山东大学 计算机科学与技术 学院

机器学习（双语） 课程实验报告

学号：	姓名：	班级：
实验题目：Experiment 5: SVM		
实验学时：4	实验日期： 2022/11/16	

实验目的：

1. 实现实验指导书中 SVM（支持向量机）的相关内容；
2. 学习使用 MATLAB、Python 等工具进行实验；
3. 通过选取拥有不同样本数量的训练集，尝试拟合在测试集上得到的结果，并对结果进行分析。

硬件环境：

Inter（R） Core（TM） i7-8750H

RAM：16.0 GB

软件环境：

Visual Studio Code

版本: 1.67.2 (user setup)

OS: Windows_NT x64 10.0.19044

Python 3.9.7

numpy 1.20.3

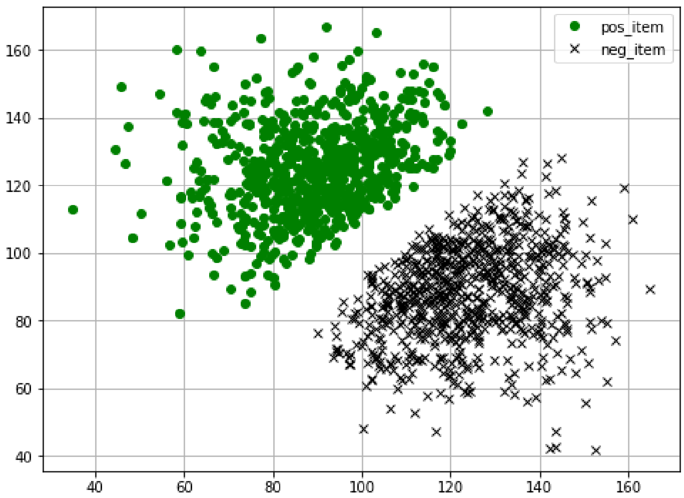
matplotlib 3.4.3

实验步骤与内容：

Dataset 1：

1. 首先加载数据集 1，并对其进行可视化：

```
training_1 = np.loadtxt('./data5/training_1.txt', dtype=np.double)  # 载入数据
test_1 = np.loadtxt('./data5/test_1.txt', dtype=np.double)
```



2. 通过求解正则化 SVM 的对偶问题来得到分离超平面：

Regularized SVM:

$$\begin{aligned} \min_{w,b,\xi} \quad & \frac{1}{2}||w||^2 + C\sum_{i=1}^{m}\xi_i \\ s.t. \quad & y^{(i)}(w^T x^{(i)} + b) \geq 1 - \xi_i, \forall i = 1, \ldots, m \\ & \xi_i \geq 0, \forall i = 1, \ldots, m \end{aligned}$$

对偶问题：

$$\begin{aligned} \max_{\alpha} \quad & \sum_{i=1}^{m}\alpha_i - \frac{1}{2}\sum_{i,j=1}^{m} y^{(i)}y^{(j)}\alpha_i\alpha_j < x^{(i)}, x^{(j)} > \\ s.t. \quad & 0 \leq \alpha_i \leq C, \forall i = 1, \ldots, m \\ & \sum_{i=1}^{m}\alpha_i y^{(i)} = 0 \end{aligned}$$

问题转变为：

$$\min_{\alpha} \quad \frac{1}{2}\sum_{i,j=1}^{m} y^{(i)}y^{(j)}\alpha_i\alpha_j < x^{(i)}, x^{(j)} > - \sum_{i=1}^{m}\alpha_i$$

由此求解：

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

$$b^* = y_j - \sum_{i=1}^{N} y_i\alpha_i^* < x_i, x_j >$$

分离超平面：

$$w^* \cdot x + b^* = 0$$

3. 求解海森矩阵：

```
def get_H(x, y):
    m, n = x.shape
    P = np.zeros((m, m))
    for i in range(m):
        for j in range(m):
            P[i, j] = np.dot(x[i], x[j]) * y[i] * y[j]
    return P
```

使用第三方库 qpsolver，cvxopt solver 对该凸优化问题进行求解：

```python
def solve_dual_1(x, y, C):
    m, n = x.shape
    P = get_H(x, y)
    q = np.ones(m) * (-1)
    G, h = None, None
    A = y.astype('float')
    b = np.zeros(1)
    lb = np.zeros(m)
    ub = np.ones(m) * C

    alpha = solve_qp(P, q, G, h, A, b, lb, ub, solver='cvxopt')
    # print(alpha)

    w = get_w(alpha, x, y)
    b = get_b(alpha, w, x, y, C)

    return w, b
```

其中，w 和 b 的计算：

```python
def get_w(alpha, x, y):
    w = np.zeros(x.shape)
    w[:, 0] = x[:, 0] * alpha * y
    w[:, 1] = x[:, 1] * alpha * y
    return np.sum(w, axis=0)
```

```python
def get_b(alpha, w, x, y, C):
    m, n = x.shape
    index = []
    for i in range(m):
        if alpha[i] > 0 and alpha[i] < C:
            index.append(i)
    index = np.array(index)

    x = x[index]
    y = y[index]

    return (y - np.dot(x, w)).sum() / len(index)
```

使用 w、b 对给定的 x 进行预测：

```python
def pred(w, b, x):
    return np.dot(x, w) + b
```

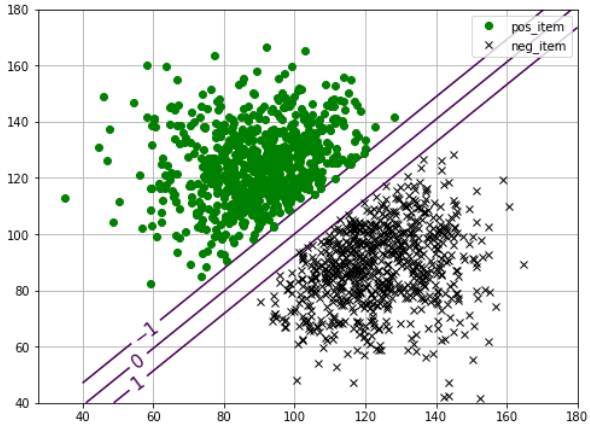
绘制等高线：

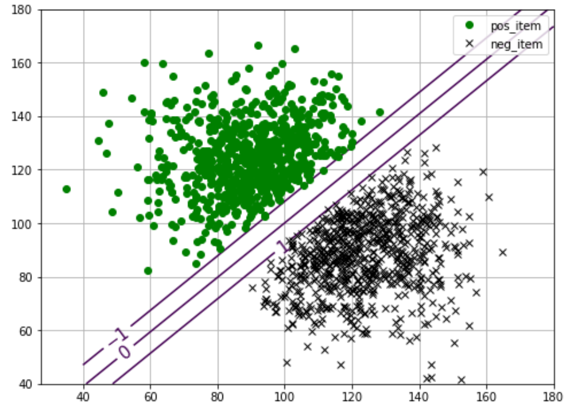
```python
def display_contour(w, b):
    x_1 = np.linspace(40, 180, 400)
    x_2 = np.linspace(40, 180, 400)
    p = np.zeros((len(x_1), len(x_2)))

    for i in range(len(x_1)):
        for j in range(len(x_2)):
            p[i][j] = pred(w, b, np.array((x_1[i], x_2[j])))

    contour_zero = plt.contour(x_1, x_2, p, [0.])
    contour_pos = plt.contour(x_1, x_2, p, [1.])
    contour_neg = plt.contour(x_1, x_2, p, [-1.])
    plt.clabel(contour_zero, inline=1, fontsize=15)
    plt.clabel(contour_pos, inline=1, fontsize=15)
    plt.clabel(contour_neg, inline=1, fontsize=15)
```

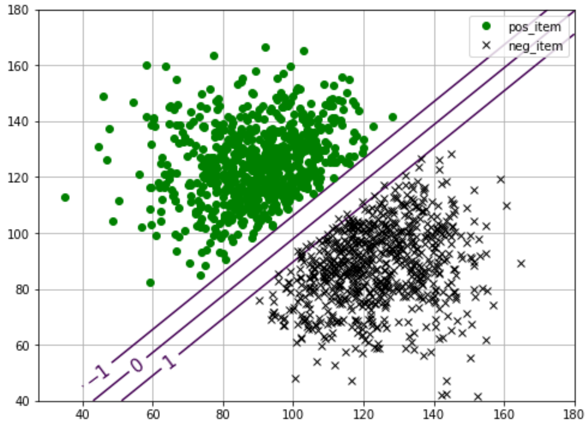
4. 分别对 C=1、0.1、0.01 的情况进行计算：

```
w_1_1, b_1_1 = solve_dual_1(train_x_1, train_y_1, 1)
w_1_2, b_1_2 = solve_dual_1(train_x_1, train_y_1, 0.1)
w_1_3, b_1_3 = solve_dual_1(train_x_1, train_y_1, 0.01)
```

