****

**机器学习**

**实验报告**

**学 号 U201913602**

**姓 名 莫茗程**

**专 业 电信1904**

**任 课 教 师 王邦**

**院（系、所） 电信学院**

**2022年 4月28日**

**任务一：分别用线性SVM和高斯核SVM预测对数据进行分类**

1.任务要求：

task1\_linear.mat中有一批数据点，试用线性SVM对他们进行分类，并在图中画分出决策边界。task1\_gaussian中也有一批数据点，试用高斯核SVM对他们进行分类，并在图中画出决策边界。

2.实验原理：

在SVM\_Functions.py中已经给定了SMO训练算法的代码。可以直接调用函数进行训练和决策。下面对支持向量机的原理、数据集和代码进行分析。

2.0原理

给定训练样本集

用划分超平面，间隔.

找到满足约束和b使得最大即得到最小

转化为拉格朗日对偶问题，每条约束添加拉格朗日乘子,则拉格朗日函数写为

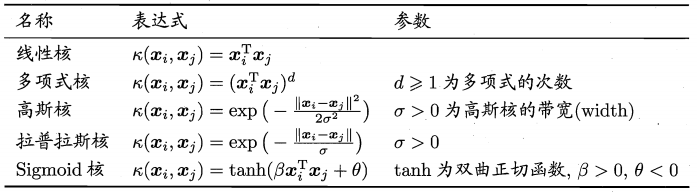
偏导为0得到 KKT条件

最终得到模型

SMO的基本思路是先固定之外的所有参数,然后求上的极值.由于存在约束,若固定之外的其他变量,则可由其他变量导出.于是,SMO每次选择两个变量和，并固定其他参数.不断选取需更新的和，固定其他参数，更新和直至收敛

引入核函数 则有

常用的核函数有



需要特别说明的是，以上的推导都是建立在“硬间隔”的基础上，“硬间隔”要求样本集中每一个样本都满足约束条件。在现实中往往很难确定合适的核函数使得训练样本在特征空间中是线性可分的，缓解该问题的一个办法是允许支持向量机在一些样本上出错，为此引入“软间隔”的概念，允许有部分样本不满足这样的约束。

2.1数据集分析

task1\_linear.mat中X为51\*2，y为51\*1

task1\_gaussian.mat中X为863\*2，y为863\*1

X中的元素取值范围为0-1，y的取值范围为0和1

2.2 loadData(filename)加载数据

输入参数为要加载的文件名，输出参数为数据集中的X，y。实际上就是loadmat

2.3 plotData(X, y,title=None)作出数据的散点图

单条X的数据为2维，可以作二维图并在图上相应的位置标出来

np.concatenate将每一条数据的X，y拼接起来，并根据y的值分为正例和反例

plt.scatter将数据点在图上标出，根据y的值，正类标为蓝色的+，负类标为黄色的o

2.4 svmTrain\_SMO(X, y, C, kernelFunction='linear', tol=1e-3, max\_iter=5, \*\*kargs)

SMO算法训练支持向量机.输入X，y，C为惩罚参数，C越大，模型越接近硬间隔SVM

kernelFunction指定了核函数类型，可选高斯核和线性核。选择高斯核时需要添加预先计算好的核矩阵作为额外参数。tol为容错率。max\_iter为最大迭代次数。

预测误差

选择了需要更新的和后，根据公式计算和，然后确定α修改时的上下界

(i,j分别用1,2表示)

α更新公式

其中,得到

b更新公式

计算公式

使用time计时程序运行时间，同时使用print打印“···”进度条，避免程序运行过长，终端难以看出运行进度。训练之后返回的模型为字典，存储了X,y,核类型,b,,

2.5 visualizeBoundaryLinear(X, y, model, title=None)

使用线性核训练后，用该函数将决策边界画出。在图中画出原始数据正反例点的代码与plotData相同，额外增加了读取模型中的 并将参数对应的直线在图上画出

*w = model['w']*

*b = model['b']*

*xp = np.linspace(min(X[:,0]), max(X[:,0]), 100)*

*yp = np.squeeze(np.array(- (w[0]\*xp + b)/w[1]))*

*ax.plot(xp, yp)*

2.6 高斯核

高斯核计算分为两个函数，gaussianKernelSub(x1, x2, sigma)和gaussianKernel(X,sigma)



gaussianKernelSub传入参数,根据公式输出计算后的结果

作为gaussianKernel中的核函数。gaussianKernel循环遍历最终生成核矩阵并返回

2.7 svmPredict(model, X, \*arg)

