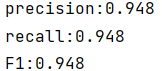
混淆矩阵就是分别统计分类模型归错类，归对类的观测值个数，然后把结果放在一个表里展示出来。

（2）SVM poly

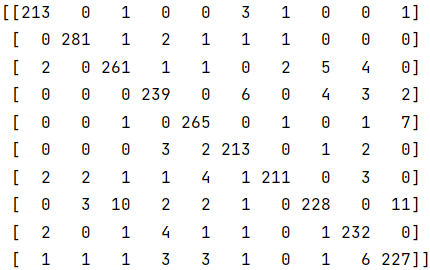
1. poly\_predictions = poly\_model.predict(pca\_x\_test)
2. **print**(classification\_report(pca\_y\_test, poly\_predictions))



1. **print**('precision:%.3f' % precision\_score(y\_true=y\_test, y\_pred=poly\_predictions,average= 'macro'))
2. **print**('recall:%.3f' % recall\_score(y\_true=y\_test, y\_pred=poly\_predictions,average= 'macro'))
3. **print**('F1:%.3f' % f1\_score(y\_true=y\_test, y\_pred=poly\_predictions,average= 'macro'))

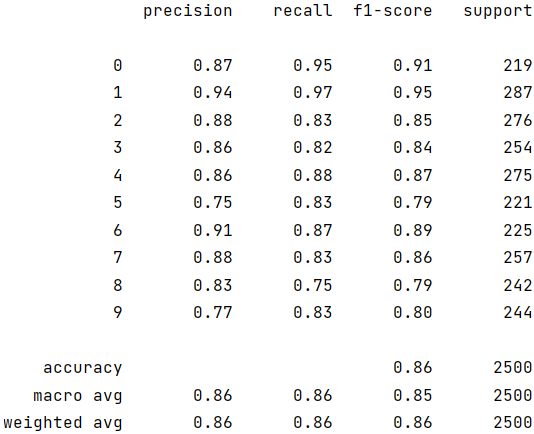


1. poly\_confmat = confusion\_matrix(y\_true=pca\_y\_test, y\_pred=poly\_predictions) *# 输出混淆矩阵*
2. **print**(poly\_confmat)

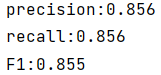


（3）SVM sigmoid

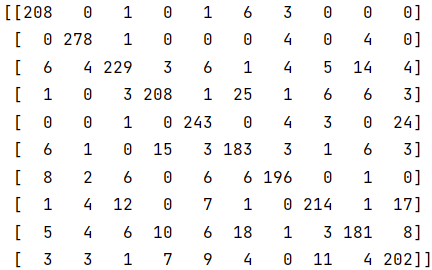
1. sigmoid\_predictions = sigmoid\_model.predict(pca\_x\_test)
2. **print**(classification\_report(pca\_y\_test, sigmoid\_predictions))



1. **print**('precision:%.3f' % precision\_score(y\_true=y\_test, y\_pred=sigmoid\_predictions,average= 'macro'))
2. **print**('recall:%.3f' % recall\_score(y\_true=y\_test, y\_pred=sigmoid\_predictions,average= 'macro'))
3. **print**('F1:%.3f' % f1\_score(y\_true=y\_test, y\_pred=sigmoid\_predictions,average= 'macro'))



1. sigmoid\_confmat = confusion\_matrix(y\_true=pca\_y\_test, y\_pred=sigmoid\_predictions) *# 输出混淆矩阵*
2. **print**(sigmoid\_confmat)



3.SVR

（1）SVR linear

1. linear\_predictions = linear\_model.predict(x\_test)
2. **print**('linear R-squared =',r2\_score(y\_test,linear\_predictions))



（2）SVR rbf

1. rbf\_predictions = rbf\_model.predict(x\_test)
2. **print**('rbf R-squared =',r2\_score(y\_test,rbf\_predictions))



（3）SVR poly

1. poly\_predictions = poly\_model.predict(x\_test)
2. **print**('poly R-squared =',r2\_score(y\_test,poly\_predictions))



（4）SVR sigmoid

1. sigmoid\_predictions = sigmoid\_model.predict(x\_test)
2. **print**('sigmoid R-squared =',r2\_score(y\_test,sigmoid\_predictions))

