华中科技大学

机器学习导论作业报告

专业院系 ：电子信息与通信学院
专业班级 ：电信1805班
学生学号 ：U201813372
学生姓名 ：朱浩然

支持向量机SVM编程作业

TASK1:分别用线性SVM和高斯核SVM预测对数据进行分类

一、软件设计

1. 设计目标

① 目标概述

(1) 问题概述：

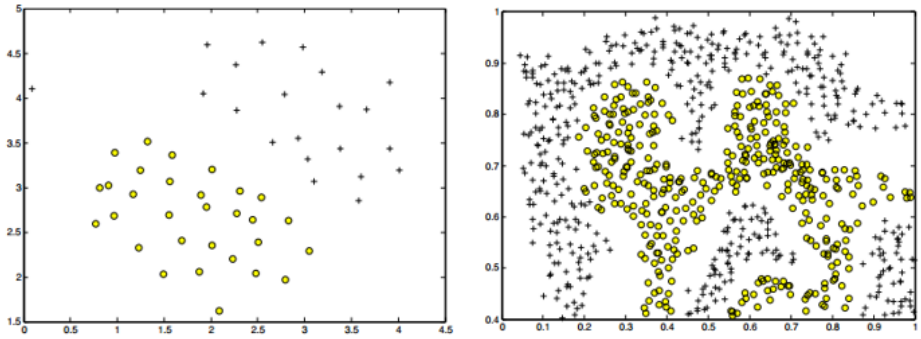
task1_linear.mat中有一批数据点，试用线性SVM对他们进行分类，并在图中画分出决策边界。
task1_gaussian中也有一批数据点，试用高斯核SVM对他们进行分类，并在图中画出决策边界。

(2) 步骤提示：

1.实验所需函数的代码在SVM_Functions.py中已经给出，请尝试读懂代码，理解算法思想与步骤，并自行读入数据，执行所需函数，观察实验结果。（也可以自己编写实验代码）

2.附件PPT中给出了SMO训练算法的详细分析。

3.task1_linear.mat和task1_gaussian.mat图像如下所示：



4.训练过程：

（a）加载与可视化数据

```
loadData(), plotData()
```

（b）训练模型

```
 model = SVM_Functions.svmTrain_SMO(X, y, C=1, max_iter=20)
 model=SVM_Functions.svmTrain_SMO(X,y,C=1,kernelFunction='gaussian',K_matrix=s.g
aussianKernel(X,sigma=0.1))
```

（c）决策边界可视化

```
 SVM_Functions.visualizeBoundaryLinear(X, y, model)
```

② 目标要求

任务一需要编写实验报告进行简述其原理，代码步骤，并根据训练结果在图中画出决策边界。

2. 算法设计

> **所用项目环境 IDE：VS code　　Python版本：python3.7.5**

① SVM支持向量机

> 给定数据样本集$D=\{(x_1,y_1),(x_2,y_2),\ldots,(x_m,y_m)\},y_i\in\{-1,+1\}$

- **基本概念**：

  划分超平面的**线性方程**：$w^Tx+b=0$,法向量$w=(w_1;w_2;\ldots;w_d)$，位移项b

  间隔$\gamma=\frac{2}{||w||}$,欲找到具有最大间隔的超平面，问题转化为下列问题，也即SVM基本型：

$$\min_{w,b}\ \frac{1}{2}||w||^2$$

$$s.t.\ y_i\left(w^Tx_i+b\right)\geq 1,i=1,2,\ldots,m$$

  使用拉格朗日乘子法可得到“**对偶问题**”：

$$L\left(w,b,\alpha\right)=\frac{1}{2}||w||^2+\sum_{i=1}^{m}\alpha_i\left(1-y_i\left(w^Tx_i+b\right)\right)$$

  可转化为：

$$\max_{\alpha}\ \sum_{i=1}^{m}\alpha_i-\frac{1}{2}\sum_{i=1}^{m}\sum_{j=1}^{m}\alpha_i\alpha_jy_iy_jx_i^Tx_j$$

$$s.t.\ \sum_{i=1}^{m}\alpha_iy_i=0,\ \alpha_i\geq 0,i=1,2,\ldots,m$$

  上述过程需要满足**KKT条件**：

$$\begin{cases}\alpha_i\geq 0;\\ y_if\left(x_i\right)-1\geq 0;\\ \alpha_i\left(y_if\left(x_i\right)-1\right)=0;\end{cases}$$

  此时**划分方程f(x)**转化为：

$$f\left(x\right)=\sum_{i=1}^{m}\alpha_iy_iK\left(x_i*x_j\right)+b$$

> 1. 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。
> 2. 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。
> 3. 具体来说，“硬间隔”要求所有参与训练的样本都必须满足SVM的约束条件，而“软间隔”允许有部分样本不满足这样的约束。

- **软间隔支持向量机**

  对偶问题转化为：

$$\max_{\alpha} \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i=1}^{m} \sum_{j=1}^{m} \alpha_i \alpha_j y_i y_j x_i^T x_j$$

$$s.t. \sum_{i=1}^{m} \alpha_i y_i = 0, \ \alpha_i \geq 0, i = 1, 2, \ldots, m$$

  KKT条件转化为：

$$\begin{cases} \alpha_i \geq 0; \mu_i \geq 0; \\ y_i f(x_i) - 1 + \xi_i \geq 0; \\ \alpha_i (y_i f(x_i) - 1 + \xi_i) = 0; \\ \xi_i \geq 0; \mu_i \xi_i = 0; \end{cases}$$

