《数据挖掘导论》实验 :支持向量机分类器实验

一、实验目的：

（1）理解支持向量机

（2）掌握利用SMO进行优化的具体方法步骤

（3）掌握利用核函数对数据进行空间转换

（4）了解常用分类器的性能特点

二、实验环境：

（1）Python编辑环境：Jupyter Notebook （或者其他Python环境均可）

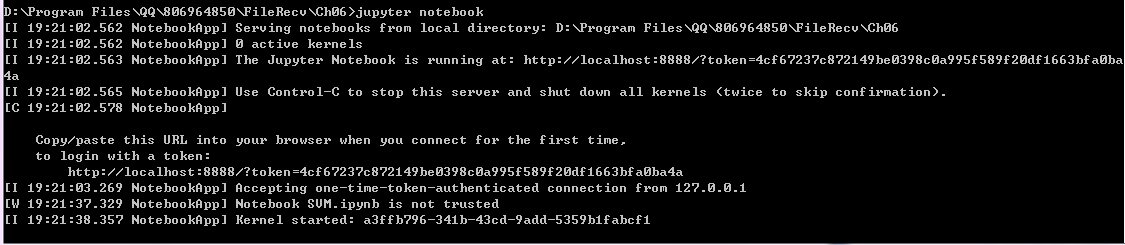
三、实验内容：

1.在命令行下输入下面命令进入jupyter notebook 环境

jupyter notebook

出现下面截图界面说明进入成功

命令行模式下：

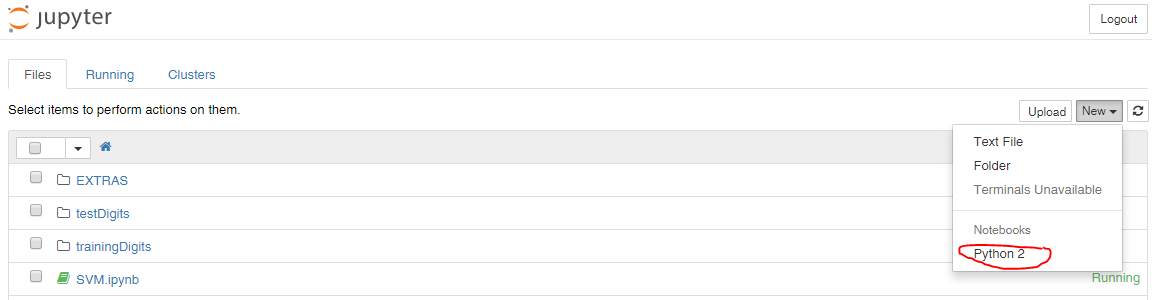


浏览器中自动打开下面截图界面：

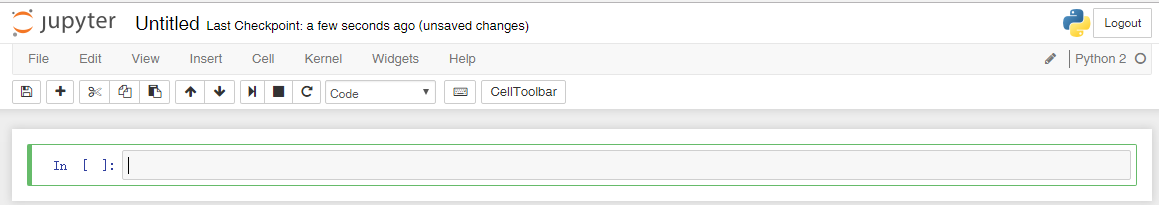


2.在jupyter环境下新建文件

如下图，点击‘New 🡪 Python2’新建Python文件。

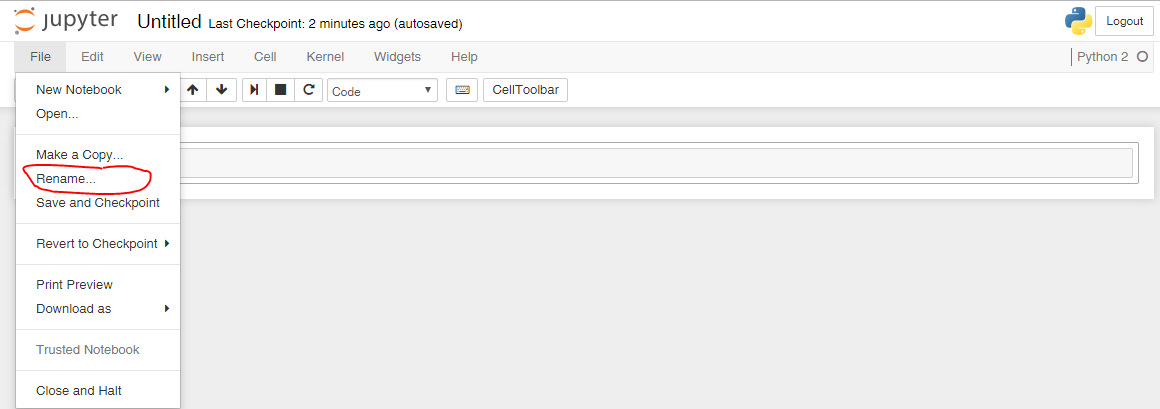


操作成功后会进入如下界面：



将代码输入到上图光标所在框内，点击‘Ctrl+ Enter’可实现运行

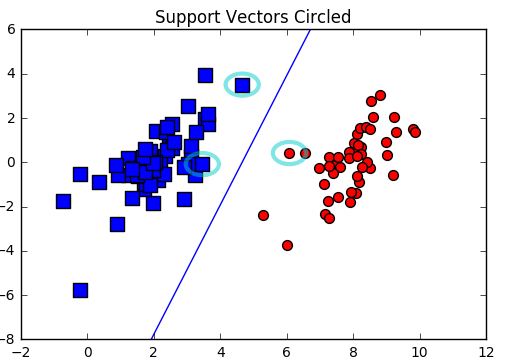
然后可以点击‘File🡪Rename’保存文件



3. 实现支持向量机分类器（SVM）

1）预备知识