显示出相应的分离平面：







此处变化不明显，但分离间隔应随着 C 的减小而增大。

5. 计算准确率：

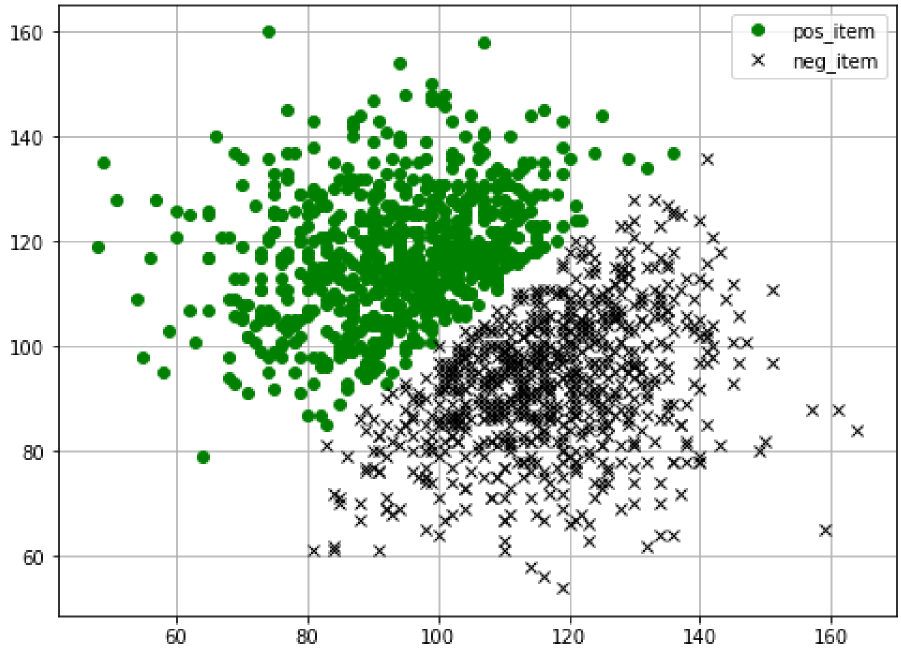
```
def get_accuracy(x, y, w, b, show_false_pos=False):
    m, n = x.shape

    p = pred(w, b, x)
    false_pos = []

    num = 0
    for i in range(m):
        if p[i] > 0 and y[i] == 1:
            num += 1
        elif p[i] < 0 and y[i] == -1:
            num += 1
        else:
            false_pos.append(i)

    if show_false_pos:
        print('false_pos:', false_pos[:10])

    return num / m
```

```
get_accuracy(test_x_1, test_y_1, w_1_1, b_1_1)
```

显然为 1：
```
1.0
```

Dataset2：

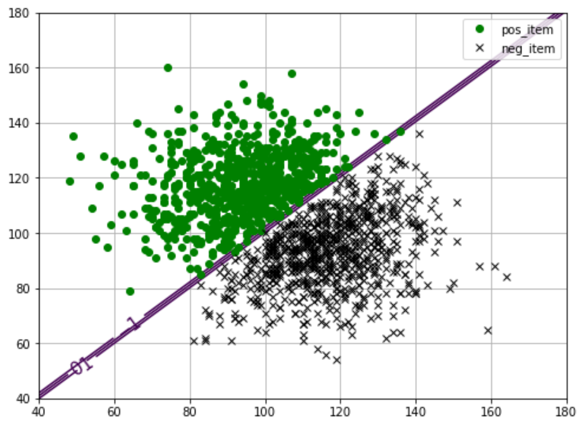
1. 同样载入数据，显示出 pos_item 及 neg_item：

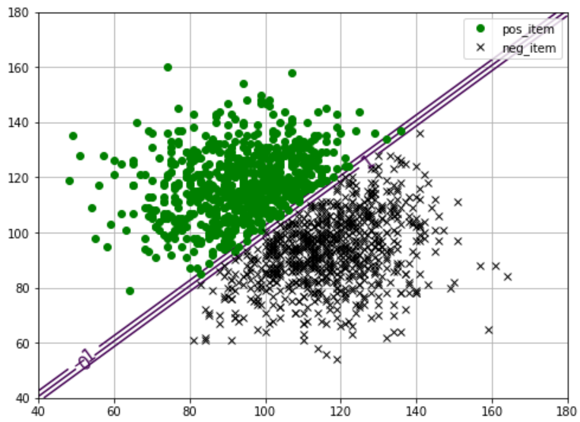


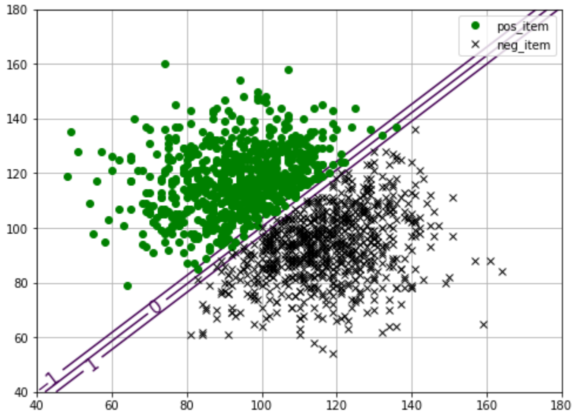
2. 直接将该数据集中的数据代入上一部分中实现的 SVM，也分别计算 C=1、0.1、0.01 的情况：

```
w_2_1, b_2_1 = solve_dual_1(train_x_2, train_y_2, 1)
w_2_2, b_2_2 = solve_dual_1(train_x_2, train_y_2, 0.1)
w_2_3, b_2_3 = solve_dual_1(train_x_2, train_y_2, 0.01)
```

3. 可视化分离超平面：







可见，随着 C 的减小，分离间隔逐渐增大，所以在 C 较大的情况下，有着更小的分离间隔，对于样本的判定更加宽松，但同时更有可能出现误分类的情况。

4. 计算准确率，仍然为 1：

```
get_accuracy(test_x_2, test_y_2, w_2_2, b_2_2)
```

```
1.0
```

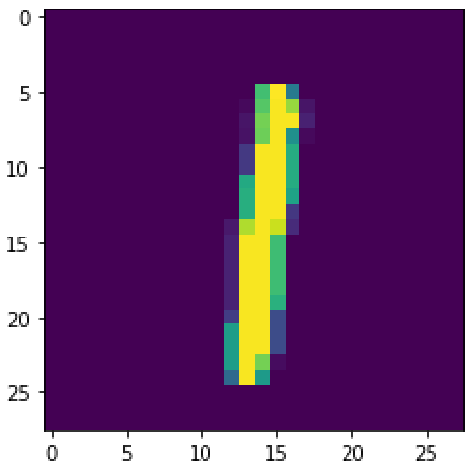
Handwritten Digit Recognition:

1. 首先实现实验指导书中提供的 strimage 方法：

```
def strimage(n):
    digit_dict = {}
    i = 0
    with open('./data5/train-01-images.svm', 'r') as f:
        for it in f.readlines():
            digit_dict[i] = it[3:]
            i += 1

    x = np.array([int(j[0]) for j in [i.split(':') for i in digit_dict[n].split()]])
    y = np.array([int(j[1]) for j in [i.split(':') for i in digit_dict[n].split()]])

    grid = np.zeros(784)
    grid[x] = y
    grid1 = grid.reshape(28, 28)
    grid1 = grid1 * 100 / 255

    # grid1 = grid1.reshape(28, 28)
    # grid1 = np.fliplr(np.diag(np.ones((28)))) * grid1
    # grid1 = np.rot90(grid1, 3)

    plt.imshow(grid1)
```

测试：

```python
strimage(10)
```



2. 再实现 matlab 中提供的 extractLBPFeature 方法：

```python
def extractLBPFeature(digit_data):
    data = np.zeros((digit_data.shape[0], 59))

    for i in range(digit_data.shape[0]):
        temp = local_binary_pattern(digit_data[i].reshape(28, 28), 8, 1, 'nri_uniform')
        data[i] = np.array(np.bincount(temp.astype(np.int64).flatten().tolist(), minlength=59))

    data = normalize(data)
    return data
```

