Homework 05

一、 实验目标：

实现 SVM 算法

二、 实验过程：

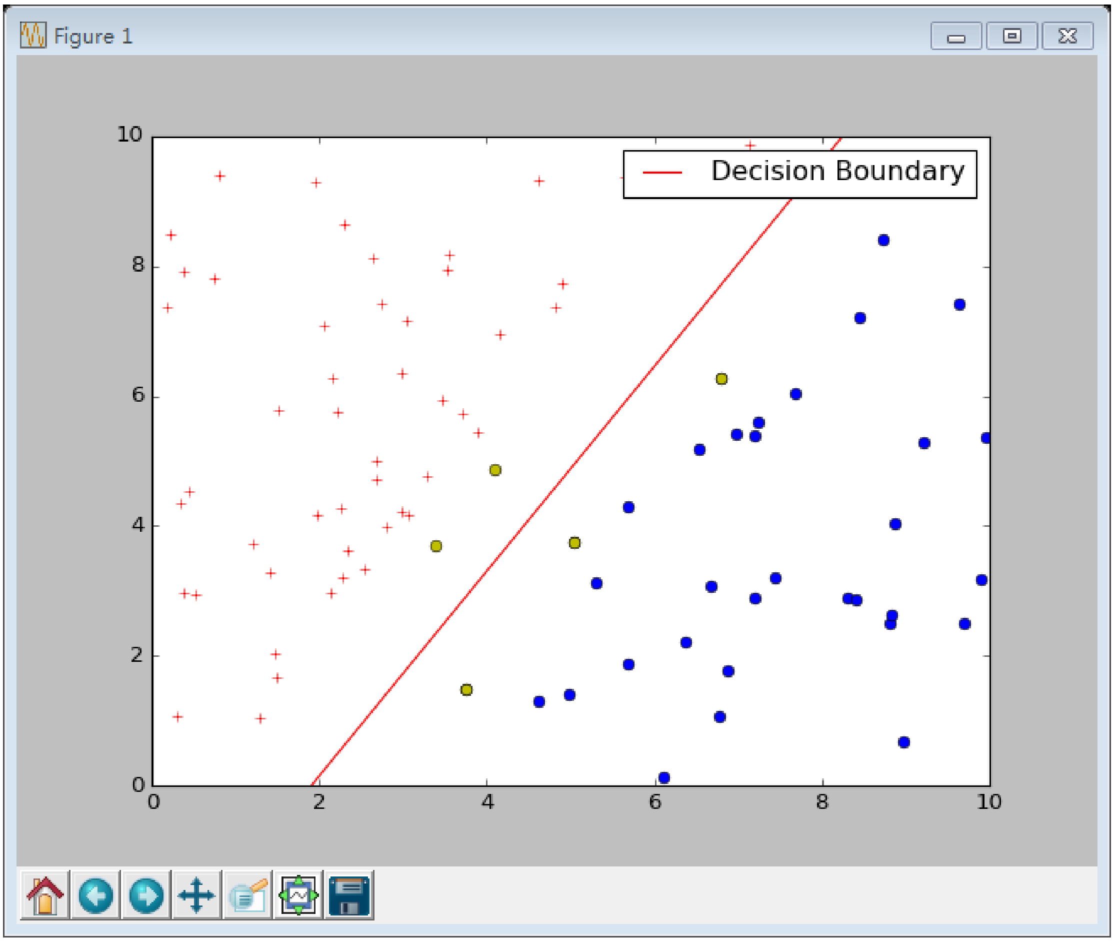
主要采用简化 SMO 算法进行处理

第一步，生成 2D 数据，编写了 GenerateData.py 用于生成数据集 DataSet.txt

第二步，读取 DataSet.txt，将 2D 点的横纵坐标保存到 dataSet 中，将各个点的 label 保存到 label 中，对 label 进行转置。

第三步，采用 Sequential Minimal Optimization，用于训练 SVM。初始化 alphas 和 b 两个参数，开始迭代过程。首先，对于数据集中每一个坐标点，带入当前 alphas 和 b，如果与实际 label 的误差在允许范围内则跳过；如果超出允许范围，那么就可以对该数据点所对应的 alpha 值进行优化。为了确定进行优化的 alpha 对，随机选择一个 alpha 值，对该 alpha 值同样计算出误差。确保 alpha 在 0 到 C 之间后，计算出随机 alpha 值的最优修改量，再对对应的 alpha 值进行修改，修改量相同，方向相反，重新计算常数项 b 后，该数据点完成，继续循环。当不发生任何 alpha 修改之后，完成迭代过程。

三、 实验结果：



实验结果如图所示，图中包括以黄色点标记的支持向量和分割超平面。所有positive samples 都标记为红色十字，negative samples 标记为蓝色原点显示在图中，分类效果良好。