SVM是最好的现成分类器，这里说的“现成”指的是分类器不加修改即可直接使用。同时，这就意味着在数据上应用基本形式的SVM分类器就可以得到低错误率的结果。SVM能够对训练集之外的数据点做出很好的分类决策。SVM有很多实现，本实验只关注其中最流行的一种实现，即序列最小化（Sequential Minimal Optimization, SMO）算法。



超平面：将数据集分隔开来的直线，平面或者n-1维的某对象，如上图蓝色的直线。超平面即分类决策的边界。分布在超平面一侧的所有数据都属于某个类别，而分布在另一侧的所有数据则属于另一个类别。本实验就是通过寻找这种最佳超平面的方式来构建分类器。我们希望找到离分隔超平面最近的点，确保他们离分隔面的距离尽可能远，这里点到分割面的距离被称为间隔。支持向量（support vector）就是离分隔超平面最近的那些点。接下来的实验部分要试着最大化支持向量到分隔面的距离，需要找到此问题的优化求解问题。

由于目前处理的数据都在二维平面上，所以此时分隔超平面就只是一条直线。因此分隔超平面的形式可以写成

wTx+b

要计算某一点A到分隔超平面的距离，就必须给出点到分隔面的法线或垂线的长度，该值为

|wTA+b|/||w||

这里的常数b类似于logistic回归中的截距w0。

向分类器输入数据，分类器会输出一个类别标签，这里定义为1和-1， 本实验使用类似海维赛德阶跃函数（即单位阶跃函数）的函数对wTx+b作用得到

f(wTx+b)