- **核函数**

  令X为输入空间，k(·,·)是定义在X*X上的对称函数，则k是核函数当且仅当对于任意数据D，“核矩阵”K总是半正定的：

$$\mathbf{K} = \begin{bmatrix} \kappa(\boldsymbol{x}_1, \boldsymbol{x}_1) & \cdots & \kappa(\boldsymbol{x}_1, \boldsymbol{x}_j) & \cdots & \kappa(\boldsymbol{x}_1, \boldsymbol{x}_m) \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ \kappa(\boldsymbol{x}_i, \boldsymbol{x}_1) & \cdots & \kappa(\boldsymbol{x}_i, \boldsymbol{x}_j) & \cdots & \kappa(\boldsymbol{x}_i, \boldsymbol{x}_m) \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ \kappa(\boldsymbol{x}_m, \boldsymbol{x}_1) & \cdots & \kappa(\boldsymbol{x}_m, \boldsymbol{x}_j) & \cdots & \kappa(\boldsymbol{x}_m, \boldsymbol{x}_m) \end{bmatrix}.$$

  常用核函数：

名称	表达式	参数
线性核	$\kappa(\boldsymbol{x}_i, \boldsymbol{x}_j) = \boldsymbol{x}_i^{\mathrm{T}} \boldsymbol{x}_j$	
多项式核	$\kappa(\boldsymbol{x}_i, \boldsymbol{x}_j) = (\boldsymbol{x}_i^{\mathrm{T}} \boldsymbol{x}_j)^d$	$d \geqslant 1$ 为多项式的次数
高斯核	$\kappa(\boldsymbol{x}_i, \boldsymbol{x}_j) = \exp\left(-\frac{\|\boldsymbol{x}_i - \boldsymbol{x}_j\|^2}{2\sigma^2}\right)$	$\sigma > 0$ 为高斯核的带宽(width)
拉普拉斯核	$\kappa(\boldsymbol{x}_i, \boldsymbol{x}_j) = \exp\left(-\frac{\|\boldsymbol{x}_i - \boldsymbol{x}_j\|}{\sigma}\right)$	$\sigma > 0$
Sigmoid 核	$\kappa(\boldsymbol{x}_i, \boldsymbol{x}_j) = \tanh(\beta \boldsymbol{x}_i^{\mathrm{T}} \boldsymbol{x}_j + \theta)$	tanh 为双曲正切函数, $\beta > 0$, $\theta < 0$

② SMO算法

SMO的基本思路是先固定α_i之外的所有参数,然后求α_i上的极值.由于存在约束$\sum_{i=1}^{m} \alpha_i y_i = 0$,若固定αi之外的其他变量,.则αi可由其他变量导出.于是,SMO每次选择两个变量ai和aj,并固定其他参数.这样,在参数初始化后,SMO不断执行如下两个步骤直至收敛:

- 选取一对需更新的变量α_i和α_j;
- 固定α_i和α_j以外的参数，求解转化的对偶问题公式获得更新后的α_i和α_j;

由约束$\sum_{i=1}^{m} \alpha_i y_i = 0$,可以得到$\alpha_1 y_1 + \alpha_2 y_2 = C \rightarrow \alpha_1 = (C - \alpha_2 y_2) y_1$

> 具体对偶问题求解过程省略，主要为求偏导，下面简单介绍求解结果
>
> 跳转回SMO算法代码实现 跳转回TASK2 跳转回TASK3

（1）决策误差

已知$f\left(x\right)=\sum_{i=1}^{m}\alpha_i y_i K\left(x_i*x_j\right)+b$,得出误差Ei为：

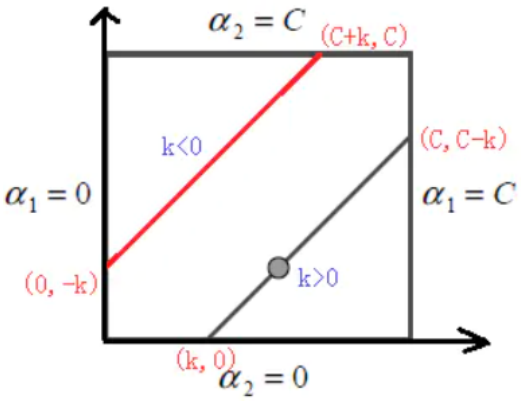
$$E_i=f(x_i)-y_i,i=1,2$$

当i=1,2时，Ei表示函数f(x)对输入xi的预测值与真实值yi之间的差值

（2）α修改时的上下界L、H

- $y_1\neq y_2$时：

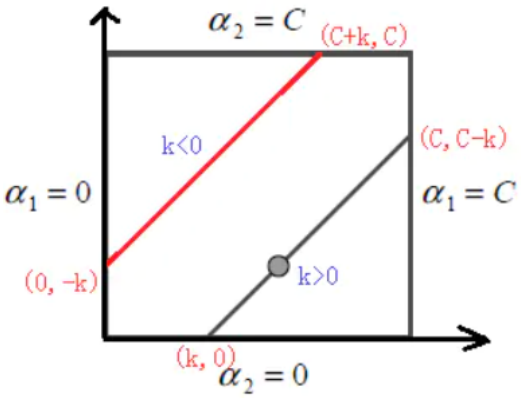
$$L=\max\left(0,\alpha_2^{old}-\alpha_1^{old}\right);\ \ H=\min\left(C,C+\alpha_2^{old}-\alpha_1^{old}\right)$$



$$y_1\neq y_2\Rightarrow\alpha_1-\alpha_2=k$$

- $y_1=y_2$时：

$$L=\max\left(0,\alpha_2^{old}-\alpha_1^{old}-C\right);\ \ H=\min\left(C,\alpha_2^{old}-\alpha_1^{old}\right)$$



$$y_1\neq y_2\Rightarrow\alpha_1-\alpha_2=k$$

（3）α更新公式

- 未剪辑前：

$$\alpha_2^{new,nuc}=\alpha_2^{old}+\frac{y_2\left(E_1-E_2\right)}{\eta},\text{其中}\eta=K_{11}+K_{22}-2K_{12}$$

- 剪辑后：

$$\alpha_2^{new} = \begin{cases} H, \alpha_2^{new,unc} > H \\ \alpha_2^{new,unc}, L \le \alpha_2^{new,unc} \le H \\ L, \alpha_2^{new,unc} < L \end{cases}$$

