信息基础 II 2023 秋

Homework 4

学生：刘浩 *(202120120312)* 时间：*10* 月 *1* 日

1.1 实验要求

有 2 类二维空间点，A 类和 B 类。A 类点以 (0, 0) 为中心、(1, 0; 0, 1) 为协方差矩阵的二维高斯分布；B 类点以 (0, 0) 为中心、(5, 0; 0, 5) 为协方差矩阵的二维高斯分布；随机生成 50 个 A 类点，50 个 B 类点，并用 SVM 的方法进行分类, 请解释你的 SVM 的方法，提供程序和描述结果 (不能调用机器学习库，可以调用优化求解器, 选择合适的 kernel)

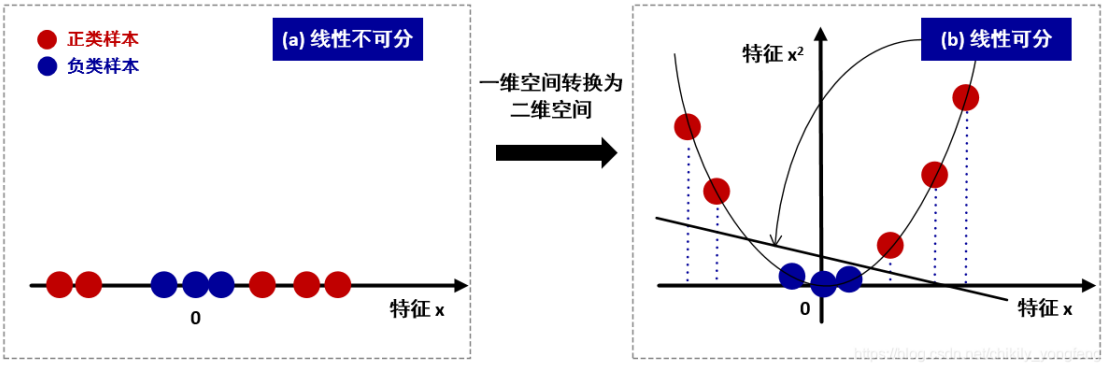
1.2 实验原理

1.2.1 kernel

首先是 kernel，其定义可以写作：

$$K(x,z) = \varphi(x)^T \varphi(z)$$

那么对于 $x^T z$ 就可以映射为 $\varphi(x)$ 和 $\varphi(z)$ 从而使得输入数据升维，在低维空间下线性不可分的数据，在 kernel 作用下在更高维得以线性可分，如下图所示：



常用的 kernel 有：

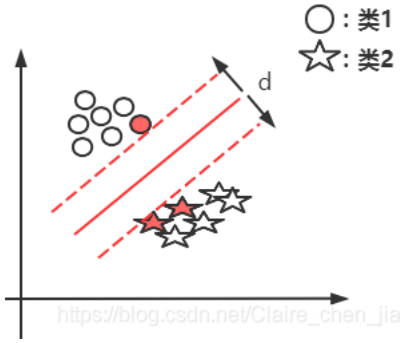
Gaussion Kernel：$K(x_i, x_j) = exp(-\frac{||x_i - x_j||^2}{2\sigma^2})$

Polynomial Kernel：$K(x_i, x_j) = (x_i x_j + 1)^2$

在本次作业中我均尝试了这两种 kernel 最终选择的最优的高斯 kernel，在实验结果中我将他们的效果进行了对比

1.2.2 SVM

我们在选取分类边界的时候，存在无数的边界线，那么哪种边界线是最佳的呢？我们采取的是最大化 margin(也就是图中的 d) 的方式找到最佳的分界线, 所谓支持向量就是将平行线向左右移动，相交的样本点就称为支持向量。如图中的红点。



那么可以得到以下优化问题：

$$\begin{aligned} \max \quad & \gamma \\ \text{s.t.} \quad & y^{(i)}\theta^T x^{(i)} \geqslant \gamma \quad \forall i \\ & \|\theta\|_2 = 1 \end{aligned} \tag{1.1}$$

而 $\|\theta\|_2 = 1$ 是一个非凸约束不可解，引入一个新变量 $w = \frac{\theta}{\gamma}$, 从而问题转化为：

$$\begin{aligned} \min \quad & (\|w\|_2)^2 \\ \text{s.t.} \quad & y^{(i)} w x^{(i)} \geqslant 1 \quad \forall i \end{aligned} \tag{1.2}$$