其中当u<0时f(u)输出-1， 反之则输出+1.当计算数据点到分隔面的距离并确定分隔面的放置位置时，间隔通过

label \* (wTx+b)

来计算。如果数据点处于正方向（即+1类）并且离分隔超平面很远位置时，wTx+b会是一个很大的正数。而如果数据点处于负方向（-1类）并且离分隔超平面很远的位置时，此时由于类别标签为-1，则label \* (wTx+b)仍然是一个很大的正数。

现在的目标是找到分类器定义中的w和b，为此，我们必须找到具有最小间隔的数据点，而这些数据点也就是前面提到的支持向量。一旦找到了具有最小间隔的数据点，我们就需要对该间隔最大化。这就写作:



如果令所有支持向量的label\*(wTx+b)都为1，那么就可以通过求||w||-1的最大值来得到最终的解。但是并非所有数据点的label\*(wTx+b)都为1，只有那些离分隔超平面最近的点得到的值才为1，而离超平面越远的数据点，其label\*(wTx+b)也就越大，所以在该问题中，可以给定一些约束条件然后求最优值。这里约束条件为label\*(wTx+b) 》1.0.然后运用拉格朗日乘子法，可以将超平面写成数据点的形式，于是优化目标函数就可以写成：



其约束条件为：

， 和 

但数据并不总是100%线性可分，所以我们需要引入所谓的松弛变量来允许有些数据点可以处于分隔面的错误一侧，因此新的约束条件为：

， 和 

这里的C用于控制“最大化间隔”和“保证大部分点的函数间隔小于1.0”这两个目标的权重。在后面的优化算法中常数C是一个参数，因此我们可以通过调节该参数得到不同的结果。一旦求出了所有的alpha，那么分隔超平面就可以通过这些alpha来表达。而SVM的工作主要是求解这些alpha。

**SVM的一般流程：**

* 收集数据：可以使用任意方法
* 准备数据：需要数值型数据
* 分析数据：有助于可视化分隔超平面
* 训练算法：SVM的大部分时间都源自训练，该过程主要实现两个参数的调优。
* 测试算法：十分简单的计算过程就可以实现
* 使用算法：几乎所有分类问题都可以使用SVM，值得一提的是，SVM本身是一个二类分类器，对多类问题应用SVM需要对代码做一些修改。

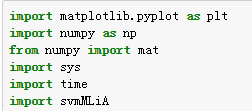
**SMO算法的目标：**

求出一系列alpha和b，一旦求出了这些alpha，就很容易计算出权重向量w并得到分隔超平面。

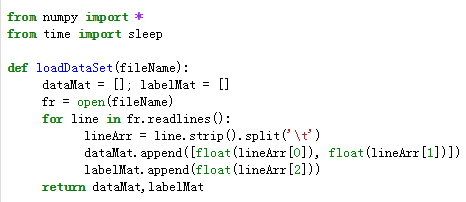
**SMO算法的工作原理**：

每次循环中选择两个alpha进行优化处理。一旦找到一对合适的alpha，那么就增大其中一个同时减小另一个。这里所谓的“合适”就是指两个alpha必须要符合一定的条件，条件之一就是这两个alpha必须要在间隔边界之外，而其第二个条件则是这两个alpha还没有进行过区间化处理或者不在边界上。