由α_2^{new}求得α_1^{new}为:

$$\alpha_1^{new} = \alpha_1^{old} + y_1 y_2 (\alpha_2^{old} - \alpha_2^{new})$$

(4) 阈值/偏差 b更新公式

$$b_1^{new} = b^{old} - E_1 - y_1 K_{11} (\alpha_1^{new} - \alpha_1^{old}) - y_2 K_{21} (\alpha_2^{new} - \alpha_2^{old})$$

$$b_2^{new} = b^{old} - E_1 - y_1 K_{12} (\alpha_1^{new} - \alpha_1^{old}) - y_2 K_{22} (\alpha_2^{new} - \alpha_2^{old})$$

$$b^{new} = \begin{cases} b_1^{new}, \ 0 \le \alpha_1^{new} \le C \\ b_2^{new}, \ 0 \le \alpha_2^{new} \le C \\ \frac{b_1 + b_2}{2}, \ otherwise \end{cases}$$

(5) w*计算公式

$$w^* = \sum_{i=1}^{m} \alpha_i^* y_i x_i$$

③ 代码实现

> **具体代码已经已经给出.py文件，只需写好调用函数即可，下面粗略分析给出的SMO算法，部分代码省略**

(1)数据预处理

算法介绍：对提供的训练文件进行读取为矩阵，并将标签提取出来

> **数据整理**：对于缺失值的情况，作业中的数据已经将缺失值全部补0了，根据设计目标中的数据集讲解可得，设为0即可不影响后续。

- **数据读取函数：**

```
def loadData(filename):
    dataDict = loadmat(filename)
    return dataDict['X'], dataDict['y']
```

- ◦ **代码细节**：

```
"""
输入：
 数据集路径
输出：
 numpy.array格式的X，y数据array
 X为m×n的数据array，m为样例数，n为特征维度
 y为m×1的标签array，1表示正例，0表示反例
"""
```

1) from scipy.io import loadmat,使用loadmat读取.mat文件

2) loadmat返回值是一个字典形式，使用dataDict['X'], dataDict['y']返回特征和标签

- **数据可视化函数：**

```
def plotData(X, y, title=None):
    plt.rcParams['font.sans-serif'] = ['KaiTi']      # 指定默认字体
    plt.rcParams['axes.unicode_minus'] = False       # 解决保存图像是负号'-'显示为方块
的问题
    .......
    plt.show()
```

- ○ **代码细节：**

  1) import matplotlib.pyplot as plt，使用pyplot库函数将读取的数据绘制即可

  2) plt.rcParams['font.sans-serif'] = ['KaiTi'] 是为了解决图像无法正常显示中文，指定默认字体为楷体
  plt.rcParams['axes.unicode_minus'] = False 是为了解决保存图像是负号'-'显示为方块的问题

(2)SMO算法实现SVM

> 具体代码解析，参考作业给出的ppt文件，下列只叙述自己认为的细节之处

算法介绍：使用SMO算法实现SVM，对处理好的数据，计算出超平面相关参数

- **sigmoid函数**

```
def svmTrain_SMO(X, y, C, kernelFunction='linear', tol=1e-3, max_iter=5,
**kargs):
    start = time.clock()

   ...........

    idx = np.where(alphas > 0)
    model = {'X': X[idx[0], :], 'y': y[idx], 'kernelFunction':
str(kernelFunction),
             'b': b, 'alphas': alphas[idx], 'w': (np.multiply(alphas, y). T *
X).T}
    return model
```

- ○ **代码细节：**

```
"""
利用简化版的SMO算法训练SVM
输入：
X，y为loadData函数的返回值    C为惩罚系数
kernelFunction表示核函数类型，对于非线性核函数，也可直接输入核函数矩阵K
tol为容错率   max_iter为最大迭代次数
输出：
model['kernelFunction']为核函数类型   model['X']为支持向量
model['y']为对应的标签   model['alpha']为对应的拉格朗日参数
model['w'], model['b']为模型参数
"""
```

1) 代码实现下列的伪代码，具体公式参考①②中叙述的算法原理公式，这里不再叙述，
跳转回SMO算法原理
主要是 对偶问题的求解，并根据KKT条件更新参数

```
简化版SMO算法
1.Initialize alphas向量为0
2.    while 迭代次数小于最大迭代次数:
3.        for each alpha[i] in alphas:
4.            if alpha[i]可优化:
5.                随机选择另一个alpha[j],同时优化这两个向量。
6.            else :
7.                退出本次内循环。
8.        if 所有的alphas都没有被优化:
9.            增加迭代次数，进行下一次外循环。
```

2) 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。

3) while iters < max_iter: 使用while循环进行不断迭代更新参数，同时对于不满足KKT条件等参数不需要更新的循环直接使用continue结束内循环。

- **线性SVM边界可视化函数**:

```
def visualizeBoundaryLinear(X, y, model, title=None):
    plt.rcParams['font.sans-serif'] = ['KaiTi']     # 指定默认字体
    plt.rcParams['axes.unicode_minus'] = False      # 解决保存图像是负号'-'显示为方块的问题
    ......

    plt.show()
```

- ◦ **代码细节**:

  1) import matplotlib.pyplot as plt，使用pyplot库函数将读取的数据绘制即可

  2) 根据$y = w^T x + b$ 计算边界代码如下:

  ```
  xp = np.linspace(min(X[:, 0]), max(X[:, 0]), 100)
  yp = np.squeeze(np.array(-(w[0] * xp + b) / w[1]))
  ```

- **高斯核计算函数:**

> 对于非线性高斯核函数的SVM，编写高斯核，同时计算训练集在高斯核函数下的矩阵值

```
def gaussianKernelSub(x1, x2, sigma):
    x1 = np.mat(x1).reshape(-1, 1)
    x2 = np.mat(x2).reshape(-1, 1)
    n = -(x1 - x2).T * (x1 - x2) / (2 * sigma**2)
    return np.exp(n)


def gaussianKernel(X, sigma):
    start = time.clock()
    ......
    for i in range(m):
        if dots % 10 == 0:
```

```
            print('.', end='')
        dots = dots + 1
        if dots > 780:
            dots = 0
            print()
        for j in range(m):
            K[i, j] = gaussianKernelSub(X[i, :].T, X[j, :].T, sigma)
            K[j, i] = K[i, j].copy()
    .....
    return K
```