进而这是一个典型的 Quadratic Programming Problem，从而问题得以解决，为了考虑到线性不可分的情况，我们需要让 margin 尽可能大，同时出错尽可能少，从而引入松弛变量 e，那么问题转化为：

$$\begin{aligned} \min \quad & (\|w\|_2)^2 + C(\textstyle\sum_{i=1}^{n} e_i) \\ \text{s.t.} \quad & y^{(i)} w x^{(i)} \geqslant 1 - e_i \quad \forall i \\ & e_i \geqslant 0 \end{aligned} \tag{1.3}$$

为了将 SVM 问题 kernel 化，我们采取拉格朗日对偶问题的处理方法，将问题转化为：

$$\begin{aligned} \min_{\alpha} \quad & \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}\alpha_i\alpha_j y_i y_j (x_i \cdot x_j) - \sum_{i=1}^{n}\alpha_i \\ \text{s.t.} \quad & \begin{cases} \sum_{i=1}^{n}\alpha_i y_i = 0 \\ 0 \le \alpha_i \le C \end{cases} \end{aligned} \tag{1.4}$$

在本实验中我是采用了 QP 求解器手动实现了 SVM 的分类方法，其中标准的 QP 问题是这样的：

$$\begin{aligned} \min \quad & \frac{1}{2}x^T P x + q^T x \\ \text{s.t.} \quad & \begin{cases} Gx \le h \\ Ax = b \end{cases} \end{aligned} \tag{1.5}$$

那么在本实验中，照着这个标准的表达式去代入即可：P 在本实验中是 Gram 矩阵，即：

$$P = y_i y_j K(i,j)$$

q 是长度为 100 的全为-1 的向量，即：

$$q = \begin{bmatrix} -1 \\ -1 \\ \vdots \\ -1 \end{bmatrix} \longrightarrow 100 \times 1 \tag{1.6}$$

G 由一个全为负的单位矩阵和一个全为正的单位矩阵拼接构成从而同时满足 $0 \le \alpha_i \le C$，即：

$$G = \begin{bmatrix} -I_n \\ I_n \end{bmatrix}, h = \begin{bmatrix} 0 \\ C * \mathbf{1}_n \end{bmatrix} \tag{1.7}$$

A 由 y 值构成，维度 1×100 即：

$$A = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_{100} \end{bmatrix} \longrightarrow 1 \times 100 \tag{1.8}$$

b 自然是 0，至此 QP 问题所有系数全部已知，代入求解器即可求得 α，进而可以求得 b：

$$b = y_j - \sum_{i=1}^{100}\alpha_i y_i k\left(x_i, x_j\right) \tag{1.9}$$

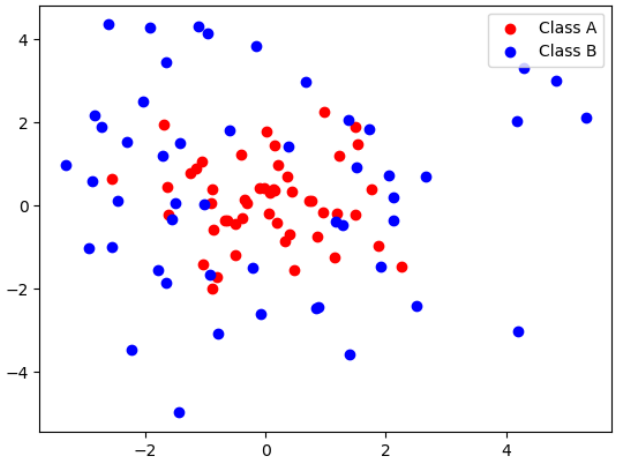
最终可以通过下式进行预测：

$$f(x) = \text{sign}\left(\sum_{i=1}^{100} \alpha_i y_i k\left(x, x_i\right) + b\right) \tag{1.10}$$

若 $f(x) > 0$ 则预测为 1，反之为 -1

1.3 实验步骤

和 Hw3 数据点是一模一样，因此对于数据的导入采取 hw3 的代码，只不过将 A 的标签值设置为-1，B 的标签值设置为 1 此处不再赘述，原始点的分布如下：



然后我完成 SVM 类，首先是高斯核和多项式核，将我们实验原理中核函数的表达式, 代入转化为代码形式即可：

```
def Gauss_kernel(self, x, y):
    # 高斯核函数
    sigma = 1.0
    return np.exp(-np.linalg.norm(x - y) ** 2 / (2 * sigma ** 2))

def Polynomial_kernel(self, x, y):
    # 多项式核函数
    return (np.dot(x, y) + 1) ** self.degree
```