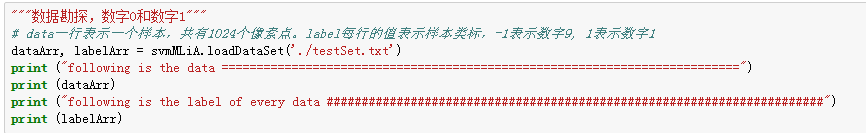
2）导入需要的模块



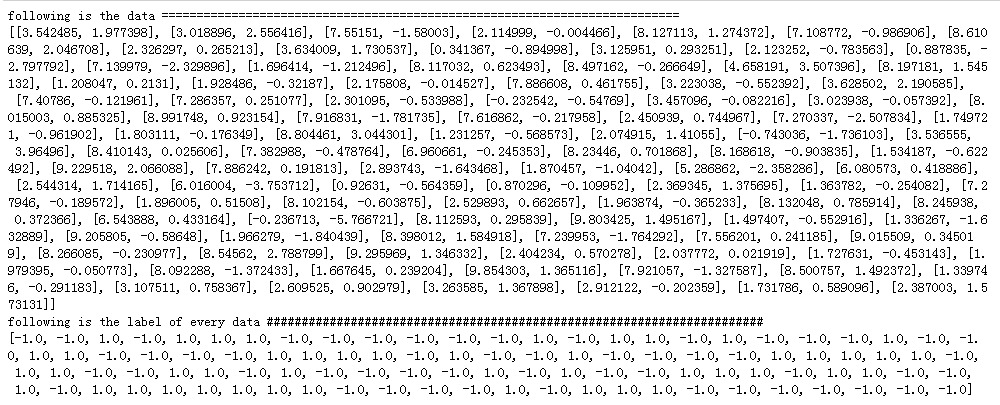
3）读取数据，这里数据保存在‘testSet.txt’文件中。



使用下面代码执行数据读取函数：

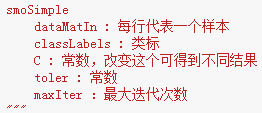


运行结果如下：



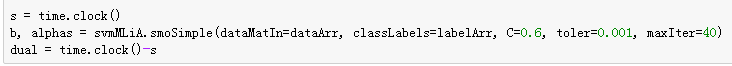
从运行结果，我们可以看出，这里采用的累呗标签是-1和1。

4）简化版的SMO算法

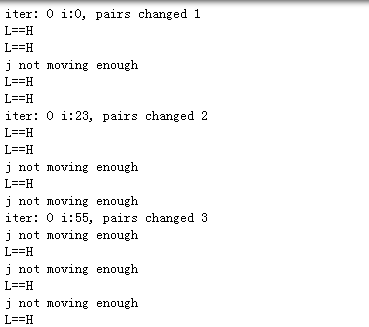




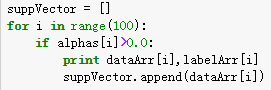
运行下面命令，了解算法的实际效果



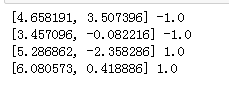
运行结果如下：



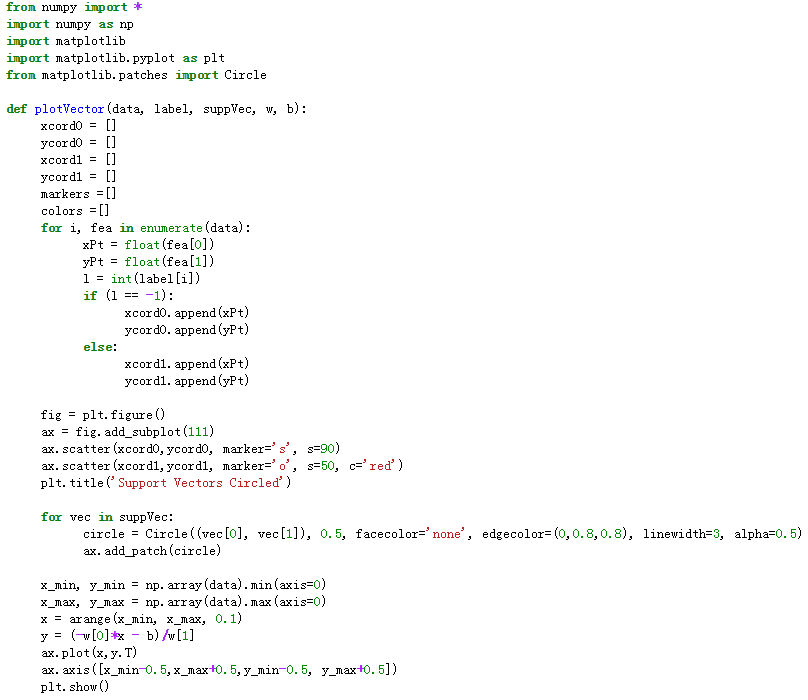
5）构建支撑向量