利用得到的model, 计算给定X的模型预测值。该函数根据模型中核函数的类型进行预测

线性 高斯核

线性代码为p = X \* model['w'] + model['b']

高斯核代码为循环

prediction += model['alphas'][:,j]\*model['y'][:,j]\*\

gaussianKernelSub(X[i,:].T, model['X'][j,:].T, \*arg)

最后再p[i] = prediction + model['b']

预测结束后，将所有元素大于0的设置为正例1，小于0的设置为反例0再返回预测值

2.8 visualizeBoundaryGaussian(X, y, model, sigma)

画出高斯核决策边界的函数。在图中画出原始数据正反例点的代码与plotData相同

*for i in range(X1.shape[1]):  
 print('.',end='')  
 dots += 1  
 if dots == 78:   
 dots = 0  
 print()  
 this\_X = np.concatenate((X1[:,i], X2[:,i]), axis=1)  
 vals[:,i] = svmPredict(model, this\_X, sigma)*

预测X中每一个数据，得到vals，再用ax.contour(X1, X2, vals, colors='black')将vals组成的决策边界画出来

3.实验思路：

3.1加载与可视化数据

用loadData加载数据，plotData可视化数据

*linear\_data\_x, linear\_data\_y = loadData('task1\_linear.mat')*

*gaussian\_data\_x, gaussian\_data\_y = loadData('task1\_gaussian.mat')*

*plotData(linear\_data\_x, linear\_data\_y)*

*plotData(gaussian\_data\_x, gaussian\_data\_y)*

3.2训练模型

用svmTrain\_SMO训练模型，高斯核需要额外传入核矩阵作为参数

*linear\_model = svmTrain\_SMO(linear\_data\_x, linear\_data\_y, C=1, max\_iter=20)*

*gaussian\_kernal = gaussianKernel(gaussian\_data\_x, sigma=0.1)*

*gaussian\_model = svmTrain\_SMO(gaussian\_data\_x, gaussian\_data\_y, C=1, \*

*kernelFunction='gaussian', K\_matrix=gaussian\_kernal)*

3.3决策边界可视化

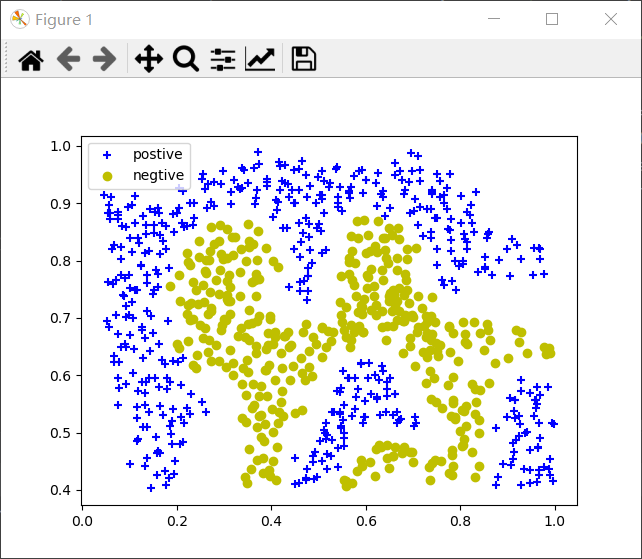
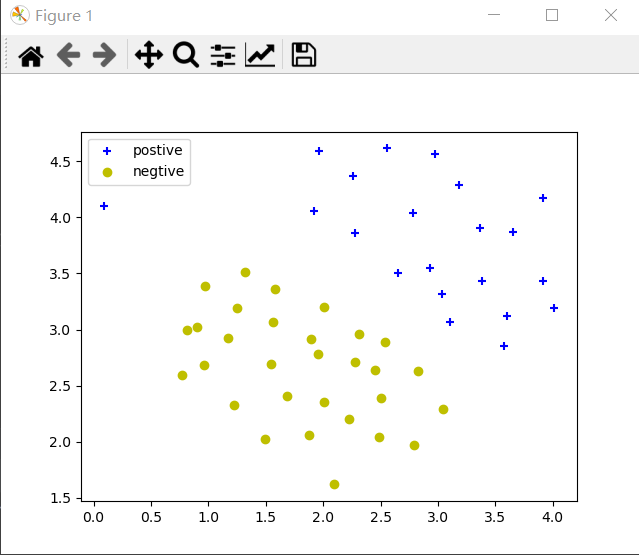
线性核用visualizeBoundaryLinear画出边界，高斯核用visualizeBoundaryGaussian

