山东大学 计算机科学与技术 学院

机器学习 课程实验报告

学号：201618130133	姓名：代荣森	班级：2016.4
实验题目：SVM		
实验学时：4	实验日期：2018.11.30	

实验目的：

学习使用支持向量机进行线性和非线性分类

硬件环境：

i5-6200U 8G RAM HD Graphics520

软件环境：

Visual Studio Code + Python

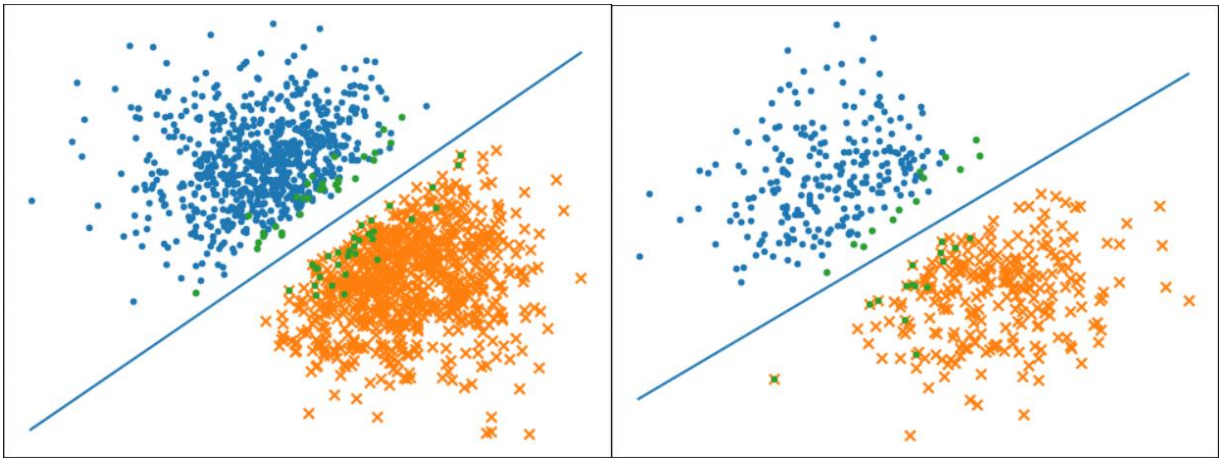
实验步骤与内容：

本次三个实验均使用 SMO 代替标准 QP 求解器。

实验一

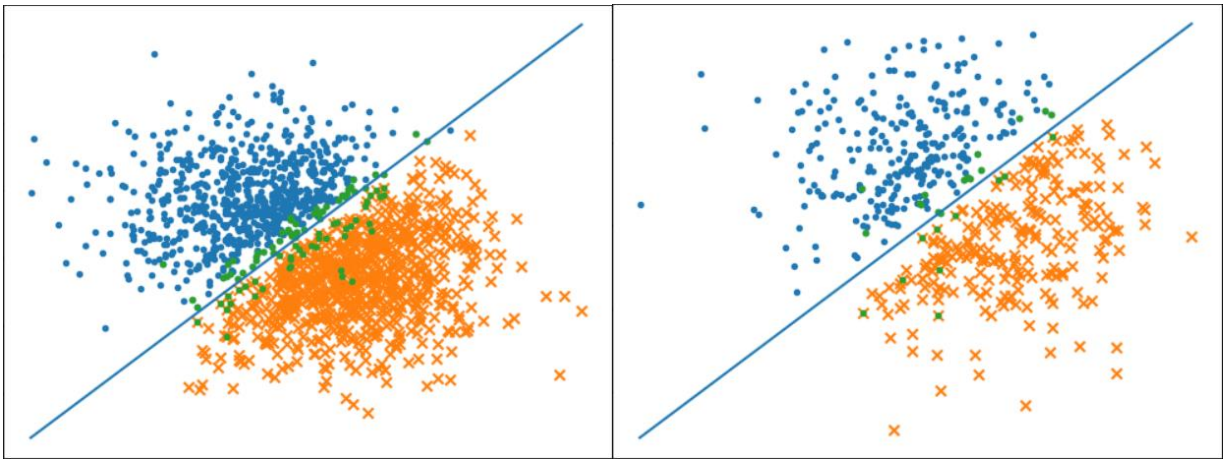