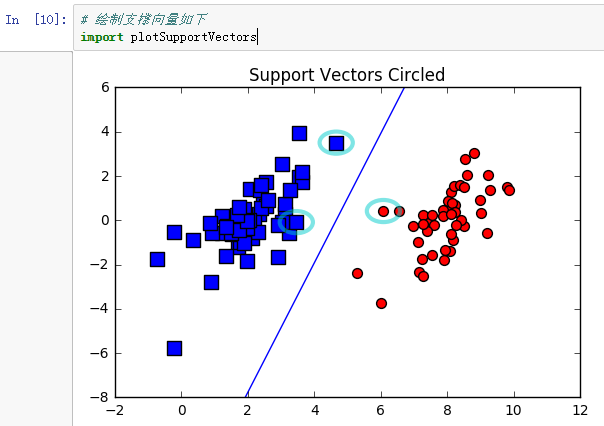
运行结果如下：



使用如下代码绘制支撑向量如下：

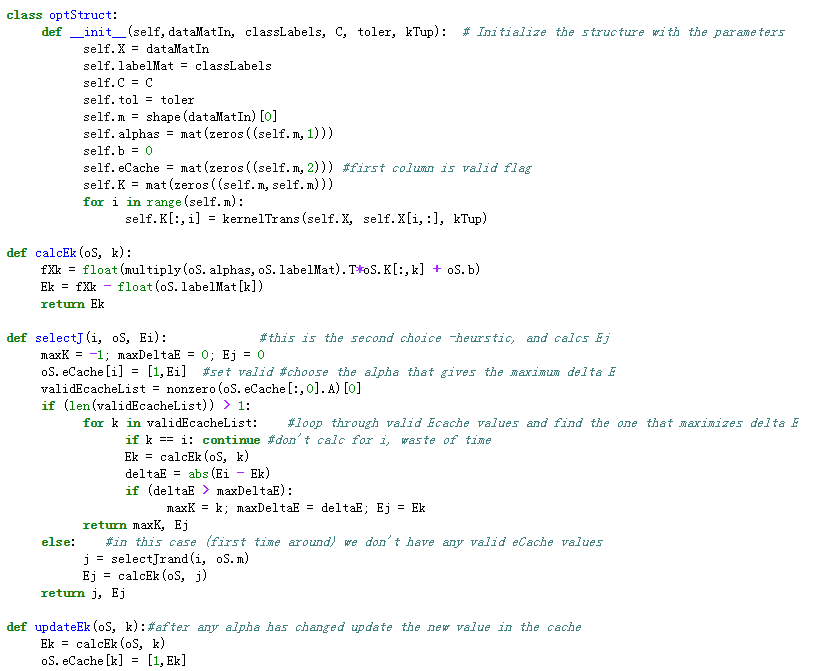


效果如下图：

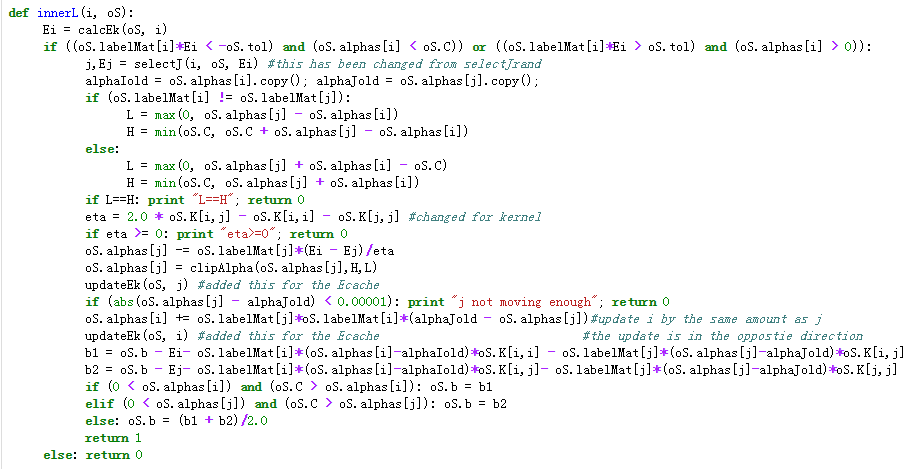


6）利用完整Platt SMO算法加速优化

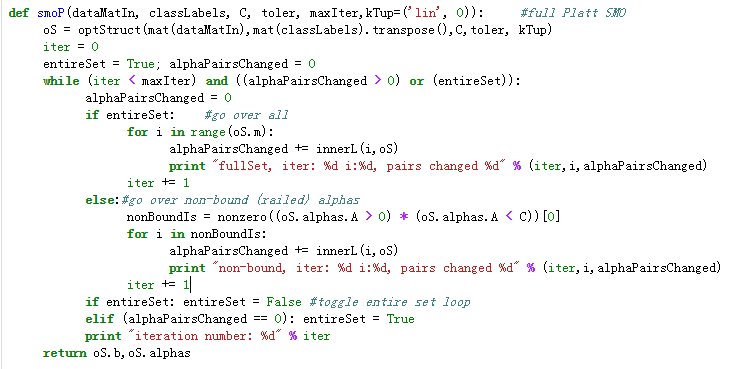
完整版Platt SMO 的支持函数如下：



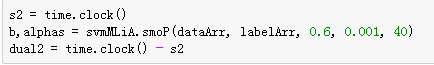
完整Platt SMO 算法中的优化例程如下：

****

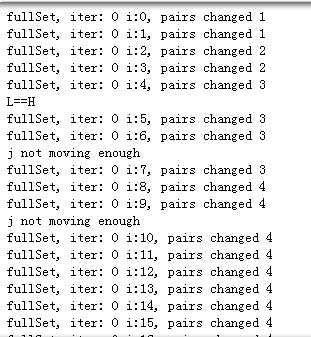
完整版的Platt SMO的外循环代码

****

**优化函数的调用：**

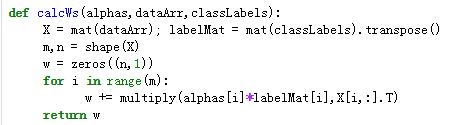
****

**运行效果如下：**

****

**7）将数据进行分类**

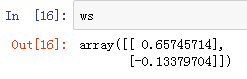