此处使用 skimage 中的 local_binary_pattern 方法，再自行计算相应 LBP 值的数量，最后进行 normalize。

此处提取结果与 matlab 中有所不同，最终预测效果也要弱于在网络上找到的往届同学在 matlab 中实现的效果，推测是此处的原因。

3. 读取训练集中的数据：

```python
def get_digit_data():
    digit_train_dict, digit_test_dict = {}, {}
    num1, num2 = 0, 0
    digit_train_label, digit_test_label = [], []
    with open('./data5/train-01-images.svm', 'r') as f:
        for it in f.readlines():
            digit_train_dict[num1] = it[3:]
            digit_train_label.append(int(it[:2]))
            num1 += 1
    with open('./data5/test-01-images.svm', 'r') as f:
        for it in f.readlines():
            digit_test_dict[num2] = it[3:]
            digit_test_label.append(int(it[:2]))
            num2 += 1

    train_index = np.random.randint(0, len(digit_train_dict), 2000)

    train_data = np.zeros((len(digit_train_dict), 784))
    test_data = np.zeros((len(digit_test_dict), 784))
    digit_train_label = np.array(digit_train_label)
    digit_test_label = np.array(digit_test_label)

    for key, value in digit_train_dict.items():
        x = np.array([int(j[0]) for j in [i.split(':') for i in value.split()]])
        y = np.array([int(j[1]) for j in [i.split(':') for i in value.split()]])
        train_data[key][x] = y

    for key, value in digit_test_dict.items():
        x = np.array([int(j[0]) for j in [i.split(':') for i in value.split()]])
        y = np.array([int(j[1]) for j in [i.split(':') for i in value.split()]])
        test_data[key][x] = y

    return extractLBPFeature(train_data[train_index]), digit_train_label[train_index], train_index, extractLBPFeat
    # return train_data, digit_train_label, test_data, digit_test_label
```

注意到，最后对于数据集中的样本，调用了 extractLBPFeature 方法进行 LBP 特征提

取。

训练数据是随机选取训练集中的 2000 个数据，此前尝试过多次直接使用全部样本进行训练，在该情况下，每一次训练在此处的硬件环境下，需要花费 50 分钟左右，并且对于此部分要求首先使用的普通 SVM 无法求解。因此受网络上往届同学在 matlab 中的实现启发，采用该种方法作为训练数据。

4. 求解普通 SVM 的对偶问题：

```
def solve_dual_2(x, y):
    m, n = x.shape
    P = get_H(x, y)
    q = np.ones(m) * (-1)
    G, h = None, None
    A = y.astype('float')
    b = np.zeros(1)
    lb = np.zeros(m)
    ub = None

    alpha = solve_qp(P, q, G, h, A, b, lb, ub, solver='cvxopt')
    # print(alpha)

    w = get_w(alpha, x, y)
    b = get_b_2(alpha, w, x, y)

    return w, b
```

求解 b：

```
def get_b_2(alpha, w, x, y):
    m, n = x.shape
    index = []
    for i in range(m):
        if alpha[i] > 0:
            index.append(i)
    index = np.array(index)

    x = x[index]
    y = y[index]

    return (y - np.dot(x, w)).sum() / len(index)
```

得到在训练集及测试集上的准确率：

```
w_3, b_3 = solve_dual_2(digit_train_data, digit_train_label)
p1 = get_accuracy(digit_train_data, digit_train_label, w_3, b_3, show_false_pos=True)
p2 = get_accuracy(digit_test_data, digit_test_label, w_3, b_3, show_false_pos=True)

p1, p2
```

```
false_pos: [5, 7, 12, 16, 17, 19, 20, 21, 22, 23]
false_pos: [6, 7, 17, 20, 37, 51, 52, 57, 63, 68]
(0.7695, 0.7735224586288416)
```

5. 再次使用之前实现的软间隔 SVM 进行测试：

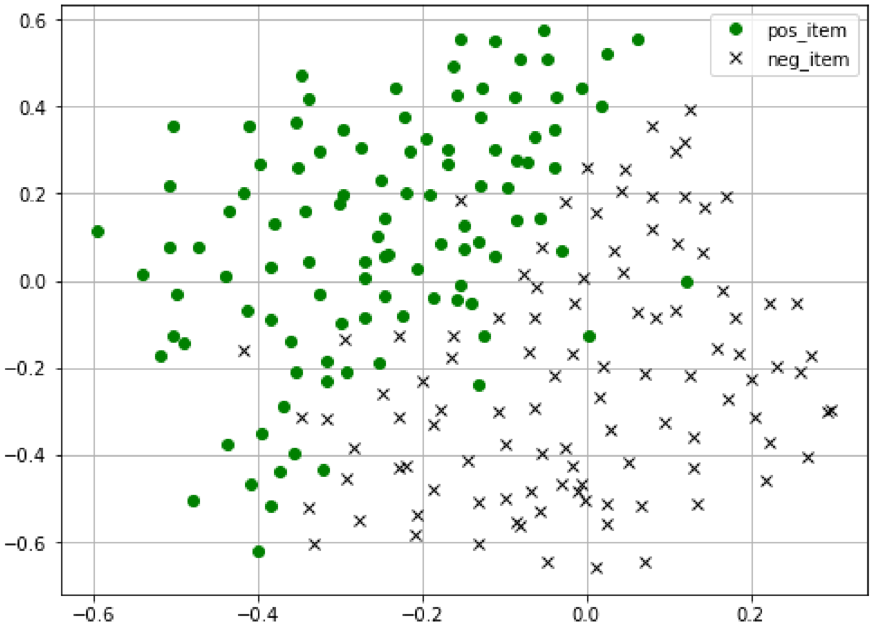
分别取 C=0.001、0.01、... 、 1e6：

```python
C = [0.001, 0.01, 0.1, 1, 10, 1e2, 1e3, 1e4, 1e5, 1e6]
for c in C:
    temp_w, temp_b = solve_dual_1(digit_train_data, digit_train_label, c)
    p1 = get_accuracy(digit_train_data, digit_train_label, temp_w, temp_b)
    p2 = get_accuracy(digit_test_data, digit_test_label, temp_w, temp_b)
    print('C = {}:'.format(c), p1, p2)
```

```
C = 0.001: 0.5465 0.5366430260047281
C = 0.01: 0.5465 0.5366430260047281
C = 0.1: 0.5465 0.5366430260047281
C = 1: 0.591 0.5787234042553191
C = 10: 0.6825 0.6728132387706856
C = 100.0: 0.7025 0.6912529550827423
C = 1000.0: 0.7215 0.7177304964539007
C = 10000.0: 0.74 0.7404255319148936
C = 100000.0: 0.768 0.7650118203309693
C = 1000000.0: 0.7695 0.7735224586288416
```

Non_Linear SVM：

1. 载入数据并可视化：



可见测出的 pos_item 及 neg_item 之间没有明显的线性分离平面，因此需要引入核方法在高维空间中进行分离。

2. 高斯核函数：

$$K(x^{(i)}, x^{(j)}) = \phi(x^{(i)})^T \phi(x^{(j)}) = exp(-\gamma||x^{(i)} - x^{(j)}||^2), \gamma > 0$$

注意到此处其一般形式中的 1/σ 使用 γ 代替：

```python
def kernel(x_i, x_j, gamma):
    x_i, x_j = x_i.reshape(-1), x_j.reshape(-1)
    return np.exp(-np.dot((x_i-x_j).T, (x_i-x_j)) * gamma)
```

3. 此最优化问题的对偶问题：

$$\begin{aligned} \min_{\alpha} \quad & \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}\alpha_i\alpha_j y_i y_j K(x_i,x_j)-\sum_{i=1}^{N}\alpha_i \\ s.t. \quad & \sum_{i=1}^{N}\alpha_i y_i = 0 \\ & 0 \le \alpha_i \le C, i=1,2,\dots,N \end{aligned}$$

4. 计算海森矩阵、b：

```
def get_H_2(x, y, gamma):
    m, n = x.shape
    P = np.zeros((m, m))
    for i in range(m):
        for j in range(m):
            P[i, j] = kernel(x[i], x[j], gamma) * y[i] * y[j]
    return P
```

```
def get_b_2(alpha, gamma, C, x, y):
    m, n = x.shape
    index = []
    for i in range(m):
        if alpha[i] > 0 and alpha[i] < C:
            index.append(i)
    index = np.array(index)

    x_j = x[index]
    y_j = y[index]

    temp = np.zeros(y_j.shape)
    for i in range(len(y)):
        for j in range(len(index)):
            temp[i] += alpha[i] * y[i] * kernel(x[i], x_j[j], gamma)

    # return (y - temp).sum()
    return (y_j - temp).sum() / len(index)
```