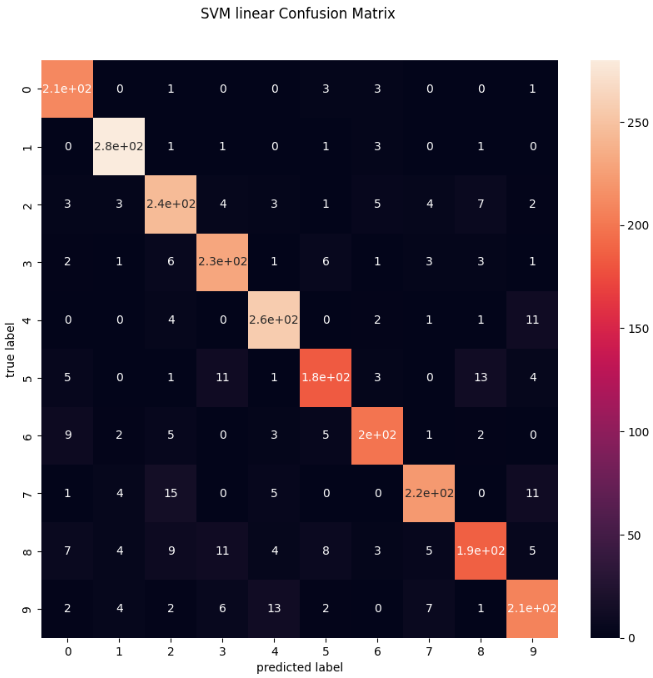


**（三）可视化结果**

1.线性SVM

1. linear\_fig, linear\_ax = plt.subplots(figsize=(10, 9))
3. sns.heatmap(linear\_confmat,annot=True,ax=linear\_ax) *#plot heatmap*

6. linear\_ax.set\_title('SVM linear Confusion Matrix **\n\n**')
7. linear\_ax.set\_xlabel('predict')
8. linear\_ax.set\_ylabel('true') *#*
10. plt.xlabel('predicted label')
11. plt.ylabel('true label')
12. plt.show()

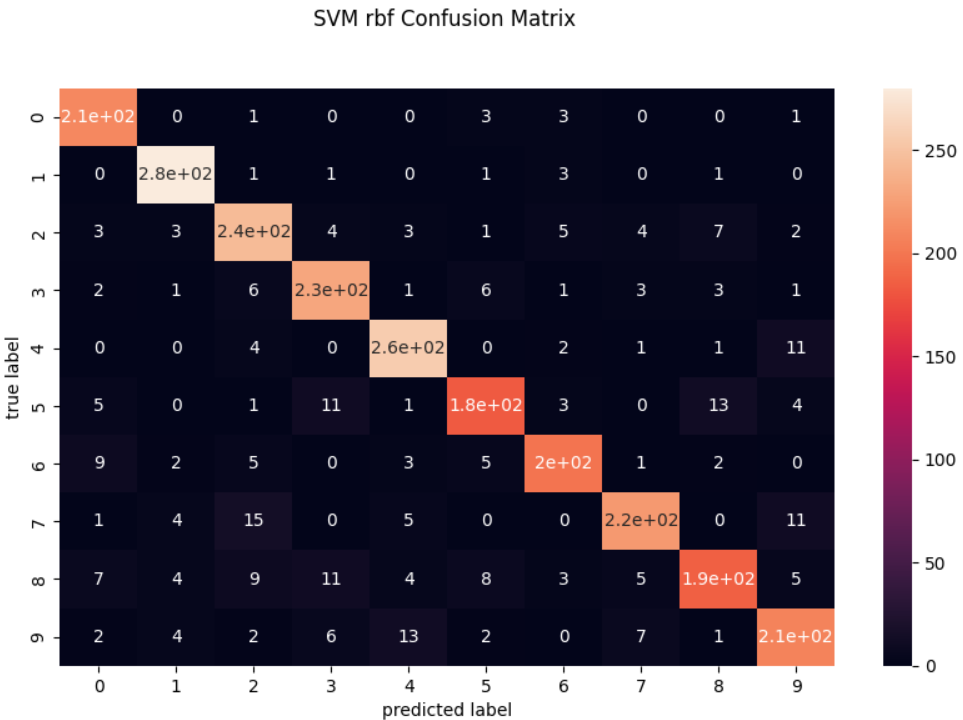


2.非线性SVM

（1）SVM rbf

1. rbf\_fig, rbf\_ax = plt.subplots(figsize=(10, 6))
3. sns.heatmap(rbf\_confmat,annot=True,ax=rbf\_ax) *#plot heatmap*