**分类算法如下：**

****

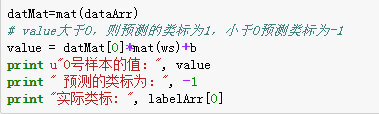
**利用alpha值进行分类：**

****

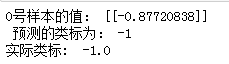
**运行效果如下：**

****

**分类数据：**

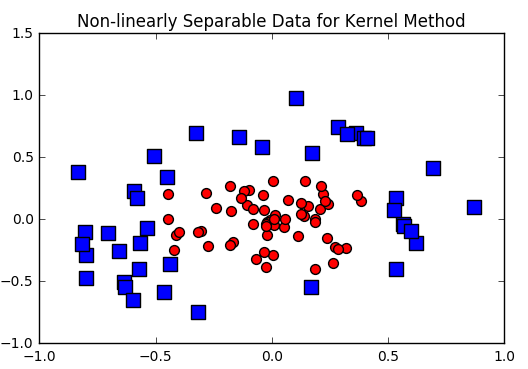
****

**运行结果如下：**

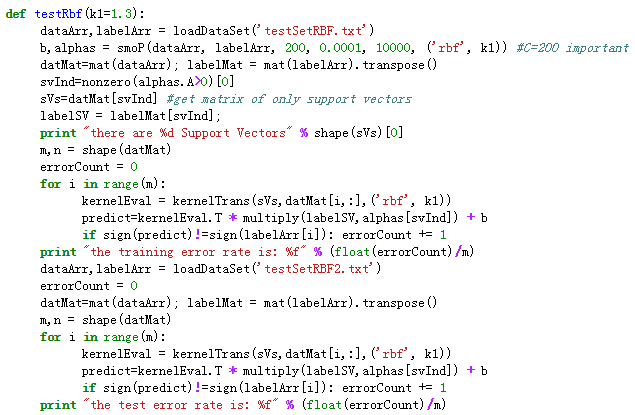
****

**8）测试算法：非线性可分来实践svm**

**数据如下图，非线性可分。**

****

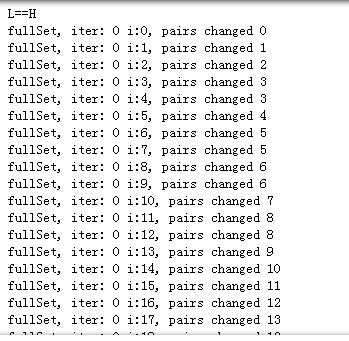
**Rbf测试算法：**

****

**测试RBF：**

****

**运行结果如下：**

****

4. 任务：使用以上实验算法思想完成独立实验---手写识别问题

基于SVM的手写体识别问题， 比较不同δ下手写数字的识别性能

1. 参考思路

收集数据：提供的文本文件