5. 求解对偶问题：

```
def solve_dual_kernel(x, y, gamma, C):
    m, n = x.shape
    P = get_H_2(x, y, gamma)
    q = np.ones(m) * (-1)
    G, h = None, None
    A = y.astype('float')
    b = np.zeros(1)
    lb = np.zeros(m)
    ub = np.ones(m) * C

    alpha = solve_qp(P, q, G, h, A, b, lb, ub, solver='cvxopt')
    # print(alpha)

    b = get_b_2(alpha, gamma, C, x, y)

    return alpha, b
```

6. 决策函数：

$$f(x) = sign(\sum_{i=1}^{N} \alpha_i^* y_i K(x, x_i) + b^*)$$

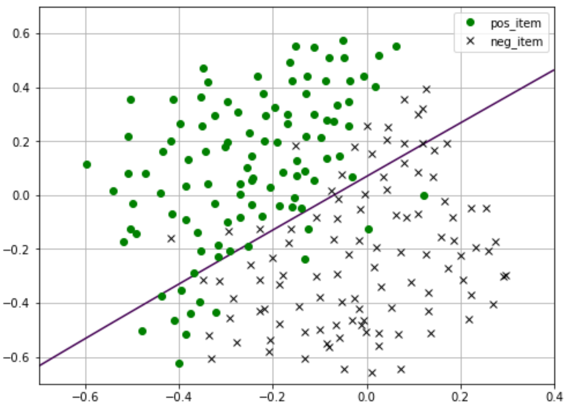
```
def pred_2(x_i, x, y, alpha, b, gamma):
    temp = []
    for i in range(x.shape[0]):
        temp.append(kernel(x_i, x[i], gamma))

    temp = np.array(temp)

    return (temp * alpha * y).sum() + b
```

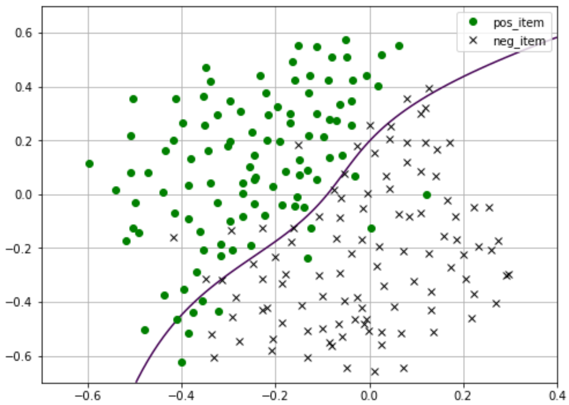
可视化分离平面：

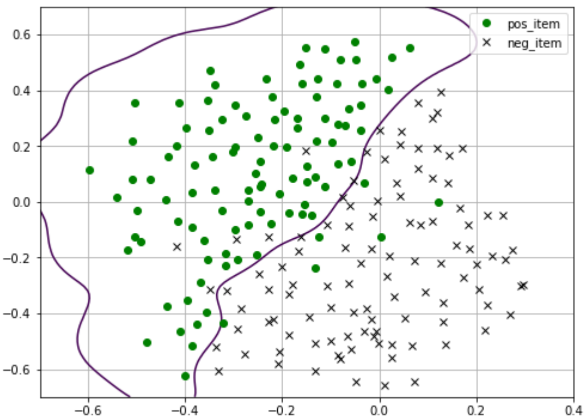
```
def showplot_3(alpha, b, gamma):
    plt.figure(figsize=(8, 6))
    x_1 = np.linspace(-0.7, 0.4, 100)
    x_2 = np.linspace(-0.7, 0.7, 100)
    p = np.zeros((len(x_1), len(x_2)))

    for i in range(len(x_1)):
        for j in range(len(x_2)):
            p[j][i] = pred_2(np.array((x_1[i], x_2[j])), train_x_3, train_y_3, alpha, b, gamma)

    plt.contour(x_1, x_2, p, [0.])

    plt.plot(pos_train_3[:, 0], pos_train_3[:, 1], 'og', label='pos_item')
    plt.plot(neg_train_3[:, 0], neg_train_3[:, 1], 'xk', label='neg_item')
    plt.grid()
    plt.legend()
    plt.show()
```

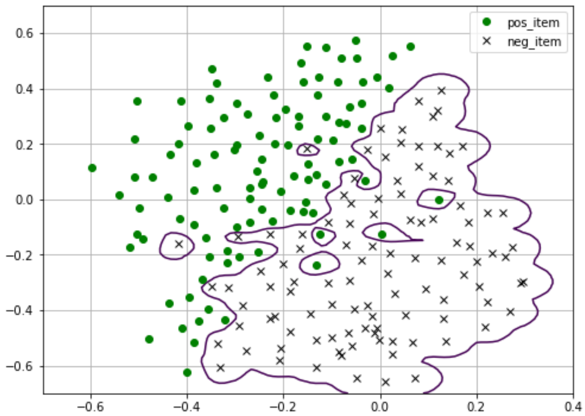
7. 设置 C=0.1，分别测试 γ =1，10，100，1000 各情况下的拟合程度：

```
C = 0.1
gamma_1, gamma_2, gamma_3, gamma_4 = 1, 10, 100, 1000
alpha_1, b_1 = solve_dual_kernel(train_x_3, train_y_3, gamma_1, C)
alpha_2, b_2 = solve_dual_kernel(train_x_3, train_y_3, gamma_2, C)
alpha_3, b_3 = solve_dual_kernel(train_x_3, train_y_3, gamma_3, C)
alpha_4, b_4 = solve_dual_kernel(train_x_3, train_y_3, gamma_4, C)
```









可以看到，随着γ的增加，拟合程度逐渐升高，而对于γ=1000的情况，过拟合严重，尽管在该情况下有着100%的准确率，在实际应用中也是不可取的。

Sequential Minimal Optimization:

1. 决策函数：

$$g(x)=\sum_{i=1}^{N}\alpha_i y_i K(x_i,x)+b$$

```python
def g(x_i, x, y, alpha, b, gamma):
    temp = []
    for i in range(x.shape[0]):
        temp.append(kernel(x_i, x[i], gamma))

    temp = np.array(temp)

    return (temp * alpha * y).sum() + b
```

2. 计算 E：

$$E_i=g(x_i)-y_i=(\sum_{j=1}^{N}\alpha_j y_j K(x_j,x_i)+b)-y_i,\quad i=1,2$$

```python
def cal_E(x_i, y_i, x, y, alpha, b, gamma):
    return g(x_i, x, y, alpha, b, gamma) - y_i
```

3. 判断是否符合 KKT 条件：

```python
def KKT(i, alpha, x, y, b, gamma, C):
    temp = y[i]*g(x[i], x, y, alpha, b, gamma)
    if (alpha[i] == 0) and (temp >= 1):
        return True, 0
    elif (alpha[i] > 0 and alpha[i] < C) and (temp == 1):
        return True, 0
    elif (alpha[i] == C) and (temp <= 1):
        return True, 0
    else:
        return False, abs(temp-1)
```

4. 停机条件，不满足该条件则停止迭代：

$$\sum_{i=1}^{N}\alpha_i y_i = 0, \quad 0 \le \alpha_i \le C, \quad i = 1, 2, \dots, N$$

$$y_i \cdot g(x_i) \begin{cases} \ge 1, & \{x_i | \alpha_i = 0\} \\ = 1, & \{x_i | 0 < \alpha_i < C\} \\ \le 1, & \{x_i | \alpha_i = C\} \end{cases}$$

```python
def judge(alpha, x, y, C, b, gamma):
    m = alpha.shape[0]
    if (alpha >= 0).sum() != m:
        return False
    if (alpha <= C).sum() != m:
        return False
    if (alpha*y).sum() != 0:
        return False
    for i in range(m):
        flag, temp = KKT(i, alpha, x, y, b, gamma, C)
        if not flag:
            return False

    return True
```