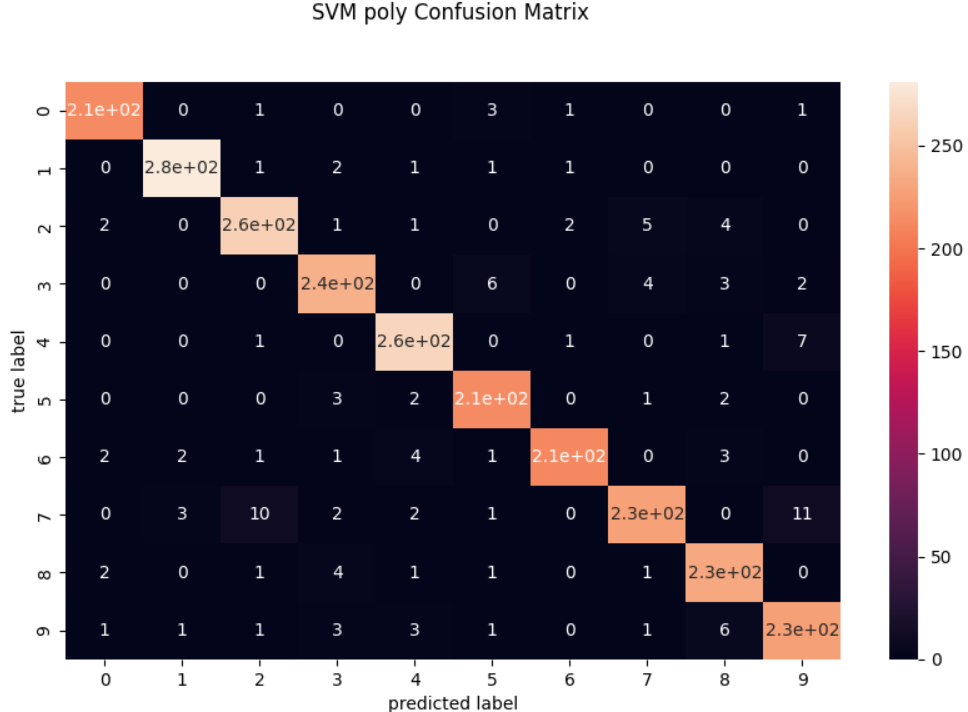
6. rbf\_ax.set\_title('SVM rbf Confusion Matrix **\n\n**')
7. rbf\_ax.set\_xlabel('predict')
8. rbf\_ax.set\_ylabel('true') *#*
10. plt.xlabel('predicted label')
11. plt.ylabel('true label')
12. plt.show()



（2）SVM poly

1. poly\_fig, poly\_ax = plt.subplots(figsize=(10, 6))
3. sns.heatmap(poly\_confmat,annot=True,ax=poly\_ax) *#plot heatmap*