准备数据： 基于二值图像构造向量

分析数据：对图像向量进行目测

训练算法：采用两种不同的核函数，并对径向基核函数采用不同的设置来运行SMO算法。

测试算法：编写一个函数来测试不同的核函数并计算错误率

使用算法：一个图像识别的完整应用还需要一些图像处理的知识。

1. 参考函数



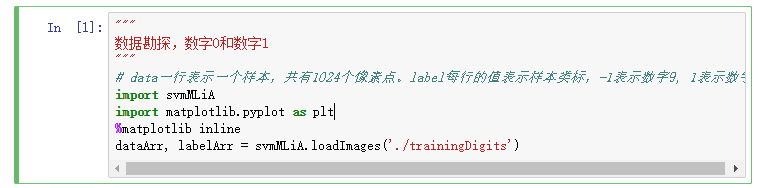
1. 比较不同δ值下的算法性能，最后填写如下列表

|  |  |  |  |
| --- | --- | --- | --- |
| 内核，设置 | 训练准确率 | 测试错误率 | 支持向量数 |
| RBF, 0.1 | 100% | 52.1505% | 402 |
| RBF, 5 | 100% | 3.2258% | 402 |
| RBF, 10 | 100% | 1.0753% | 120 |
| RBF, 50 | 98.7562% | 2.6882% | 40 |
| RBF, 100 | 99.5025% | 1.6129% | 36 |
| Linear | 99.92537% | 1.0753% | 33 |