5. 更新 b：

$$\begin{aligned} b_1^{new} &= -E_1 - y_1 K_{11}(\alpha_1^{new} - \alpha_1^{old}) - y_2 K_{21}(\alpha_2^{new} - \alpha_2^{old}) + b^{old} \\ b_2^{new} &= -E_2 - y_1 K_{12}(\alpha_1^{new} - \alpha_1^{old}) - y_2 K_{22}(\alpha_2^{new} - \alpha_2^{old}) + b^{old} \\ b^{new} &= \frac{b_1 + b_2}{2} \end{aligned}$$

```python
def cal_b(gamma, E_1, E_2, x_1, x_2, y_1, y_2, alpha_1_new, alpha_1_old, alpha_2_new, alpha_2_old, b_old):
    b_1_new = -E_1-y_1*kernel(x_1, x_1, gamma)*(alpha_1_new-alpha_1_old)-y_2*kernel(x_2, x_1, gamma)*(alpha_2_new-
    b_2_new = -E_2-y_1*kernel(x_1, x_2, gamma)*(alpha_1_new-alpha_1_old)-y_2*kernel(x_2, x_2, gamma)*(alpha_2_new-

    return (b_1_new+b_2_new) / 2
```

6. 计算所有样本的 E，便于找到与 α_1 对应的 E_1 差别最大的 E_2，从而得到最快的训练速度：

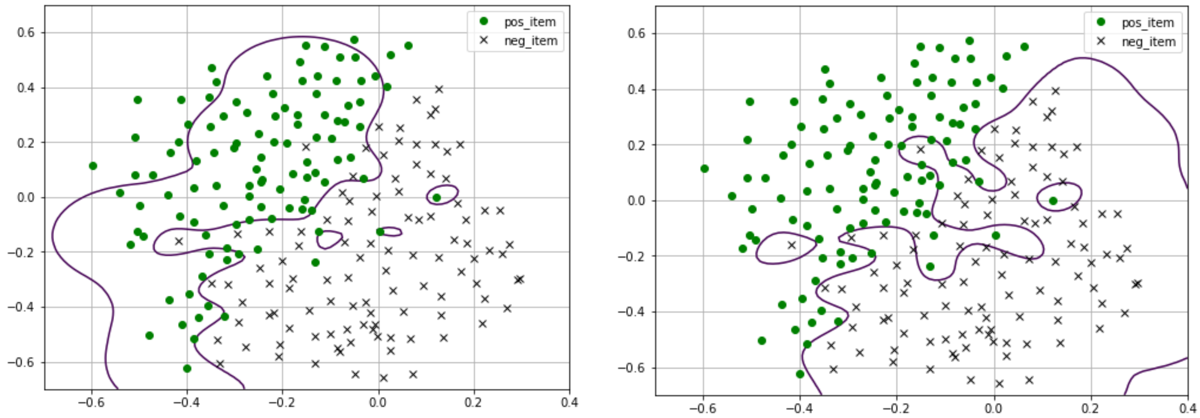
```python
def get_E(alpha, x, y, b, C, gamma):
    m = alpha.shape[0]
    E_new = []
    for i in range(m):
        E_new.append(cal_E(x[i], y[i], x, y, alpha, b, gamma))

    return E_new
```

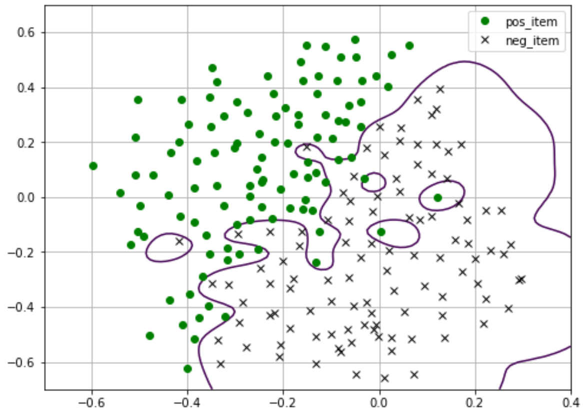
7. SMO 算法：

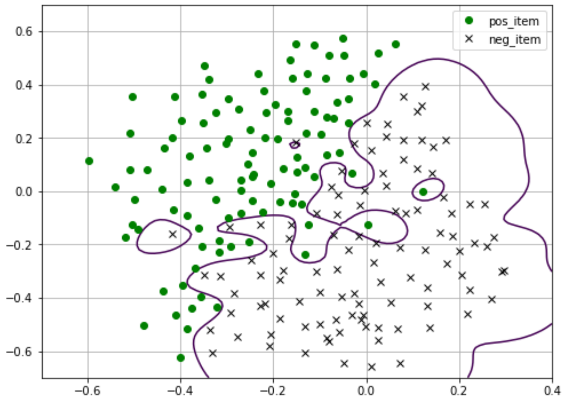
```python
def SMO(x, y, gamma, C, iter=400):
    m, n = x.shape
    alpha_hat = np.zeros(m)
    b = 1
    index_1, index_2 = 1, 0

    while (iter > 0):
        E = get_E(alpha_hat, x, y, b, C, gamma)

        max_ = -1
        for i in range(m):
            flag, temp = KKT(i, alpha_hat, x, y, b, gamma, C)
            if not flag and max_ < temp:
                index_1 = i
                max_ = temp

        if E[index_1] > 0:
            index_2 = E.index(min(E))
        else:
            index_2 = E.index(max(E))

        alpha_1, alpha_2 = alpha_hat[index_1], alpha_hat[index_2]
        x_1, x_2 = x[index_1], x[index_2]
        y_1, y_2 = y[index_1], y[index_2]
        E_1, E_2 = E[index_1], E[index_2]

        if y_1 == y_2:
            L, H = max(0, alpha_2+alpha_1-C), min(C, alpha_2+alpha_1)
        else:
            L, H = max(0, alpha_2-alpha_1), min(C, C+alpha_2-alpha_1)

        alpha_2_new_unc = alpha_2 + (y_2*(E_1-E_2)/(kernel(x_1, x_1, gamma)+kernel(x_2, x_2, gamma)-2*kernel(x_1,

        if alpha_2_new_unc > H:
            alpha_2_new = H
        elif alpha_2_new_unc < L:
            alpha_2_new = L
        else:
            alpha_2_new = alpha_2_new_unc

        alpha_2_new = alpha_2_new_unc

        alpha_1_new = alpha_1 + y_1*y_2*(alpha_2-alpha_2_new)

        alpha_hat[index_1], alpha_hat[index_2] = alpha_1_new, alpha_2_new

        b = cal_b(gamma, E_1, E_2, x_1, x_2, y_1, y_2, alpha_1_new, alpha_1, alpha_2_new, alpha_2, b)

        if judge(alpha_hat, x, y, C, b, gamma):
            break

        iter -= 1

    return alpha_hat, b
```

该算法首先选取最不满足 KKT 条件的α_1，再寻找|E_1-E_2|有最大值的α_2，求得 alpha_2_new 后，通过约束条件求得 alpha_1_new，再更新 b，重复迭代直到求出最优解。

8. 令 C=0.01，γ为 100，分别查看迭代 100、200、300、400 次的预测边界：







上面四种情况的准确率分别为 0.8767、0.9668、1.0、1.0。
可见此处实现的 SMO 算法成功完成了预测任务。

结论分析与体会：

1. 在实验前，需要充分理解使用 matlab、python 等工具，才能更好地进行实验，实现实验中的各个步骤。
2. 在实验中，需要理解掌握 SVM 的实现原理，掌握其深层含义，结合实验指导书，才能更好地完成实验；
3. 通过使用 qpsolvers 等第三方库，可以便捷地对凸优化等问题进行求解，而对于 extractLBPFeature 等 matlab 中独有的方法，通过 python 难以达到相同效果，需要实验者自行实现，有一定的挑战性。