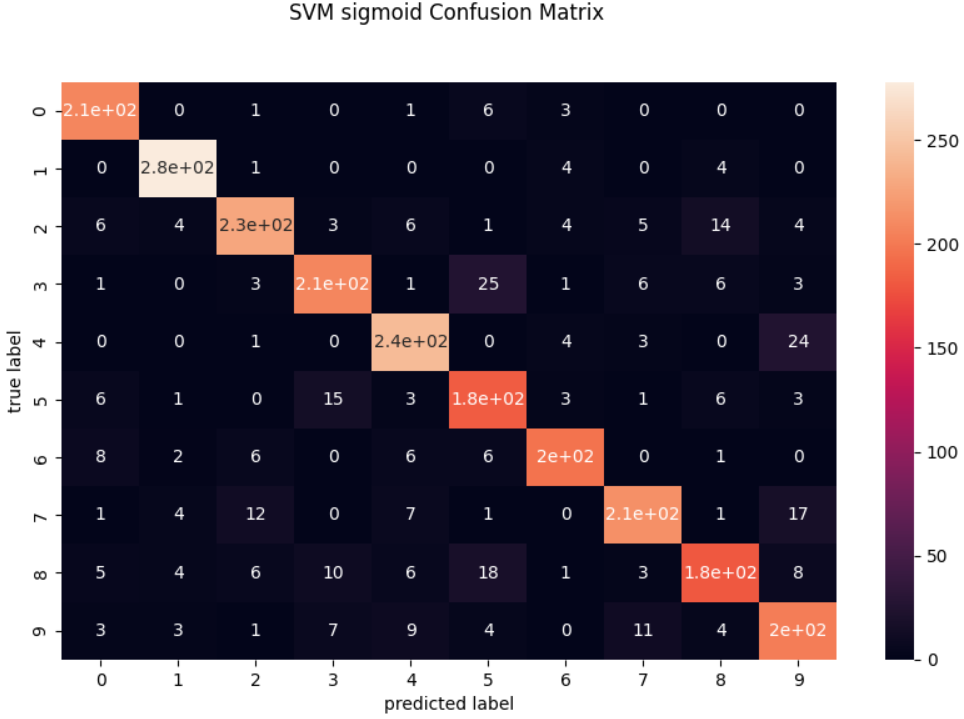
6. poly\_ax.set\_title('SVM poly Confusion Matrix **\n\n**')
7. poly\_ax.set\_xlabel('predict')
8. poly\_ax.set\_ylabel('true') *#*
10. plt.xlabel('predicted label')
11. plt.ylabel('true label')
12. plt.show()



（3）SVM sigmoid

1. sigmoid\_fig, sigmoid\_ax = plt.subplots(figsize=(10, 6))
3. sns.heatmap(sigmoid\_confmat,annot=True,ax=sigmoid\_ax) *#plot heatmap*

6. sigmoid\_ax.set\_title('SVM sigmoid Confusion Matrix **\n\n**')
7. sigmoid\_ax.set\_xlabel('predict')
8. sigmoid\_ax.set\_ylabel('true') *#*
10. plt.xlabel('predicted label')
11. plt.ylabel('true label')
12. plt.show()



**九、实验优化**

**（一）调参**

调参的最终目的是要使训练之后的模型检测物体更精确，向程序的方向更靠近一步的话，就是使得损失函数尽量小。

选择SVM中分类结果较好的RBF核函数进行调参优化

1. param\_grid={'C':[0.1,1,5,10,100],
2. 'gamma':[0.0005,0.001,0.005,0.01],}
3. parameter\_model = GridSearchCV(svm.SVC(kernel='rbf', class\_weight='balanced'), param\_grid)
4. parameter\_model.fit(pca\_x\_train,pca\_y\_train)
5. **print**(parameter\_model.best\_estimator\_)*#打印最好的参数*



根据此次结果继续细分，进行更进一步的调参，

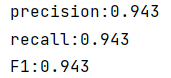
设置C在5附近

设置gamma在0.01附近

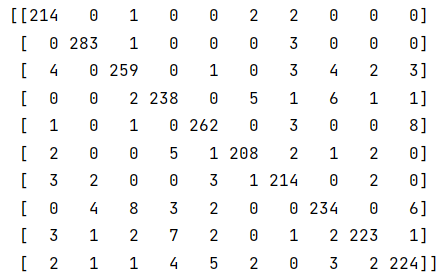
1. param\_grid\_1={'C':[4,5,6],
2. 'gamma':[0.007,0.01,0.013],}
3. parameter\_model\_1 = GridSearchCV(svm.SVC(kernel='rbf', class\_weight='balanced'), param\_grid)
4. parameter\_model\_1.fit(pca\_x\_train,pca\_y\_train)
5. **print**(parameter\_model\_1.best\_estimator\_)*#打印最好的参数*



1. **print**('precision:%.3f' % precision\_score(y\_true=y\_test, y\_pred=parameter\_predictions\_1,average= 'macro'))
2. **print**('recall:%.3f' % recall\_score(y\_true=y\_test, y\_pred=parameter\_predictions\_1,average= 'macro'))
3. **print**('F1:%.3f' % f1\_score(y\_true=y\_test, y\_pred=parameter\_predictions\_1,average= 'macro'))