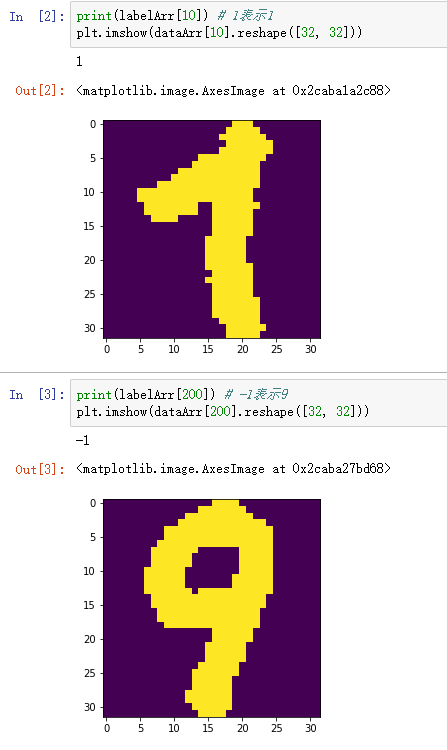
1. 编写实验报告

①.使用文件中写好的SMO进行我们的实验

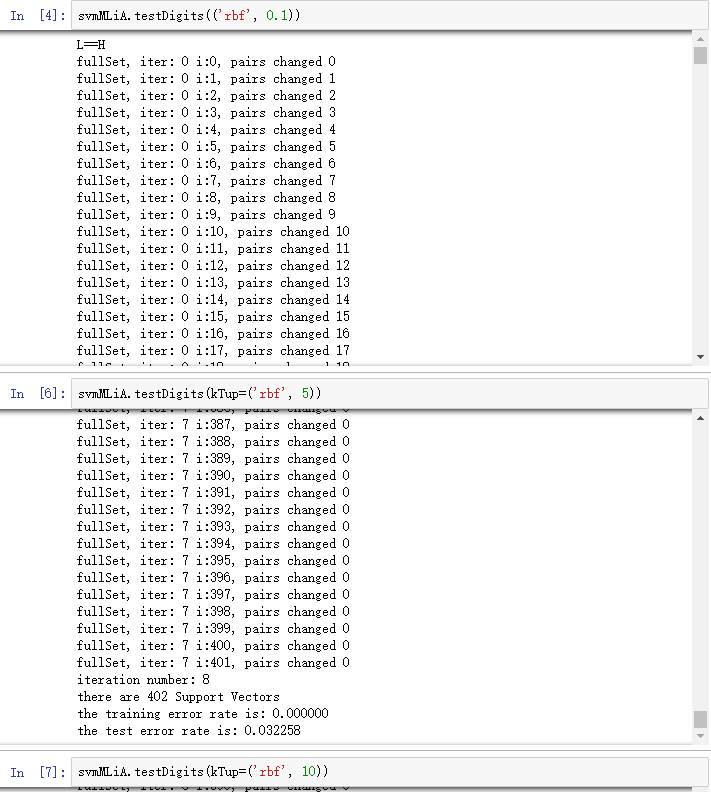
⒈先进行需要的包的导入，以及数据集的导入，代码如下图所示。



⒉分别显示数字1和数字9的一个例子，如下图所示。



⒊直接调用测试函数得到结果，如下图所示。



②．使用sklearn包进行实验

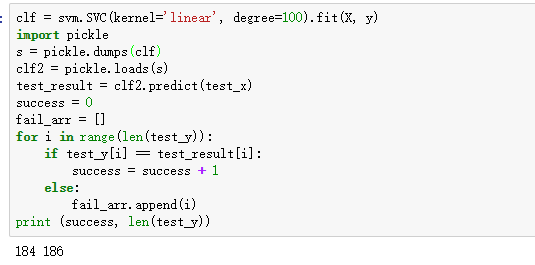
⒈导入需要使用的包以及图片内联，如下图所示。



⒉分别导入训练数据和测试数据，如下图所示。



⒊使用linear的kernel对我们的值进行训练，然后测试得到错误的仅有两个，如下图所示。



⒋因为我们的错误率比较低，所以调出我们识别错误的图片进行查看，如下图所示。



可以见到，上面的两幅图，就算让人做识别也很难识别的出来，毕竟没有一个统一的标准。

而机器可以做到如此程度，也多亏了sklearn中写的svm算法，这样才能有如此识别成功率。

当然，内置方法里面实现的线性得出的错误率其实是和使用sklearn是一样的，充分证明了，这玩意只有性能比较差，其他算法本身还是没有精简的。