1、用训练数据绘制 SVM 的决策边界。

训练集 1　　测试集 1



准确率：1.0

训练集 2



准确率：0.996

绿色的点为支持向量

出错在第 109 和 278 行

2、尝试正则化项 C 的二元数值，并报告您的观察结果。

改变 C 的值对结果影响很小。

实验二、手写数字识别

由于原数据集过大，故随机抽取 200 个数据进行训练。

用训练集直接测试得到正确率为

```
训练值数量：200
准确率：1.0
```

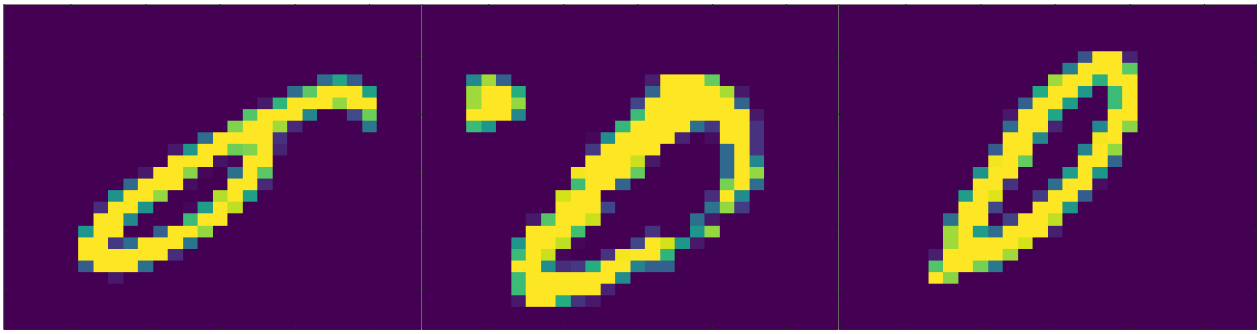
用测试集进行测试得到正确率为

```
训练值数量：200
准确率：0.9985815602836879
```