- ◦ **代码细节**:

  1) 高斯核直接用代码实现公式即可n = -(x1 - x2).T * (x1 - x2) / (2 * sigma**2)

  2) 计算高斯核的矩阵使用公式K[i, j] = gaussianKernelSub(X[i, :].T, X[j, :].T, sigma)
  调用高斯核数学公式计算对应位置的值即可

- **结果预测分类器函数:**

```
def svmPredict(model, X, *arg):
    ......
    if model['kernelFunction'] == 'linear':
        p = X * model['w'] + model['b']
    else:
        for i in range(m):
            prediction = 0
            for j in range(model['X'].shape[0]):
                prediction += model['alphas'][:, j] * model['y'][:, j] * \
                    gaussianKernelSub(X[i, :].T, model['X'][j, :].T, *arg)
            p[i] = prediction + model['b']
    pred[np.where(p >= 0)] = 1
    pred[np.where(p < 0)] = 0
    return pred
```

- ◦ **代码细节**:

```
"""
利用得到的model, 计算给定X的模型预测值
输入:
model为svmTrain_SMO返回值
X为待预测数据
sigma为训练参数
"""
```

(1) 根据传入的SVM模型的核函数类型进行不同的计算(线性或者高斯型)

线性$y = w^T x + b$, 高斯核$f(x) = \sum_{i=1}^{m} \alpha_i y_i K(x_i * x_j) + b$

代码如下:

```
线性: p = X * model['w'] + model['b']
高斯: prediction += model['alphas'][:, j] * model['y'][:, j] * \
                    gaussianKernelSub(X[i, :].T, model['X'][j, :].T, *arg)
```

(2) pred[np.where(p >= 0)] = 1大于零的位置为正例设为1； pred[np.where(p < 0)] = 0,小于零的位置为反例设为0

- **高斯SVM分类边界可视化函数：**

```
def visualizeBoundaryGaussian(X, y, model, sigma, title=None):

    .......

    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)
    print('Done')

    .........

    plt.show()
```

- ◦ **代码细节**：

  (1) 调用分类器函数将分类边界计算出来： vals[:, i] = svmPredict(model, this_X, sigma)

  (2) import matplotlib.pyplot as plt，使用pyplot库函数将读取的数据绘制即可

(3)main函数

算法介绍：调用上述编写的算法函数

```
import SVM_Functions as s

if __name__ == "__main__":
    X_lin, y_lin = s.loadData("task1/task1_linear.mat")
    s.plotData(X_lin, y_lin, title="linear图像")
    model1 = s.svmTrain_SMO(X_lin, y_lin, C=1, max_iter=20)
    s.visualizeBoundaryLinear(X_lin, y_lin, model1, title='linear_SVM边界图')

    X_gauss, y_gauss = s.loadData("task1/task1_gaussian.mat")
    s.plotData(X_gauss, y_gauss, title="gaussian图像")
    model2 = s.svmTrain_SMO(X_gauss, y_gauss, C=1, kernelFunction='gaussian',
K_matrix=s.gaussianKernel(X_gauss, sigma=0.1))
    s.visualizeBoundaryGaussian(X_gauss, y_gauss, model2, sigma=0.1,
title='gaussian_SVM边界图')
```

- **代码细节**：

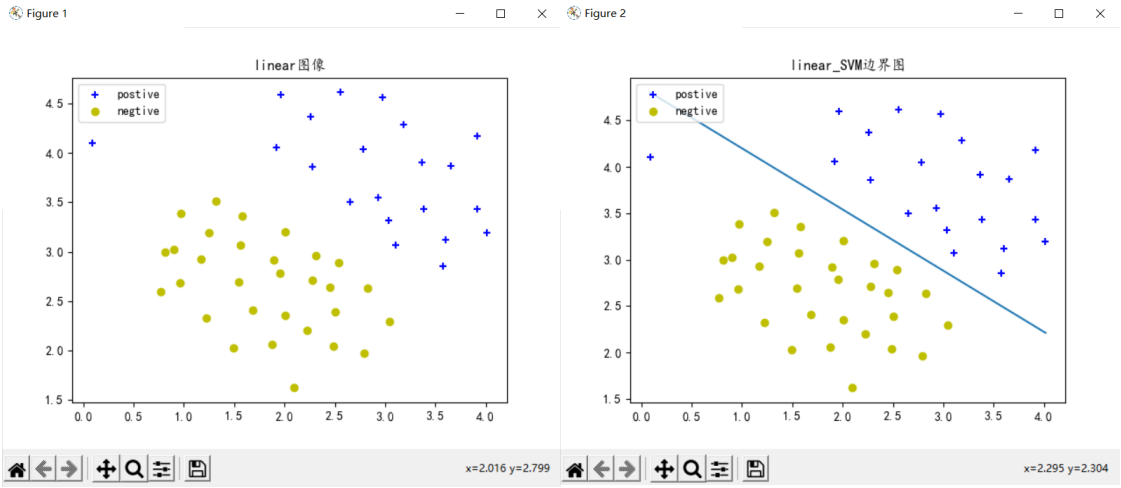
  (1) 调用给出的SVM_Functions.py中的函数，注意函数参数的对应，线性和高斯核对应参数不一致