*visualizeBoundaryLinear(linear\_data\_x, linear\_data\_y, linear\_model)*

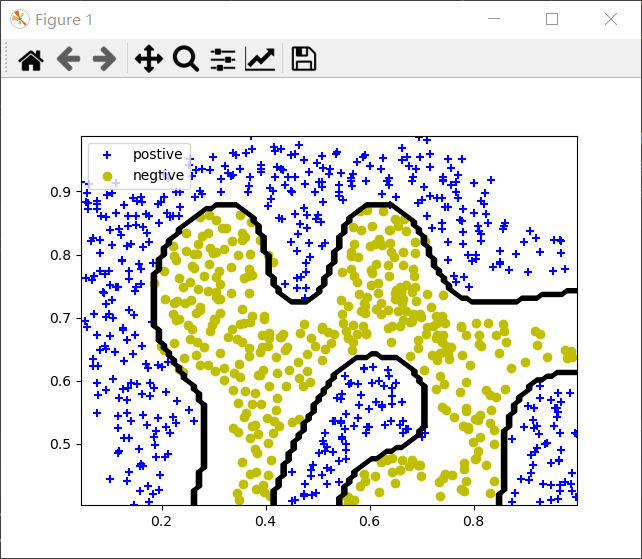
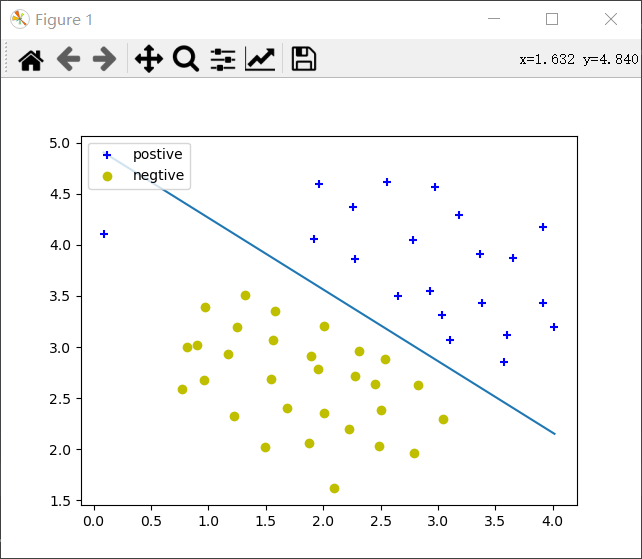
*visualizeBoundaryGaussian(gaussian\_data\_x, gaussian\_data\_y, gaussian\_model, 0.1)*

4.实验结果

如下图，为原始数据可视化，左边为task1\_linear.mat，右边为task1\_gaussian.mat



下图为画出决策边界的数据，可见划分效果还不错



**任务二：使用高斯核SVM对给定数据集进行分类**

1.任务要求：

给定数据集（文件task2.mat）, 参考task1的代码, 编程实现一个高斯核SVM进行分类。输出训练参数C, sigma分别取0.01, 0.03, 0.1, 0.3, 1, 3, 10, 30时(共64组参数组合)的训练集上的准确率。