附录：程序源代码

```
# %%
import numpy as np
import matplotlib.pyplot as plt
from qpsolvers import solve_qp
from skimage.feature import local_binary_pattern
from sklearn.preprocessing import normalize

# %% [markdown]
# Dataset 1

# %%
training_1 = np.loadtxt('./data5/training_1.txt', dtype=np.double)  # 载入数据
test_1 = np.loadtxt('./data5/test_1.txt', dtype=np.double)

# %%
train_x_1 = training_1[:, :-1]
train_y_1 = training_1[:, -1]
```

```python
test_x_1 = test_1[:, :-1]
test_y_1 = test_1[:, -1]

# %%
pos_train_1 = training_1[training_1[:, 2] == 1]
neg_train_1 = training_1[training_1[:, 2] == -1]

# %%
plt.figure(figsize=(8, 6))
plt.plot(pos_train_1[:, 0], pos_train_1[:, 1], 'og', label='pos_item')
plt.plot(neg_train_1[:, 0], neg_train_1[:, 1], 'xk', label='neg_item')
plt.grid()
plt.legend()
plt.show()

# %% [markdown]
# Regularized SVM:
# $$
# \begin{align*}
# \mathop{min}\limits_{w,b,\xi}\quad & \frac{1}{2}||w||^2 +
C\sum_{i=1}^m\xi_i \\
# s.t.\quad & y^{(i)}(w^Tx^{(i)}+b)\ge 1-\xi_i,\forall i=1, ...,m \\
# & \xi_i \ge 0, \forall i=1, ..., m
# \end{align*}
# $$
# 对偶问题:
# $$
# \begin{align*}
# \mathop{max}\limits_{\alpha}\quad & \sum_{i=1}^m\alpha_i-
\frac{1}{2}\sum_{i,j=1}^my^{(i)}y^{(j)}\alpha_i\alpha_j <x^{(i)},x^{(j)}>
\\
# s.t.\quad & 0\le\alpha_i\le C,\forall i=1, ...,m \\
# & \sum_{i=1}^m\alpha_iy^{(i)}=0
# \end{align*}
# $$
# 问题转变为:
# $$
# \mathop{min}\limits_{\alpha}\quad
\frac{1}{2}\sum_{i,j=1}^my^{(i)}y^{(j)}\alpha_i\alpha_j
<x^{(i)},x^{(j)}>-\sum_{i=1}^m\alpha_i
# $$
# 由此求解:
# $$
# \begin{align*}
```

```
# &w^*=\sum^N_{i=1}\alpha_i^*y_ix_i \\
# &b^*=y_j-\sum^N_{i=1}y_i\alpha_i^*<x_i, x_j>
# \end{align*}
# $$
# 分离超平面:
# $$
# w^*·x+b^*=0
# $$

# %%
def get_H(x, y):
    m, n = x.shape
    P = np.zeros((m, m))
    for i in range(m):
        for j in range(m):
            P[i, j] = np.dot(x[i], x[j]) * y[i] * y[j]
    return P

# %%
def get_w(alpha, x, y):
    w = np.zeros(x.shape)
    w[:, 0] = x[:, 0] * alpha * y
    w[:, 1] = x[:, 1] * alpha * y
    return np.sum(w, axis=0)

# %%
def get_b(alpha, w, x, y, C):
    m, n = x.shape
    index = []
    for i in range(m):
        if alpha[i] > 0 and alpha[i] < C:
            index.append(i)
    index = np.array(index)

    x = x[index]
    y = y[index]

    return (y - np.dot(x, w)).sum() / len(index)

# %%
def solve_dual_1(x, y, C):
    m, n = x.shape
    P = get_H(x, y)
    q = np.ones(m) * (-1)
```

```python
    G, h = None, None
    A = y.astype('float')
    b = np.zeros(1)
    lb = np.zeros(m)
    ub = np.ones(m) * C

    alpha = solve_qp(P, q, G, h, A, b, lb, ub, solver='cvxopt')
    # print(alpha)

    w = get_w(alpha, x, y)
    b = get_b(alpha, w, x, y, C)

    return w, b

# %%
def pred(w, b, x):
    return np.dot(x, w) + b

# %%
def display_contour(w, b):
    x_1 = np.linspace(40, 180, 400)
    x_2 = np.linspace(40, 180, 400)
    p = np.zeros((len(x_1), len(x_2)))

    for i in range(len(x_1)):
        for j in range(len(x_2)):
            p[i][j] = pred(w, b, np.array((x_1[i], x_2[j])))

    contour_zero = plt.contour(x_1, x_2, p, [0.])
    contour_pos = plt.contour(x_1, x_2, p, [1.])
    contour_neg = plt.contour(x_1, x_2, p, [-1.])
    plt.clabel(contour_zero, inline=1, fontsize=15)
    plt.clabel(contour_pos, inline=1, fontsize=15)
    plt.clabel(contour_neg, inline=1, fontsize=15)

# %%
w_1_1, b_1_1 = solve_dual_1(train_x_1, train_y_1, 1)
w_1_2, b_1_2 = solve_dual_1(train_x_1, train_y_1, 0.1)
w_1_3, b_1_3 = solve_dual_1(train_x_1, train_y_1, 0.01)

# %%
w_1_1, b_1_1

# %%
```

```python
plt.figure(figsize=(8, 6))
display_contour(w_1_1, b_1_1)
plt.plot(pos_train_1[:, 0], pos_train_1[:, 1], 'og', label='pos_item')
plt.plot(neg_train_1[:, 0], neg_train_1[:, 1], 'xk', label='neg_item')
plt.grid()
plt.legend()
plt.show()

# %%
plt.figure(figsize=(8, 6))
display_contour(w_1_2, b_1_2)
plt.plot(pos_train_1[:, 0], pos_train_1[:, 1], 'og', label='pos_item')
plt.plot(neg_train_1[:, 0], neg_train_1[:, 1], 'xk', label='neg_item')
plt.grid()
plt.legend()
plt.show()

# %%
plt.figure(figsize=(8, 6))
display_contour(w_1_3, b_1_3)
plt.plot(pos_train_1[:, 0], pos_train_1[:, 1], 'og', label='pos_item')
plt.plot(neg_train_1[:, 0], neg_train_1[:, 1], 'xk', label='neg_item')
plt.grid()
plt.legend()
plt.show()

# %%
def get_accuracy(x, y, w, b, show_false_pos=False):
    m, n = x.shape

    p = pred(w, b, x)
    false_pos = []

    num = 0
    for i in range(m):
        if p[i] > 0 and y[i] == 1:
            num += 1
        elif p[i] < 0 and y[i] == -1:
            num += 1
        else:
            false_pos.append(i)

    if show_false_pos:
        print('false_pos:', false_pos[:10])
```

```python
    return num / m

# %%
get_accuracy(test_x_1, test_y_1, w_1_1, b_1_1)

# %% [markdown]
# Dataset 2

# %%
training_2 = np.loadtxt('./data5/training_2.txt', dtype=np.int32)  # 载入数据
test_2 = np.loadtxt('./data5/test_2.txt', dtype=np.int32)

# %%
train_x_2 = training_2[:, :-1]
train_y_2 = training_2[:, -1]
test_x_2 = test_2[:, :-1]
test_y_2 = test_2[:, -1]

# %%
pos_train_2 = training_2[training_2[:, 2] == 1]
neg_train_2 = training_2[training_2[:, 2] == -1]

# %%
plt.figure(figsize=(8, 6))
plt.plot(pos_train_2[:, 0], pos_train_2[:, 1], 'og', label='pos_item')
plt.plot(neg_train_2[:, 0], neg_train_2[:, 1], 'xk', label='neg_item')
plt.grid()
plt.legend()
plt.show()

# %%
w_2_1, b_2_1 = solve_dual_1(train_x_2, train_y_2, 1)
w_2_2, b_2_2 = solve_dual_1(train_x_2, train_y_2, 0.1)
w_2_3, b_2_3 = solve_dual_1(train_x_2, train_y_2, 0.01)

# %%
w_2_1, b_2_1

# %%
plt.figure(figsize=(8, 6))
display_contour(w_2_1, b_2_1)
plt.plot(pos_train_2[:, 0], pos_train_2[:, 1], 'og', label='pos_item')
```

```python
plt.plot(neg_train_2[:, 0], neg_train_2[:, 1], 'xk', label='neg_item')
plt.grid()
plt.legend()
plt.show()

# %%
plt.figure(figsize=(8, 6))
display_contour(w_2_2, b_2_2)
plt.plot(pos_train_2[:, 0], pos_train_2[:, 1], 'og', label='pos_item')
plt.plot(neg_train_2[:, 0], neg_train_2[:, 1], 'xk', label='neg_item')
plt.grid()
plt.legend()
plt.show()

# %%
plt.figure(figsize=(8, 6))
display_contour(w_2_3, b_2_3)
plt.plot(pos_train_2[:, 0], pos_train_2[:, 1], 'og', label='pos_item')
plt.plot(neg_train_2[:, 0], neg_train_2[:, 1], 'xk', label='neg_item')
plt.grid()
plt.legend()
plt.show()

# %%
get_accuracy(test_x_2, test_y_2, w_2_2, b_2_2)

# %% [markdown]
# Handwritten Digit Recognition

# %%
def strimage(n):
    digit_dict = {}
    i = 0