二、运行结果

截图一_运行结果，终端显示

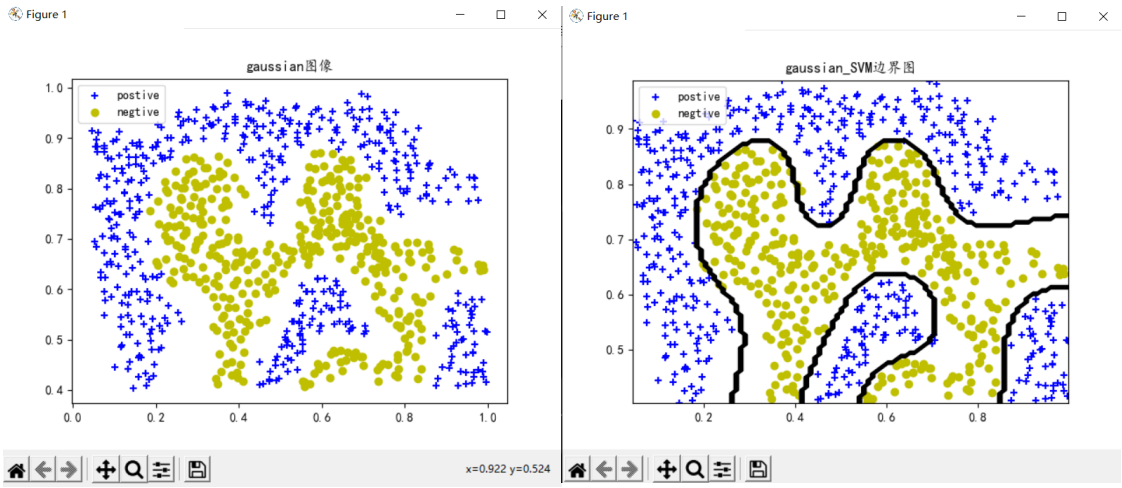
```
PS F:\own_file\课程文件\机器学习导论\支持向量机\2021年春季_第7章_SVM_编程作业> python -u "f:\ow
季_第7章_SVM_编程作业\task1\task1.py"
Training ...........................................................................
...........................................................................
..Done( 0.7634035000000026s )

GaussianKernel Computing ..........................................................
.....................................Done( 31.834170400000005s )

Training ...........................................................................
...........................................................................
...........................................................................
...........................................................................
.....................................................Done( 26.927977s )

Predicting .........................................................................
.....................................Done
PS F:\own_file\课程文件\机器学习导论\支持向量机\2021年春季_第7章_SVM_编程作业>
```

截图一_运行结果，线性核



- 图片分析：上图为线性核情况下原始数据图以及边界可视化图，由上图显示，边界正确

截图二_运行结果，高斯核



- 图片分析：上图为高斯核情况下原始数据图以及边界可视化图，由上图显示，边界正确

三、总结体会

程序设计心得:

1. 本次算法过程中，遇到程序错误在正常不过，知错能改，善莫大焉，一个完整的工程背后一定是一次次的debug，自身犯错最多的地方在于差错处理，可以多考虑极端情况，多进行程序的调试，对每一种特殊情况进行处理，避免程序运行的自行结束。通过这次软件课设，自身养成了不惧困难，不退缩，提高思维能力
2. 软件工程量较大，建议采用模块化设计，同时不同的内容可以分为不同的文件，优先设计函数头，确定参数及返回值后具体设计逻辑
3. SVM算法的构建，重点在于对于数学公式的理解，遇到难以解决的问题可以先单步调试，找到问题一步步修改

TASK2: 使用高斯核SVM对给定数据集进行分类

一、软件设计

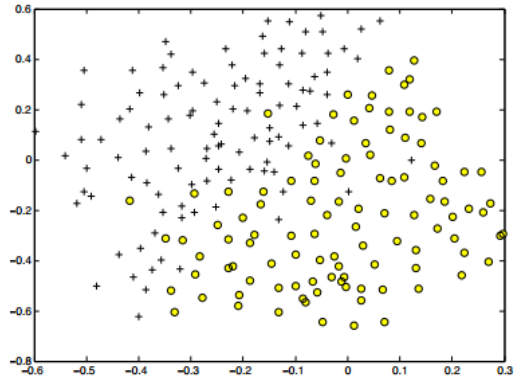
1. 设计目标

① 数据集讲解:

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。
（程序运行时间8mins左右，准确率 = 预测正确样本数/样本总数 ）

② 设计提示思路

1. 数据可视化:



2. 程序运行结果举例：由于SMO算法的随机性，你的结果应该跟下面的例子不完全相同：

```
Training Accuracy:
sigma   0.01   0.03    0.1    0.3      1      3     10     30
C
0.01    0.498  0.502  0.498  0.498  0.498  0.820  0.502  0.502
0.03    0.502  0.502  0.521  0.867  0.498  0.829  0.498  0.810
0.1     0.498  0.502  0.948  0.867  0.834  0.502  0.498  0.498
0.3     0.502  0.981  0.948  0.900  0.867  0.649  0.806  0.502
1       1.000  0.995  0.948  0.934  0.905  0.858  0.502  0.498
3       1.000  1.000  0.943  0.943  0.919  0.863  0.796  0.498
10      1.000  1.000  0.962  0.948  0.924  0.891  0.853  0.498
30      1.000  1.000  0.948  0.938  0.929  0.924  0.867  0.735
```

③ 目标要求

- 编写实验报告，报告中需要包含：实验设置，编程思路，实验结果展示以及自己的思考等。
- 其中实验结果展示按照上图的格式在实验报告中给出。

2. 算法设计分析

> **所用项目环境 IDE：VS code　　Python版本：python3.7.5**

①SVM/SMO算法原理

这部分的原理及代码实现在task1中已经详细描述过，这里不再重复叙述
跳转回SMO算法代码实现

②代码实现

> SVM的详细实现代码跳转到task1阅读即可，下列是基于task1中的SMO算法使用

（1）子模块

- **SVM模型精度计算算法：**对于参数c与sigma不同的组合，得出不同的SVM模型model并计算精度

```
def predict_rate(sigmaList, CList):
    # error_num = 0
    index = 0
    X_gauss, y_gauss = s.loadData("task2/task2.mat")
    rate_list = [[] for i in range(len(CList))]
    # s.plotData(X_gauss, y_gauss, title="gaussian图像")
    for c in CList:
        for Sigma in sigmaList:
            model = s.svmTrain_SMO(X_gauss, y_gauss, C=c,
kernelFunction='gaussian', K_matrix=s.gaussianKernel(X_gauss, sigma=Sigma))
            pre = s.svmPredict(model, X_gauss, Sigma)
            rate = 1 - (np.sum(abs(np.array(pre) - np.array(y_gauss))) /
float(len(y_gauss)))
            rate_list[index].append(('%.3f' % rate))
        # print(rate_list[index])
        index += 1
    return rate_list
```