*for sigma in sigmas:*

*kernal = gaussianKernel(data\_x, sigma)*

*kernals.append(kernal)*

2.实验原理：

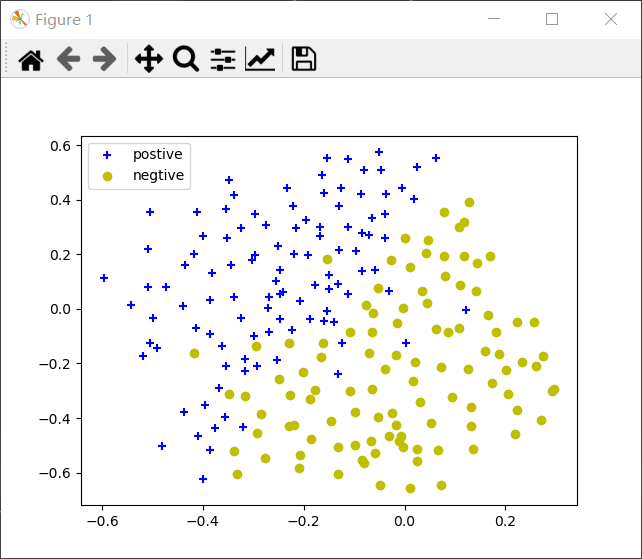
采用任务一中给出的代码进行调用。通过循环调用传入不同的参数。

本实验采用高斯核进行训练。

3.实验思路：

loadData读取数据*data\_x, data\_y = loadData('task2.mat')*

数据可视化



设置参数集为*sigmas = [0.01, 0.03, 0.1, 0.3, 1, 3, 10, 30]*

*C = [0.01, 0.03, 0.1, 0.3, 1, 3, 10, 30]*

遍历每一个sigmas，得到对应的高斯核矩阵。

*for sigma in sigmas:*

*kernal = gaussianKernel(data\_x, sigma)*

*kernals.append(kernal)*

再二重循环遍历每一行C与sigmas的组合，训练出每个组合对应的模型

*for c in C:*

*c\_model = []*

*for kernal in kernals:*

*model = svmTrain\_SMO(data\_x, data\_y, C=c, kernelFunction='gaussian', \*

*K\_matrix=kernal)*

*c\_model.append(model)*

*models.append(c\_model)*

一共得到64个模型，每个模型进行预测得到结果

*for c\_model in models:*

*c\_pred = []*

*for i in range(len(c\_model)):*

*start = time.perf\_counter()*

*print('pridicting ...', end='')*

*pred = svmPredict(c\_model[i], data\_x, sigmas[i])*

*c\_pred.append(pred)*

*print('Done', end='')*

*end = time.perf\_counter()*

*print('( ' + str(end - start) + 's )')*

*print()*

*preds.append(c\_pred)*

preds就是64个模型预测得到的结果，再与data\_y比较得到正确个数，除以总个数得到精度*accuracy = np.sum(pred == data\_y)/l*

4.实验结果：

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| c\sigma | 0.01 | 0.03 | 0.1 | 0.3 | 1 | 3 | 10 | 30 |
| 0.01 | 0.502 | 0.498 | 0.498 | 0.498 | 0.498 | 0.502 | 0.502 | 0.498 |
| 0.03 | 0.502 | 0.502 | 0.502 | 0.867 | 0.825 | 0.498 | 0.502 | 0.502 |
| 0.1 | 0.502 | 0.502 | 0.948 | 0.867 | 0.839 | 0.82 | 0.502 | 0.502 |
| 0.3 | 0.502 | 0.986 | 0.948 | 0.905 | 0.872 | 0.758 | 0.502 | 0.498 |
| 1 | 1 | 0.995 | 0.948 | 0.938 | 0.877 | 0.853 | 0.502 | 0.502 |
| 3 | 1 | 1 | 0.943 | 0.929 | 0.929 | 0.872 | 0.692 | 0.502 |
| 10 | 1 | 1 | 0.962 | 0.943 | 0.929 | 0.9 | 0.853 | 0.502 |
| 30 | 1 | 1 | 0.948 | 0.938 | 0.929 | 0.919 | 0.867 | 0.654 |

**任务三：使用线性SVM实现对垃圾邮件分类**

1.任务要求：

编程实现一个垃圾邮件SVM线性分类器，分别在训练集和测试集上计算准确率。其中训练数据文件：task3\_train.mat，要求导入数据时输出样本数和特征维度。测试数据文件：task3\_test.mat，要求导入数据时输出样本数和特征维度，测试数据标签未给出。

2.实验原理：

采用任务一中给出的代码进行调用。采用线性核进行训练。

3.实验思路：

task3\_test.mat是测试集，没有y标签，不能直接用loadData

读取数据

*test\_features = loadmat('task3\_test.mat')['X']*

*train\_features, train\_label = loadData('task3\_train.mat')*

为了测试训练的模型精度，调用sklearn库的train\_test\_split划分为训练集和测试集

*x\_train, x\_test, y\_train, y\_test = train\_test\_split(train\_features, train\_label, test\_size=0.2)*

训练模型

linear\_model = svmTrain\_SMO(x\_train, y\_train, C=1, max\_iter=20)

得到模型之后进行预测

*predict = svmPredict(linear\_model, test\_features)*

*pred = svmPredict(linear\_model, x\_test)*

计算模型的精度

*accuracy = np.sum(y\_test == pred)/l*

保存预测结果

*f = open('test\_labels.txt', 'w')*

*for i in predict:*

*s = str(int(i[0])) + '\n'*

*f.write(s)*

*f.close()*

4.实验结果：



精度的计算由划分出来的验证集进行测试