用测试集训练测试得到正确率为

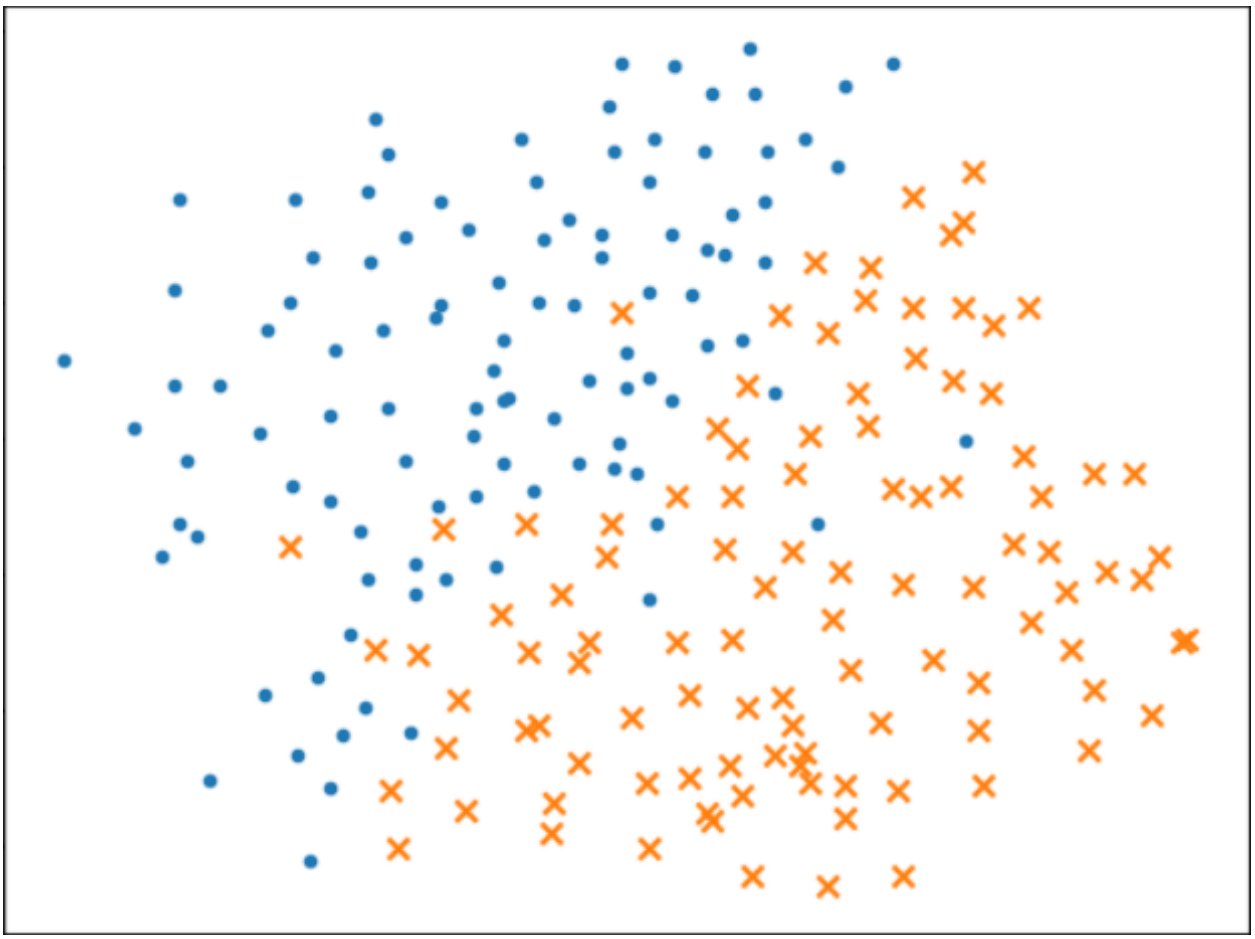
```
训练值数量：2115
准确率：1.0
```

得到 3 个错误图像



实验三、非线性 SVM

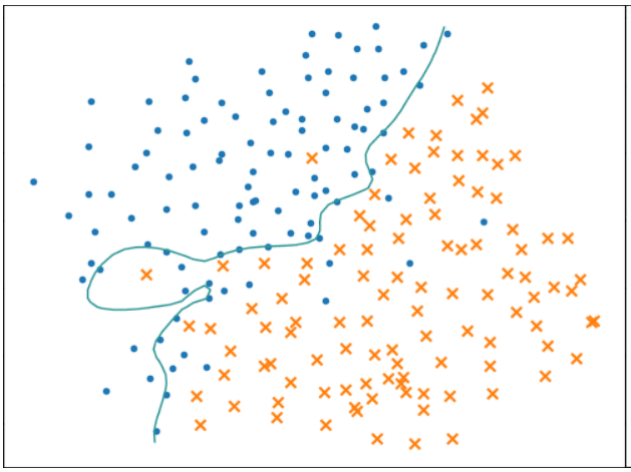
绘制散点图，发现其线性不可分



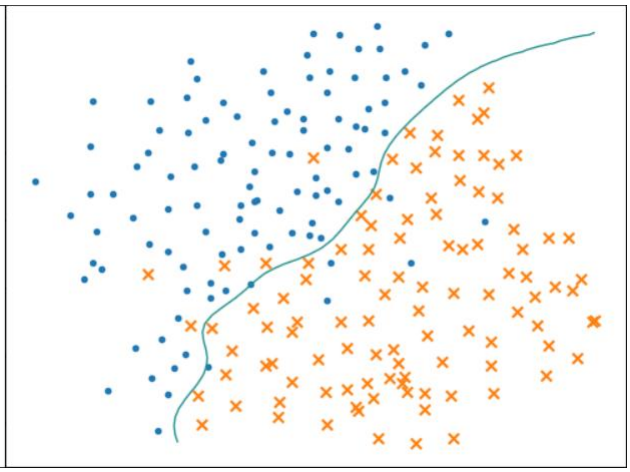
使用加入高斯内核后的 SMO

```
for xi in range(length):
            for xj in range(length):
                zij[xi,xj] = np.exp(-gama * np.sum((x[xi,:] - x[xj,:])**2))
```