- ◦ **代码细节**：

  1) 使用两层for循环，对于参数c与sigma不同的组合，调用SVM算法

  2) 标签只有1,0两种，计算精度使用下列代码：
  rate = 1 - (np.sum(abs(np.array(pre) - np.array(y_gauss))) / float(len(y_gauss)))
  计算预测值与真实值两个数组之间的差的绝对值并求和得出，预测值与真实值不同的个数，求和个数与总标签数相除得到错误率，1-错误率=精度

- **列表打印算法：**将传入的列表打印成任务要求中的对齐格式

```
def List_print(List):
    for i in range(len(List)):
        if i == (len(List) - 1):
            print(List[i])
        else:
            print(List[i], end='\t')
```

- ○ **代码细节**：代码简单，使用for循环，每一个print使用参数end='\t',当打印列表最后一个元素的时候默认以换行结束

(2) main函数

```
if __name__ == "__main__":

    sigmaList = [0.01, 0.03, 0.1, 0.3, 1, 3, 10, 30]
    CList = [0.01, 0.03, 0.1, 0.3, 1, 3, 10, 30]
    rate_list = predict_rate(sigmaList, CList)
    print('Training Accuracy:\n' + 'sigma', end='\t')
    List_print(sigmaList)
    print('c')
    for i in range(len(rate_list)):
        print(CList[i], end='\t')
        List_print(rate_list[i])
```

- ○ **代码细节**：

(1) 调用之前设计的数个函数即可

(2) 开始打印print('Training Accuracy:\n' + 'sigma', end='\t')； List_print(sigmaList)打印最开始的提示信息；对于返回的二维精度列表，循环打印每一行。

二、运行结果

- **采用Google colab运行代码**

```
GaussianKernel Computing .....................Done( 6.335259999999835s )

Training ..............................................Done( 1.9721180000005916s )

GaussianKernel Computing .....................Done( 6.13982599999963s )

Training ..............................................Done( 2.1284799999993993s )

Training Accuracy:
sigma   0.01    0.03    0.1     0.3     1       3       10      30
c
0.01    0.502   0.502   0.502   0.502   0.825   0.498   0.502   0.502
0.03    0.498   0.502   0.502   0.867   0.502   0.810   0.502   0.498
0.1     0.498   0.502   0.948   0.863   0.825   0.498   0.829   0.502
0.3     0.502   0.976   0.948   0.905   0.867   0.815   0.825   0.502
1       1.000   0.995   0.948   0.934   0.905   0.858   0.502   0.502
3       1.000   1.000   0.943   0.943   0.919   0.867   0.692   0.502
10      1.000   1.000   0.962   0.948   0.924   0.900   0.825   0.829
30      1.000   1.000   0.948   0.938   0.929   0.929   0.872   0.716
```

- **自己电脑运行代码：**

```
GaussianKernel Computing .......................Done( 1.6520464999999983s )

Training ..........................Done( 0.44537719999999958s )

Training Accuracy:
sigma   0.01    0.03    0.1     0.3     1       3       10      30
C
0.01    0.502   0.502   0.502   0.502   0.820   0.498   0.502   0.787
0.03    0.498   0.498   0.502   0.867   0.502   0.502   0.502   0.498
0.1     0.498   0.498   0.948   0.863   0.839   0.502   0.502   0.502
0.3     0.502   0.986   0.948   0.896   0.872   0.806   0.502   0.825
1       1.000   0.995   0.948   0.934   0.910   0.848   0.498   0.502
3       1.000   1.000   0.943   0.943   0.924   0.877   0.521   0.498
10      1.000   1.000   0.962   0.943   0.924   0.900   0.853   0.502
30      1.000   1.000   0.948   0.948   0.924   0.924   0.872   0.654
PS F:\own_file\课程文件\机器学习导论\支持向量机\2021年春季_第7章_SVM_编程作业>
```

- ○ 实验结果分析：

  由上面两个运行结果截图，与任务要求中的示例(下)对比知：程序设计成功，输出结果大致是相同的

```
Training Accuracy:
sigma   0.01   0.03    0.1    0.3      1      3     10     30
C
0.01   0.498  0.502  0.498  0.498  0.498  0.820  0.502  0.502
0.03   0.502  0.502  0.521  0.867  0.498  0.829  0.498  0.810
0.1    0.498  0.502  0.948  0.867  0.834  0.502  0.498  0.498
0.3    0.502  0.981  0.948  0.900  0.867  0.649  0.806  0.502
1      1.000  0.995  0.948  0.934  0.905  0.858  0.502  0.498
3      1.000  1.000  0.943  0.943  0.919  0.863  0.796  0.498
10     1.000  1.000  0.962  0.948  0.924  0.891  0.853  0.498
30     1.000  1.000  0.948  0.938  0.929  0.924  0.867  0.735
```

  具体分析知：不同的sigma和c，精度结果会出现较大差异，下面四行的精度明显比上四行好很多；

  对于参数c，随着图中8个c的不同取值增大，精度也会越来越好；

  对于参数sigma，随着值的增大，大概呈一种先上升后下降的趋势。

三、总结体会

程序设计心得:

1. 本次算法过程中，遇到程序错误在正常不过，知错能改，善莫大焉，一个完整的工程背后一定是一次次的debug，自身犯错最多的地方在于差错处理，可以多考虑极端情况，多进行程序的调试，对每一种特殊情况进行处理，避免程序运行的自行结束。通过这次软件课设，自身养成了不惧困难，不退缩，提高思维能力
2. 本次代码编写过程中，由于task1中已经实现SMO算法，故重点在于对于该算法函数的调用，注意参数的设置，编写对应的格式化输出函数即可