然后导入 QP 库，将我们所需要的参数 (实验原理中已经详细说明每个参数代表的含义如公式 (1.6)-(1.9) 所示将其转化为代码形式即可) 导入到求解器中，完成 α 的求解，从而再结合求解 b 的公式完成 bias 的求解：

```
# 计算Gram矩阵
```

```
K = np.zeros((n_samples, n_samples))
for i in range(n_samples):
    for j in range(n_samples):
        if self.kernel == 'linear':
            K[i, j] = Y[i] * Y[j] * np.dot(X[i], X[j])
        elif self.kernel == 'gaussian':
            K[i, j] = Y[i] * Y[j] * self.Gauss_kernel(X[i], X[j])
        elif self.kernel == 'polynomial':
            K[i, j] = Y[i] * Y[j] * self.Polynomial_kernel(X[i], X[j])

# 定义凸二次规划问题
P = cvxopt.matrix(K)
q = cvxopt.matrix(-np.ones(n_samples))
G = cvxopt.matrix(np.vstack((-np.eye(n_samples), np.eye(n_samples))))
h = cvxopt.matrix(np.hstack((np.zeros(n_samples), self.C * np.ones(n_samples))))
A = cvxopt.matrix(Y, (1, n_samples))
b = cvxopt.matrix(0.0)

# 求解凸二次规划问题
solution = cvxopt.solvers.qp(P, q, G, h, A, b)
alphas = np.ravel(solution['x'])
support_vectors = alphas > self.tolerance

# 计算偏置项
bias = 0
for i in range(n_samples):
    if support_vectors[i]:
        bias += Y[i]
        for j in range(n_samples):
            if support_vectors[j]:
                if self.kernel == 'linear':
                    bias -= alphas[j] * Y[j] * np.dot(X[i], X[j])
                elif self.kernel == 'gaussian':
                    bias -= alphas[j] * Y[j] * self.Gauss_kernel(X[i], X[j])
                elif self.kernel == 'polynomial':
                    bias -= alphas[j] * Y[j] * self.Polynomial_kernel(X[i], X[j])
        break

self.alphas = alphas
self.support_vectors = support_vectors
self.bias = bias
```

然后我们只需要完成 f(x) 的，如公式 (1.10) 所示代入进行预测即可：

```
def predict(self, data):
    if self.kernel == 'linear':
        kernel_values = np.array([np.dot(data, X[i]) for i in range(len(X))])
    elif self.kernel == 'gaussian':
        kernel_values = np.array([self.Gauss_kernel(data, X[i]) for i in range(len(X))])
    elif self.kernel == 'polynomial':
        kernel_values = np.array([self.Polynomial_kernel(data, X[i]) for i in range(len(X))])
```

```
    return np.sign(np.dot(kernel_values.T, self.alphas * Y) + self.bias)
```

最后采取 Hw3 的方法我们计算预测的准确度和画图，最终得到的结果如下：

```
-------------Result-----------------
Line Accuracy: 57.0
Poly Accuracy: 83.0
RBF Accuracy: 93.0
```

后面我在网上还了解到了 SMO 算法进行 α 的求解，我利用 SMO 算法进行验证，采取同样的参数进行验证我的 QP 算法解出的准确率正确与否：

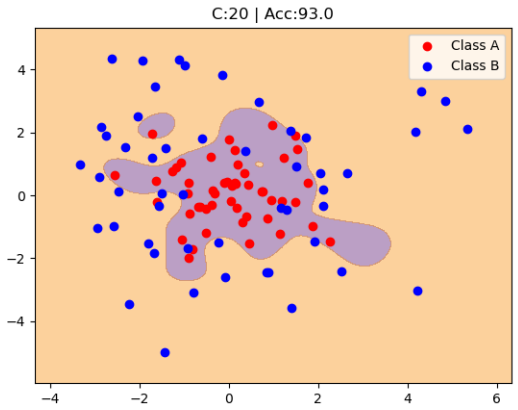


图 1.1: QP 解法的高斯 kernel

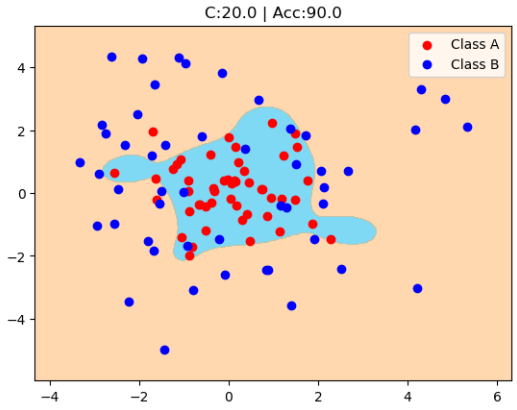


图 1.2: SMO 解法的高斯 kernel

经过验证使用 QP 算出来的结果应该是没问题的! 再次验证了实验原理的正确性!