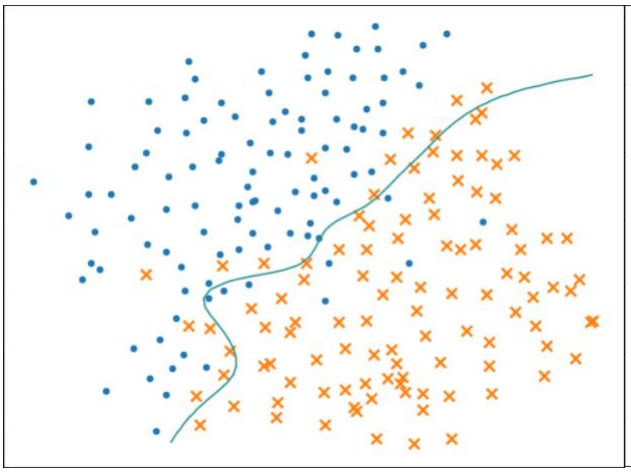
当 C = 10， γ = 100 时：



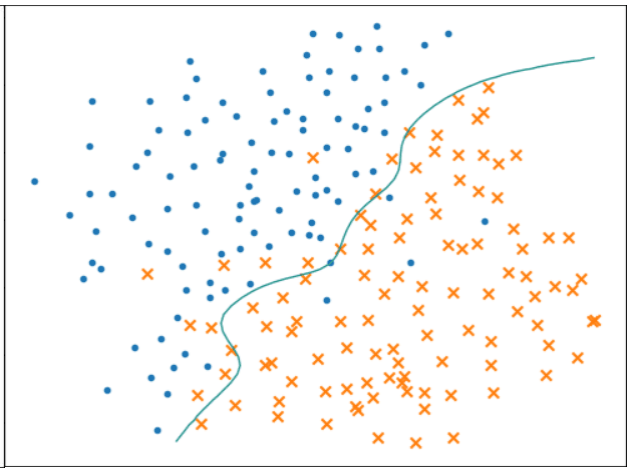
当 C = 10， γ = 1000 时：



当 C = 1， γ = 100 时：



当 C = 1， γ = 1000 时：



结论分析与体会：

最近在学习 python，所以使用 python 来写了这个实验，由于一些规则用法不一样，所以有些地方需要特别注意

之前用 libSVM 库直接可以得出训练测试结果，但由于完全交给库来做，能学到的只有对训练和测试集进行字符串处理，最后又决定重新用 SMO 算法来完成实验

实验难度主要在对于 SMO 和核函数的理解上，以及对于算法算出的 α 和支持向量如何画图上。写完之后对于 SMO 还是有很多地方不是很清楚，还需多多体会。

附录：程序源代码

SMO.py

```
import numpy as np
def SMO(x, y, C):
    # x = np.matrix(x)
    length = len(y)
    alpha = np.zeros(length)
    b = 0
```

```
    tol = 1e-7
    while 1:
        num_changed_alphas = 0
        xt = np.transpose(x)
        xt = np.matrix(xt)
        #计算内积
        zij = x * xt
        for i in range(length):
            Ei = alpha * y * zij[:,i] + b - y[i]
            #不满足 kkt 条件
            if (y[i] * Ei < -tol and alpha[i] < C) or (y[i] * Ei > tol and
alpha[i] > 0):
                for j in range(0,i):
                    Ej = alpha * y * zij[:,j] + b - y[j]
                    alpha_old_i = alpha[i]
                    alpha_old_j = alpha[j]
                    if y[i] == y[j]:
                        L = np.maximum(0, alpha[i] + alpha[j] - C)
                        H = np.minimum(C, alpha[j] + alpha[i])
                    else:
                        L = np.maximum(0, alpha[j] - alpha[i])
                        H = np.minimum(C, C + alpha[j] - alpha[i])
                    if L == H:
                        continue
                    yita = zij[i,j] * 2 - zij[i,i] - zij[j,j]
                    if yita >= 0:
                        continue
                    alpha[j] = alpha[j] - y[j] * (Ei - Ej) / yita
                    if alpha[j] > H:
                        alpha[j] = H
                    elif alpha[j] < L:
                        alpha[j] = L
                    if np.linalg.norm(alpha[j] - alpha_old_j) < tol:
                        continue
                    alpha[i] = alpha[i] + y[i] * y[j] * (alpha_old_j - alpha[j])
                    b1 = b - Ei - y[i] * (alpha[i] - alpha_old_i) * zij[i,i] - y[j]
* (alpha[j]-alpha_old_j) * zij[i,j]
                    b2 = b - Ej - y[i] * (alpha[i] - alpha_old_i) * zij[i,j] - y[j]
* (alpha[j]-alpha_old_j) * zij[j,j]
                    if alpha[i] < C and alpha[i] > 0:
                        b = b1
                    elif alpha[j] < C and alpha[j] > 0:
                        b = b2
                    else:
                        b = (b1 + b2) / 2
                    #重新计算 Ei,如果还不满足 kkt 条件,继续,如果满足则寻找下一个 i
```

```
                    Ei = alpha * y * zij[:,i] + b - y[i]
                    if (y[i] * Ei < -tol and alpha[i] < C) or (y[i] * Ei > tol and
alpha[i] > 0):
                        num_changed_alphas = num_changed_alphas + 1
                    else:
                        break
        #去除 alpha==0 和 alpha==c
        x1 = []
        y1 = []
        alpha1 = []
        for k in range(len(alpha)):
            if alpha[k] != 0 and alpha[k] != C:
                # la[k] = 1
                x1.append(x[k])
                y1.append(y[k])
                alpha1.append(alpha[k])

        x = x1
        y = np.array(y1)
        alpha = np.array(alpha1)
        length = len(y)
        if num_changed_alphas == 0:
            break
    return alpha,x,y,b  #x:支持向量，y:label
```