TASK2: 使用线性SVM实现对垃圾邮件分类

一、软件设计

1. 设计目标

① 数据集讲解：

- 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。（程序运行时间10mins左右）

② 设计提示思路

1. 对SMO算法的实现进行举例说明:

```
简化版SMO算法
1.Initialize alphas向量为0
2.    while 迭代次数小于最大迭代次数:
3.        for each alpha[i] in alphas:
4.            if alpha[i]可优化:
5.                随机选择另一个alpha[j],同时优化这两个向量。
6.            else :
7.                退出本次内循环。
8.        if 所有的alphas都没有被优化:
9.            增加迭代次数，进行下一次外循环。
```

③ 目标要求

- 编写实验报告，报告中需要包含：实验设置，编程思路等
- 将测试数据预测结果按顺序存储为txt文件，每一行为一个样本的标签，上传到系统对应作业栏。

2. 算法设计分析

> **所用项目环境 IDE：VS code　　Python版本：python3.7.5**

①SVM/SMO算法原理

这部分的原理及代码实现在task1中已经详细描述过，这里不再重复叙述

跳转回SMO算法代码实现

②代码实现

> SVM的详细实现代码跳转到task1阅读即可，下列是基于task1中的SMO算法具体使用代码

（1）子模块

- **数据读取函数_修改：**task1中的数据读取返回特征和标签，而测试集中不含有标签故修改读取函数

```
def loadData(filename):
    dataDict = loadmat(filename)
    if 'y' in dataDict.keys():
        return dataDict['X'], dataDict['y']
    else:
        return dataDict['X']
```

- ◦ **代码细节：**判断是否存在标签y，若存在返回特征x和标签y；不存在则只返回特征x

- **分类器预测算法：**

```
# 调用sklearn库函数
def predict_sklearn(x, y, x_test):
    x_train, x_validation, y_train, y_validation = train_test_split(x, y,
test_size=0.2)
    svm_clf = svm.LinearSVC()
```

```
    svm_clf.fit(x_train, y_train)
    acc = svm_clf.score(x_validation, y_validation)
    print('验证集精度为: %.4f' % acc)

    # svm_clf.fit(x, y)
    dec = svm_clf.predict(x_test)
    np.savetxt('task3.txt', dec, fmt="%d")

# 调用task1中的算法，并划分验证集
def predict_rate_valid(x, y, x_test):
    x_train, x_validation, y_train, y_validation = train_test_split(x, y,
test_size=0.2)
    model = s.svmTrain_SMO(x_train, y_train, C=1, tol=1e-3, max_iter=20)

    pre_train = s.svmPredict(model, x_train)
    pre_validation = s.svmPredict(model, x_validation)
    y_test = s.svmPredict(model, x_test)

    rate1 = 1 - (np.sum(abs(np.array(pre_train) - np.array(y_train))) /
float(len(y_train)))
    rate2 = 1 - (np.sum(abs(np.array(pre_validation) - np.array(y_validation)))
/ float(len(y_validation)))
    # rate1 = ('%.4f' % rate1)
    # rate2 = ('%.4f' % rate2)
    print('用训练集20%%划分为验证集，训练集模型准确度为：%.4f，验证集准确度为：%.4f' %
(rate1, rate2))

    txtsave('task3/task3_testlabels2.txt', y_test)
    return rate1, rate2

#调用task1中的算法，不划分验证集
def predict_rate(x, y, x_test):
    model = s.svmTrain_SMO(x, y, C=1, tol=1e-3, max_iter=20)
    pre_train = s.svmPredict(model, x)
    y_test = s.svmPredict(model, x_test)

    rate = 1 - (np.sum(abs(np.array(pre_train) - np.array(y))) / float(len(y)))
    # rate = ('%.4f' % rate)
    print('训练集精度为: %.4f' % rate)

    txtsave('task3/task3_testlabels1.txt', y_test)
    return rate
```

- ◦ **代码细节**:

  1) 含有三个函数,分别作用如下:
  predict_sklearn(): 调用sklearn库函数svm;
  predict_rate_valid(): 调用task1中的svm算法，并划分验证集；
  predict_rate():调用task1中的svm算法，不划分验证集

  2) 采用rate = 1 - (np.sum(abs(np.array(pre_train) - np.array(y))) / float(len(y)))计算精度，与task2中方法相同
  sklearn中采用acc = svm_clf.score(x_validation, y_validation)计算精度

  3) 采用from sklearn.model_selection import train_test_split划分数据集

- **数据保存算法：**将预测的标签值保存为txt文件

```
def txtsave(filename, data):  # filename为写入txt文件的路径，data为要写入数据列表.
    file = open(filename, 'w')
    for i in range(len(data)):
        s = str(data[i]).replace('[', '').replace(']', '').replace(',', '\t')  # 去除[]
        s = s.replace("'", '').replace(',', '').replace(',', '\t') + '\n'  # 去除单引号，逗号，每行末尾追加换行符
        file.write(s)
    file.close()
    print(filename, "保存文件成功")
```