    with open('./data5/train-01-images.svm', 'r') as f:
        for it in f.readlines():
            digit_dict[i] = it[3:]
            i += 1

    x = np.array([int(j[0]) for j in [i.split(':') for i in
digit_dict[n].split()]])
    y = np.array([int(j[1]) for j in [i.split(':') for i in
digit_dict[n].split()]])

    grid = np.zeros(784)
```

```python
    grid[x] = y
    grid1 = grid.reshape(28, 28)
    grid1 = grid1 * 100 / 255

    # grid1 = grid1.reshape(28, 28)
    # grid1 = np.fliplr(np.diag(np.ones((28)))) * grid1
    # grid1 = np.rot90(grid1, 3)

    plt.imshow(grid1)

# %%
strimage(10)

# %%
def extractLBPFeature(digit_data):
    data = np.zeros((digit_data.shape[0], 59))

    for i in range(digit_data.shape[0]):
        temp = local_binary_pattern(digit_data[i].reshape(28, 28), 8, 1,
'nri_uniform')
        data[i] =
np.array(np.bincount(temp.astype(np.int64).flatten().tolist(),
minlength=59))

    data = normalize(data)
    return data

# %%
def get_digit_data():
    digit_train_dict, digit_test_dict = {}, {}
    num1, num2 = 0, 0
    digit_train_label, digit_test_label = [], []
    with open('./data5/train-01-images.svm', 'r') as f:
        for it in f.readlines():
            digit_train_dict[num1] = it[3:]
            digit_train_label.append(int(it[:2]))
            num1 += 1
    with open('./data5/test-01-images.svm', 'r') as f:
        for it in f.readlines():
            digit_test_dict[num2] = it[3:]
            digit_test_label.append(int(it[:2]))
            num2 += 1

    train_index = np.random.randint(0, len(digit_train_dict), 2000)
```

```
    train_data = np.zeros((len(digit_train_dict), 784))
    test_data = np.zeros((len(digit_test_dict), 784))
    digit_train_label = np.array(digit_train_label)
    digit_test_label = np.array(digit_test_label)

    for key, value in digit_train_dict.items():
        x = np.array([int(j[0]) for j in [i.split(':') for i in
value.split()]])
        y = np.array([int(j[1]) for j in [i.split(':') for i in
value.split()]])
        train_data[key][x] = y

    for key, value in digit_test_dict.items():
        x = np.array([int(j[0]) for j in [i.split(':') for i in
value.split()]])
        y = np.array([int(j[1]) for j in [i.split(':') for i in
value.split()]])
        test_data[key][x] = y

    return extractLBPFeature(train_data[train_index]),
digit_train_label[train_index], train_index,
extractLBPFeature(test_data), digit_test_label
    # return train_data, digit_train_label, test_data, digit_test_label

# %%
digit_train_data, digit_train_label, train_index, digit_test_data,
digit_test_label = get_digit_data()

# %%
digit_train_data.shape

# %%
digit_train_data[0]

# %%
def get_b_2(alpha, w, x, y):
    m, n = x.shape
    index = []
    for i in range(m):
        if alpha[i] > 0:
            index.append(i)
    index = np.array(index)
```

```python
    x = x[index]
    y = y[index]

    return (y - np.dot(x, w)).sum() / len(index)

# %%
def solve_dual_2(x, y):
    m, n = x.shape
    P = get_H(x, y)
    q = np.ones(m) * (-1)
    G, h = None, None
    A = y.astype('float')
    b = np.zeros(1)
    lb = np.zeros(m)
    ub = None

    alpha = solve_qp(P, q, G, h, A, b, lb, ub, solver='cvxopt')
    # print(alpha)

    w = get_w(alpha, x, y)
    b = get_b_2(alpha, w, x, y)

    return w, b

# %%
w_3, b_3 = solve_dual_2(digit_train_data, digit_train_label)
p1 = get_accuracy(digit_train_data, digit_train_label, w_3, b_3,
show_false_pos=True)
p2 = get_accuracy(digit_test_data, digit_test_label, w_3, b_3,
show_false_pos=True)

p1, p2

# %%
strimage(23)

# %%
C = [0.001, 0.01, 0.1, 1, 10, 1e2, 1e3, 1e4, 1e5, 1e6]
for c in C:
    temp_w, temp_b = solve_dual_1(digit_train_data, digit_train_label, c)
    p1 = get_accuracy(digit_train_data, digit_train_label, temp_w,
temp_b)
    p2 = get_accuracy(digit_test_data, digit_test_label, temp_w, temp_b)
    print('C = {}:'.format(c), p1, p2)
```

```
# %% [markdown]
# ### Non-Linear SVM

# %%
training_3 = np.loadtxt('./data5/training_3.text', dtype=np.double)  # 载入数据

# %%
train_x_3 = training_3[:, :-1]
train_y_3 = training_3[:, -1]

# %%
pos_train_3 = training_3[training_3[:, -1] == 1]
neg_train_3 = training_3[training_3[:, -1] == -1]

# %%
plt.figure(figsize=(8, 6))
plt.plot(pos_train_3[:, 0], pos_train_3[:, 1], 'og', label='pos_item')
plt.plot(neg_train_3[:, 0], neg_train_3[:, 1], 'xk', label='neg_item')
plt.grid()
plt.legend()
plt.show()

# %% [markdown]
# Radial Basis Function kernel:
# $$
# K(x^{(i)}, x^{(j)})=\phi(x^{(i)})^T\phi(x^{(j)})=exp(-\gamma||x^{(i)}-x^{(j)}||^2),\gamma>0
# $$

# %%
def kernel(x_i, x_j, gamma):
    x_i, x_j = x_i.reshape(-1), x_j.reshape(-1)
    return np.exp(-np.dot((x_i-x_j).T, (x_i-x_j)) * gamma)

# %% [markdown]
# 求解最优化问题:
# $$
# \begin{align*}
# \mathop{min}\limits_{\alpha}\quad & \frac{1}{2}\sum^{N}_{i=1}\sum^{N}_{j=1}\alpha_i\alpha_jy_iy_jK(x_i, x_j)-\sum^{N}_{i=1}\alpha_i \\
# s.t.\quad & \sum^N_{i=1}\alpha_iy_i=0 \\
```

```python
# & 0\le\alpha_i\le C, i=1,2,\dots,N
# \end{align*}
# $$

# %%
def get_H_2(x, y, gamma):
    m, n = x.shape
    P = np.zeros((m, m))
    for i in range(m):
        for j in range(m):
            P[i, j] = kernel(x[i], x[j], gamma) * y[i] * y[j]
    return P

# %% [markdown]
# $$
# b^*=y_j-\sum^{N}_{i=1}a^*_iy_iK(x_i,x_j)
# $$

# %%
def get_b_2(alpha, gamma, C, x, y):
    m, n = x.shape
    index = []
    for i in range(m):
        if alpha[i] > 0 and alpha[i] < C:
            index.append(i)
    index = np.array(index)

    x_j = x[index]
    y_j = y[index]

    temp = np.zeros(y_j.shape)
    for i in range(len(y)):
        for j in range(len(index)):
            temp[i] += alpha[i] * y[i] * kernel(x[i], x_j[j], gamma)

    # return (y - temp).sum()
    return (y_j - temp).sum() / len(index)

# %%
def solve_dual_kernel(x, y, gamma, C):
    m, n = x.shape
    P = get_H_2(x, y, gamma)
    q = np.ones(m) * (-1)
    G, h = None, None
```

```python
    A = y.astype('float')
    b = np.zeros(1)
    lb = np.zeros(m)
    ub = np.ones(m) * C

    alpha = solve_qp(P, q, G, h, A, b, lb, ub, solver='cvxopt')
    # print(alpha)

    b = get_b_2(alpha, gamma, C, x, y)

    return alpha, b

# %%
C = 0.1
gamma_1, gamma_2, gamma_3, gamma_4 = 1, 10, 100, 1000
alpha_1, b_1 = solve_dual_kernel(train_x_3, train_y_3, gamma_1, C)
alpha_2, b_2 = solve_dual_kernel(train_x_3, train_y_3, gamma_2, C)