四、 实验代码及注释：

```
import numpy as np
import matplotlib.pyplot as plt

#using simplified sequential minimal optimization
def smo(dataMatrix, label, C, tolerate, iterationMax):
    iterator = 0
    b = 0
    num, dim = np.shape(dataMatrix)
    alphas = np.mat(np.zeros((num, 1)))
    while iterator < iterationMax:
        #initialize alphaPairsChanged which stores whether alpha has been
```

```
optimized
        alphaPairsChanged = 0
        for i in range(num):
            #the class we anticipate
            label_i = float(np.multiply(alphas, label).T * (dataMatrix * dataMatrix[i, :].T)) + b
            #calculate deviation
            deviation_i = label_i - float(label[i])
            #if deviation is too big, start optimization
            if((label[i] * deviation_i < -tolerate) and (alphas[i] < C) or (label[i] * deviation_i > tolerate) and (alphas[i] > 0)):
                #select a random alpha index
                r = i
                while r == i:
                    r = int(np.random.uniform(0, num))
                #calculate r's label and deviation
                label_r = float(np.multiply(alphas, label).T * (dataMatrix * dataMatrix[r, :].T)) + b
                deviation_r = label_r - float(label[r])
                alpha_i_old = alphas[i].copy()
                alpha_r_old = alphas[r].copy()
                #ensure alpha[r] is between 0 and C
                if label[i] != label[r]:
                    L = max(0, alphas[r] - alphas[i])
                    H = min(C, C + alphas[r] - alphas[i])
                else:
                    L = max(0, alphas[r] + alphas[i] - C)
                    H = min(C, alphas[r] + alphas[i])
                if L == H:
                    continue
                eta = 2.0 * dataMatrix[i, :] * dataMatrix[r, :].T - dataMatrix[i, :] * dataMatrix[i, :].T - dataMatrix[r, :] * dataMatrix[r, :].T
                if eta >= 0:
                    continue
                alphas[r] -= label[r] * (deviation_i - deviation_r) / eta
                #adjust alpha if alpha is bigger than H or less than L
                if alphas[r] > H:
                    alphas[r] = H
                if L > alphas[r]:
                    alphas[r] = L
                if abs(alphas[r] - alpha_r_old) < 0.00001:
                    continue
                #modify i with same step but opposite direction compared to r
                alphas[i] += label[r] * label[i] * (alpha_r_old - alphas[r])
                b1 = b - deviation_i - label[i] * (alphas[i] - alpha_i_old) * dataMatrix[i, :]
* dataMatrix[i, :].T - label[r] * (alphas[r] - alpha_r_old) * dataMatrix[i, :] * dataMatrix[r, :].T
                b2 = b - deviation_r - label[i] * (alphas[i] - alpha_i_old) * dataMatrix[i, :]
* dataMatrix[r, :].T - label[r] * (alphas[r] - alpha_r_old) * dataMatrix[r, :] * dataMatrix[r, :].T
                if (0 < alphas[i]) and (C > alphas[i]):
                    b = b1
                elif (0 < alphas[r]) and (C > alphas[r]):
                    b = b2
                else:
                    b = (b1 + b2) / 2.0
                alphaPairsChanged = alphaPairsChanged + 1
        #only stop when no changes happen on alpha
        if alphaPairsChanged == 0:
            iterator = iterator + 1
        else:
            iterator = 0
    return b, alphas

#read data set, construct dataSet and label
dataSet = []
label = []
for line in open("DataSet.txt"):
    line = line.rstrip().split(" ")
    dataSet.append([float(line[0]), float(line[1])])
    label.append(float(line[2]))
dataMatrix = np.mat(dataSet)
label = np.mat(label).transpose()
num, dim = np.shape(dataMatrix)
b, alphas = smo(dataMatrix, label, 0.6, 0.01, 100)

#plot
plt.axis([0, 10, 0, 10])
for i in range(num):
    if label[i] == -1:
        plt.plot(dataMatrix[i, 0], dataMatrix[i, 1], 'r+')
    elif label[i] == 1:
        plt.plot(dataMatrix[i, 0], dataMatrix[i, 1], 'ob')
supportVectorsIndex = np.nonzero(alphas > 0)[0]
for i in supportVectorsIndex:
    plt.plot(dataMatrix[i, 0], dataMatrix[i, 1], 'oy')
w = np.zeros((2, 1))
for i in supportVectorsIndex:
    w += np.multiply(alphas[i] * label[i], dataMatrix[i, :].T)
margin = 2/np.sqrt(np.dot(w[1:3],w[1:3]))
```

```
min_x = min(dataMatrix[:, 0])[0, 0]
max_x = max(dataMatrix[:, 0])[0, 0]
y_min_x = float(-b - w[0] * min_x) / w[1]
y_max_x = float(-b - w[0] * max_x) / w[1]
plt.plot([min_x, max_x], [y_min_x, y_max_x], '-r', label = "Decision Boundary")
plt.legend()
plt.show()
```