- ○ **代码细节**：代码简单，使用python文件操作，注意使用.replace()去除不需要的符号

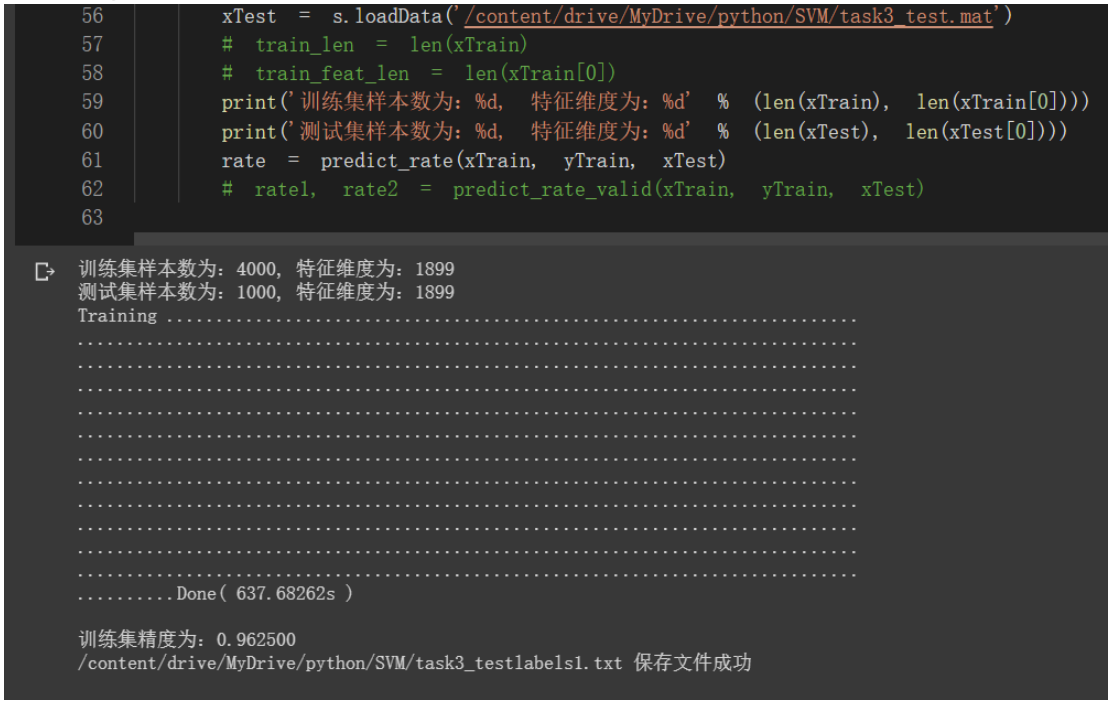
(2) main函数

```
if __name__ == '__main__':
    xTrain, yTrain = s.loadData('task3/task3_train.mat')
    xTest = s.loadData('task3/task3_test.mat')
    # train_len = len(xTrain)
    # train_feat_len = len(xTrain[0])
    print('训练集样本数为：%d, 特征维度为：%d' % (len(xTrain), len(xTrain[0])))
    print('测试集样本数为：%d, 特征维度为：%d' % (len(xTest), len(xTest[0])))
    rate = predict_rate(xTrain, yTrain, xTest)
    rate1, rate2 = predict_rate_valid(xTrain, yTrain, xTest)
    predict_sklearn(xTrain, yTrain, xTest)
```

- ○ **代码细节**：调用之前设计的数个函数即可

二、运行结果

- **采用Google colab运行代码**(调用手写的svm算法)



```
+ 代码  + 文本
49
50          txtsave('/content/drive/MyDrive/python/SVM/task3_testlabels1.txt',  y_test)
51          return  rate
52
53
54  if  __name__  ==  '__main__':
55          xTrain,  yTrain  =  s.loadData('/content/drive/MyDrive/python/SVM/task3_train.mat')
56          xTest  =  s.loadData('/content/drive/MyDrive/python/SVM/task3_test.mat')
57          #  train_len  =  len(xTrain)
58          #  train_feat_len  =  len(xTrain[0])
59          print('训练集样本数为：%d,  特征维度为：%d'  %  (len(xTrain),  len(xTrain[0])))
60          print('测试集样本数为：%d,  特征维度为：%d'  %  (len(xTest),  len(xTest[0])))
61          rate  =  predict_rate(xTrain,  yTrain,  xTest)
62          rate1,  rate2  =  predict_rate_valid(xTrain,  yTrain,  xTest)
63
训练集样本数为：4000, 特征维度为：1899
测试集样本数为：1000, 特征维度为：1899
Training ..........
..........
..........
..........
..........
..........
..........
..........
..........
..........
..........Done( 397.5071230000001s )

训练集精度为：0.960000
```

- **自己电脑运行代码：**（sklearn库函数）

```
PS F:\own_file\课程文件\机器学习导论\支持向量机\2021年春季_第7章_SVM_编程作业> python -u "f:\own
季_第7章_SVM_编程作业\task3\test3.py"
训练集样本数为：4000, 特征维度为：1899
测试集样本数为：1000, 特征维度为：1899
C:\Users\22986\AppData\Local\Programs\Python\Python37\lib\site-packages\sklearn\utils\validation
y was passed when a 1d array was expected. Please change the shape of y to (n_samples, ), for ex
  return f(*args, **kwargs)
验证集精度为：0.9675
PS F:\own_file\课程文件\机器学习导论\支持向量机\2021年春季_第7章_SVM_编程作业> python -u "f:\own
季_第7章_SVM_编程作业\task3\test3.py"
训练集样本数为：4000, 特征维度为：1899
测试集样本数为：1000, 特征维度为：1899
C:\Users\22986\AppData\Local\Programs\Python\Python37\lib\site-packages\sklearn\utils\validation
y was passed when a 1d array was expected. Please change the shape of y to (n_samples, ), for ex
  return f(*args, **kwargs)
C:\Users\22986\AppData\Local\Programs\Python\Python37\lib\site-packages\sklearn\svm\_base.py:986
rge, increase the number of iterations.
  "the number of iterations.", ConvergenceWarning)
验证集精度为：0.9812
PS F:\own_file\课程文件\机器学习导论\支持向量机\2021年春季_第7章_SVM_编程作业>
```

- ◦ **实验结果分析：上图结果都是多次调参运行后的最好结果**
  1. 调用手写的SVM算法运行时间较长大概10分钟左右，精度为0.96左右（会随着参数改变而改变
  2. 调用sklearn库函数运行时间很短(几秒钟)，精度大约为0.97（会随着参数改变而改变

三、总结体会

程序设计心得:

1. 本次算法过程中，遇到程序错误在正常不过，知错能改，善莫大焉，一个完整的工程背后一定是一次次的debug，自身犯错最多的地方在于差错处理，可以多考虑极端情况，多进行程序的调试，对每一种特殊情况进行处理，避免程序运行的自行结束。通过这次软件课设，自身养成了不惧困难，不退缩，提高思维能力
2. 本次代码编写过程中，由于task1中已经实现SMO算法，故重点在于对于该算法函数的调用，注意参数的设置，编写对应的格式化输出函数即可；也可以调用sklearn库函数中的svm相关包实现。