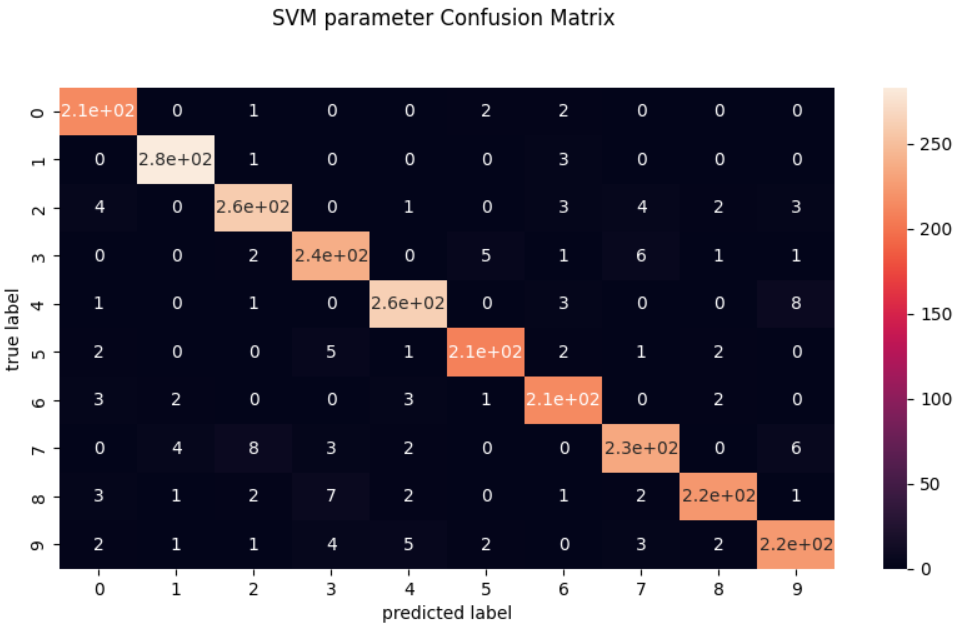


1. parameter\_confmat = confusion\_matrix(y\_true=pca\_y\_test, y\_pred=parameter\_predictions\_1) *# 输出混淆矩阵*
2. **print**(parameter\_confmat)



1. parameter\_fig, parameter\_ax = plt.subplots(figsize=(10, 6))
3. sns.heatmap(parameter\_confmat,annot=True,ax=parameter\_ax) *#plot heatmap*

6. parameter\_ax.set\_title('SVM parameter Confusion Matrix **\n\n**')
7. parameter\_ax.set\_xlabel('predict')
8. parameter\_ax.set\_ylabel('true') *#*
10. plt.xlabel('predicted label')
11. plt.ylabel('true label')
12. plt.show()



**十、总结**

**（一）SVM**

SVM使用铰链损失函数作为代理损失，铰链损失函数的取值特点使SVM具有稀疏性，即其决策边界仅由支持向量决定，其余的样本点不参与经验风险最小化 。在使用核方法的非线性学习中，SVM的稳健性和稀疏性在确保了可靠求解结果的同时降低了核矩阵的计算量和内存开销。

SVM是一个广义线性分类器，通过在SVM的算法框架下修改损失函数和优化问题可以得到其它类型的线性分类器，例如将SVM的损失函数替换为logistic损失函数就得到了接近于logistic回归的优化问题 。

**（二）SVR**

作为一种监督学习方法，SVR 使用对称损失函数进行训练，这同样会惩罚高错误估计和低错误估计。使用 Vapnik 的-不敏感的方法，在估计函数周围对称地形成一个最小半径的柔性管，使得小于某个阈值的误差绝对值在估计值之上和之下都被忽略。以这种方式，管外的点会受到惩罚，但管内的点，无论是高于还是低于功能，都不会受到惩罚。

SVR 的主要优点之一是其计算复杂度不依赖于输入空间的维度。此外，它具有出色的泛化能力，具有很高的预测精度。