alpha_3, b_3 = solve_dual_kernel(train_x_3, train_y_3, gamma_3, C)
alpha_4, b_4 = solve_dual_kernel(train_x_3, train_y_3, gamma_4, C)

# %% [markdown]
# 决策函数:
# $$
# f(x)=sign(\sum^N_{i=1}\alpha^*_iy_iK(x, x_i)+b^*)
# $$

# %%
def pred_2(x_i, x, y, alpha, b, gamma):
    temp = []
    for i in range(x.shape[0]):
        temp.append(kernel(x_i, x[i], gamma))

    temp = np.array(temp)

    return (temp * alpha * y).sum() + b

# %%
def showplot_3(alpha, b, gamma):
    plt.figure(figsize=(8, 6))
    x_1 = np.linspace(-0.7, 0.4, 100)
    x_2 = np.linspace(-0.7, 0.7, 100)
    p = np.zeros((len(x_1), len(x_2)))

    for i in range(len(x_1)):
```

```
        for j in range(len(x_2)):
            p[j][i] = pred_2(np.array((x_1[i], x_2[j])), train_x_3,
train_y_3, alpha, b, gamma)

    plt.contour(x_1, x_2, p, [0.])

    plt.plot(pos_train_3[:, 0], pos_train_3[:, 1], 'og',
label='pos_item')
    plt.plot(neg_train_3[:, 0], neg_train_3[:, 1], 'xk',
label='neg_item')
    plt.grid()
    plt.legend()
    plt.show()

# %%
showplot_3(alpha_1, b_1, gamma_1)

# %%
showplot_3(alpha_2, b_2, gamma_2)

# %%
showplot_3(alpha_3, b_3, gamma_3)

# %%
showplot_3(alpha_4, b_4, gamma_4)

# %% [markdown]
# ### Sequential Minimal Optimization

# %% [markdown]
# $$
# g(x)=\sum^N_{i=1}\alpha_iy_iK(x_i, x)+b
# $$

# %%
def g(x_i, x, y, alpha, b, gamma):
    temp = []
    for i in range(x.shape[0]):
        temp.append(kernel(x_i, x[i], gamma))

    temp = np.array(temp)

    return (temp * alpha * y).sum() + b
```

```python
# %% [markdown]
# $$
# E_i=g(x_i)-y_i=(\sum^N_{j=1}\alpha_jy_jK(x_j,x_i)+b)-y_i,\quad i=1,2
# $$

# %%
def cal_E(x_i, y_i, x, y, alpha, b, gamma):
    return g(x_i, x, y, alpha, b, gamma) - y_i

# %%
def KKT(i, alpha, x, y, b, gamma, C):
    temp = y[i]*g(x[i], x, y, alpha, b, gamma)
    if (alpha[i] == 0) and (temp >= 1):
        return True, 0
    elif (alpha[i] > 0 and alpha[i] < C) and (temp == 1):
        return True, 0
    elif (alpha[i] == C) and (temp <= 1):
        return True, 0
    else:
        return False, abs(temp-1)

# %% [markdown]
# 停机条件:
# $$
# \sum^N_{i=1}\alpha_iy_i=0, \quad 0\le\alpha_i\le C, \quad i=1, 2,
\dots, N
# $$
#
# $$
# y_i · g(x_i)
# \begin{cases}
# & \ge 1, \quad \{x_i|\alpha_i=0\} \\
# & = 1, \quad \{x_i|0<\alpha_i<C\} \\
# & \le 1, \quad \{x_i|\alpha_i=C\}
# \end{cases}
# $$

# %%
def judge(alpha, x, y, C, b, gamma):
    m = alpha.shape[0]
    if (alpha >= 0).sum() != m:
        return False
    if (alpha <= C).sum() != m:
        return False
```

```python
    if (alpha*y).sum() != 0:
        return False
    for i in range(m):
        flag, temp = KKT(i, alpha, x, y, b, gamma, C)
        if not flag:
            return False

    return True

# %% [markdown]
# $$
# \begin{align*}
# &b_1^{new}=-E_1-y_1K_{11}(\alpha_1^{new}-\alpha_1^{old})-y_2K_{21}(\alpha_2^{new}-\alpha_2^{old})+b^{old} \\
# &b_2^{new}=-E_2-y_1K_{12}(\alpha_1^{new}-\alpha_1^{old})-y_2K_{22}(\alpha_2^{new}-\alpha_2^{old})+b^{old} \\
# &b^{new}=\frac{b_1+b_2}{2}
# \end{align*}
# $$

# %%
def cal_b(gamma, E_1, E_2, x_1, x_2, y_1, y_2, alpha_1_new, alpha_1_old, alpha_2_new, alpha_2_old, b_old):
    b_1_new = -E_1-y_1*kernel(x_1, x_1, gamma)*(alpha_1_new-alpha_1_old)-y_2*kernel(x_2, x_1, gamma)*(alpha_2_new-alpha_2_old)+b_old
    b_2_new = -E_2-y_1*kernel(x_1, x_2, gamma)*(alpha_1_new-alpha_1_old)-y_2*kernel(x_2, x_2, gamma)*(alpha_2_new-alpha_2_old)+b_old

    return (b_1_new+b_2_new) / 2

# %% [markdown]
# $$
# E_i^{new}=\sum_{S}y_j\alpha_jK(x_i,x_j)+b^{new}-y_i
# $$

# %%
def get_E(alpha, x, y, b, C, gamma):
    m = alpha.shape[0]
    E_new = []
    for i in range(m):
        E_new.append(cal_E(x[i], y[i], x, y, alpha, b, gamma))

    return E_new
```

```python
# %%
def SMO(x, y, gamma, C, iter=400):
    m, n = x.shape
    alpha_hat = np.zeros(m)
    b = 1
    index_1, index_2 = 1, 0

    while (iter > 0):
        E = get_E(alpha_hat, x, y, b, C, gamma)

        max_ = -1
        for i in range(m):
            flag, temp = KKT(i, alpha_hat, x, y, b, gamma, C)
            if not flag and max_ < temp:
                index_1 = i
                max_ = temp

        if E[index_1] > 0:
            index_2 = E.index(min(E))
        else:
            index_2 = E.index(max(E))

        alpha_1, alpha_2 = alpha_hat[index_1], alpha_hat[index_2]
        x_1, x_2 = x[index_1], x[index_2]
        y_1, y_2 = y[index_1], y[index_2]
        E_1, E_2 = E[index_1], E[index_2]

        if y_1 == y_2:
            L, H = max(0, alpha_2+alpha_1-C), min(C, alpha_2+alpha_1)
        else:
            L, H = max(0, alpha_2-alpha_1), min(C, C+alpha_2-alpha_1)

        alpha_2_new_unc = alpha_2 + (y_2*(E_1-E_2)/(kernel(x_1, x_1,
gamma)+kernel(x_2, x_2, gamma)-2*kernel(x_1, x_2, gamma)))

        if alpha_2_new_unc > H:
            alpha_2_new = H
        elif alpha_2_new_unc < L:
            alpha_2_new = L
        else:
            alpha_2_new = alpha_2_new_unc

        alpha_2_new = alpha_2_new_unc
```

```python
        alpha_1_new = alpha_1 + y_1*y_2*(alpha_2-alpha_2_new)

        alpha_hat[index_1], alpha_hat[index_2] = alpha_1_new, alpha_2_new

        b = cal_b(gamma, E_1, E_2, x_1, x_2, y_1, y_2, alpha_1_new,
alpha_1, alpha_2_new, alpha_2, b)

        if judge(alpha_hat, x, y, C, b, gamma):
            break

        iter -= 1

    return alpha_hat, b

# %%
def get_accuracy_2(train_x, train_y, alpha, b, gamma):
    m = train_x.shape[0]
    cnt = 0
    for i in range(m):
        temp = pred_2(train_x[i], train_x, train_y, alpha, b, gamma)
        if temp >= 0 and train_y[i] == 1:
            cnt += 1
        elif temp < 0 and train_y[i] == -1:
            cnt += 1

    return cnt / m

# %%
gamma, C = 100, 0.1

# %%
alpha_4_1, b_4_1 = SMO(train_x_3, train_y_3, gamma, C, iter=100)
print(get_accuracy_2(train_x_3, train_y_3, alpha_4_1, b_4_1, gamma))
showplot_3(alpha_4_1, b_4_1, gamma)

# %%
alpha_4_2, b_4_2 = SMO(train_x_3, train_y_3, gamma, C, iter=200)
print(get_accuracy_2(train_x_3, train_y_3, alpha_4_2, b_4_2, gamma))
showplot_3(alpha_4_2, b_4_2, gamma)

# %%
alpha_4_3, b_4_3 = SMO(train_x_3, train_y_3, gamma, C, iter=300)
print(get_accuracy_2(train_x_3, train_y_3, alpha_4_3, b_4_3, gamma))
showplot_3(alpha_4_3, b_4_3, gamma)
```

```python
# %%
alpha_4_4, b_4_4 = SMO(train_x_3, train_y_3, gamma, C, iter=400)
print(get_accuracy_2(train_x_3, train_y_3, alpha_4_4, b_4_4, gamma))
showplot_3(alpha_4_4, b_4_4, gamma)

# %%
```