Gaussiansmo.py

```
import numpy as np
def Gaussiansmo(x, y, C, gama):
    # x = np.matrix(x)
    length = len(y)
    alpha = np.zeros(length)
    b = 0
    tol = 1e-7
    while 1:
        num_changed_alphas = 0
        zij = np.zeros((length,length))
        zij = np.matrix(zij)
        for xi in range(length):
            for xj in range(length):
                zij[xi,xj] = np.exp(-gama * np.sum((x[xi,:] - x[xj,:])**2))

        for i in range(length):
            Ei = alpha * y * zij[:,i] + b - y[i]
            #不满足 kkt 条件
            if (y[i] * Ei <- tol and alpha[i] < C) or (y[i] * Ei > tol and
alpha[i] > 0):
                for j in range(0,length,2):
```

```
                    Ej = alpha * y * zij[:,j] + b - y[j]
                    alpha_old_i = alpha[i]
                    alpha_old_j = alpha[j]
                    if y[i] == y[j]:
                        L = np.maximum(0, alpha[i] + alpha[j] - C)
                        H = np.minimum(C, alpha[j] + alpha[i])
                    else:
                        L = np.maximum(0, alpha[j] - alpha[i])
                        H = np.minimum(C, C + alpha[j] - alpha[i])
                    if L == H:
                        continue
                    yita = zij[i,j] * 2 - zij[i,i] - zij[j,j]
                    if yita >= 0:
                        continue
                    alpha[j] = alpha[j] - y[j] * (Ei - Ej) / yita
                    if alpha[j] > H:
                        alpha[j] = H
                    elif alpha[j] < L:
                        alpha[j] = L
                    if np.linalg.norm(alpha[j] - alpha_old_j) < tol:
                        continue
                    alpha[i] = alpha[i] + y[i] * y[j] * (alpha_old_j - alpha[j])
                    b1 = b - Ei - y[i] * (alpha[i] - alpha_old_i) * zij[i,i] - y[j]
* (alpha[j]-alpha_old_j) * zij[i,j]
                    b2 = b - Ej - y[i] * (alpha[i] - alpha_old_i) * zij[i,j] - y[j]
* (alpha[j]-alpha_old_j) * zij[j,j]
                    if alpha[i] < C and alpha[i] > 0:
                        b = b1
                    elif alpha[j] < C and alpha[j] > 0:
                        b = b2
                    else:
                        b = (b1 + b2) / 2
                    #重新计算 Ei,如果还不满足 kkt 条件,继续,如果满足则寻找下一个 i
                    Ei = alpha * y * zij[:,i] + b - y[i]
                    # if (y[i] * Ei <- tol and alpha[i] < C) or (y[i] * Ei > tol
and alpha[i] > 0):
                    #   num_changed_alphas = num_changed_alphas + 1
                    # else:
                    #   break
                    num_changed_alphas = num_changed_alphas + 1
            #去除 alpha==0 和 alpha==c
            # la = alpha
            x1 = []
            y1 = []
            alpha1 = []
            for k in range(len(alpha)):
```

```
            if alpha[k] != 0 and alpha[k] != C:
                # la[k] = 1
                x1.append(x[k])
                y1.append(y[k])
                alpha1.append(alpha[k])

        x = np.array(x1)
        y = np.array(y1)
        alpha = np.array(alpha1)
        length = len(y)
        if num_changed_alphas == 0:
            break
    return alpha,x,y,b  #x:支持向量，y:label
```