最后我还探究了 C 的值对实验结果的影响我对比了 C=1 和 C=20：

可以发现 C 的值越大的话，边界的形状是越奇怪的，越容易受到某些异常点的影响，虽然在训练集上的准确率有所增加，但是实际上模型的效果是变差的，而较合适的 C 是边界范围较广的!

1.4 实验结果

程序结果一览：

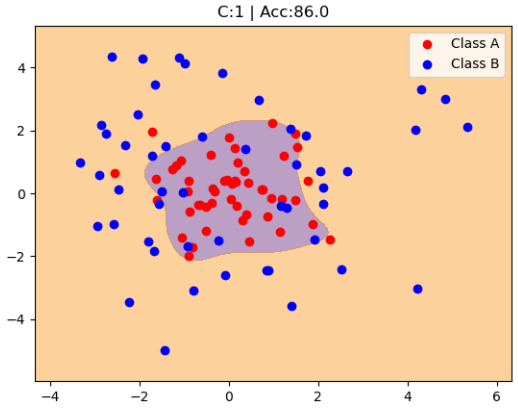


图 1.3: C=1 的高斯 kernel

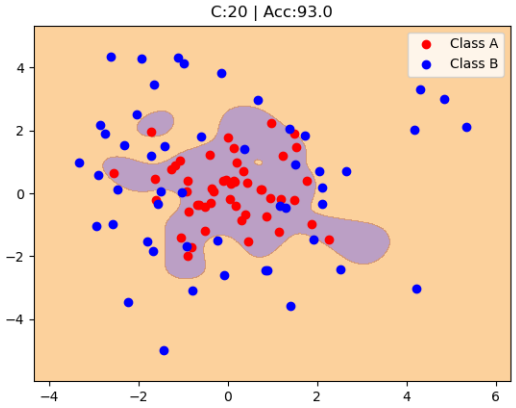
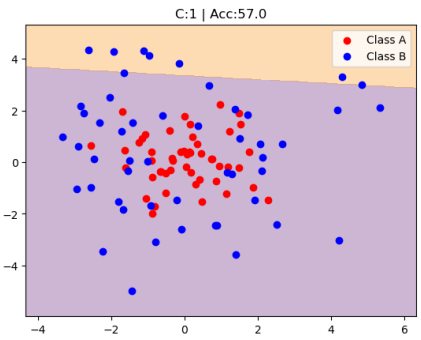
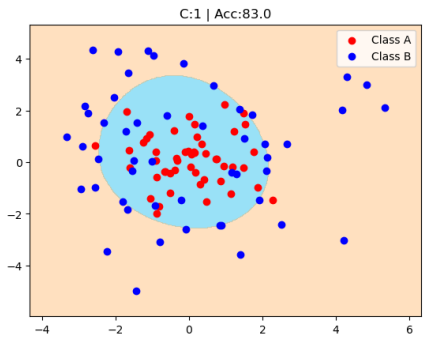


图 1.4: C=20 的高斯 kernel

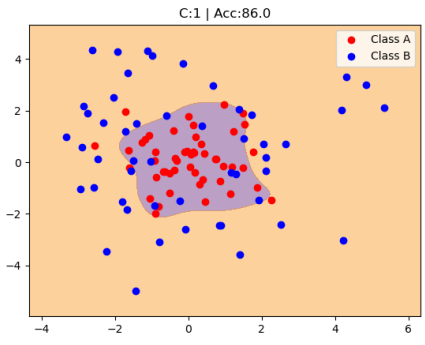
	准确率
线性 kernel	57%
多项式 kernel	83%
高斯 kernel	86%



(a) 线性 kernel



(b) 多项式 kenel



(c) 高斯 kernal

1.5 实验感想

本次实验和 Hw3 是一组数据，同样是进行分类，在做 Hw3 的时候我就用 sklearn 的库中的 SVM 方法进行了验证准确率是否能再次提高，而本次实验恰好让我们手动实现 SVM，本来上次由于没有学到 SVM 仅仅是用来验证，对于 SVM 原理还是迷迷糊糊的，这次听完课之后对 SVM 的原理进一步理解，然后我接着在 b 站上听了不少关于原理的讲解，终于大概搞懂了 SVM 的原理和如何去写代码，由于本次实验可以调用优化求解器，我进行了两种尝试，分别是上课讲的 PPT 中提到的 QP 库，然后还有网上查到的 SMO 优化算法进行求解 α；

本次实验和上次实验都是分类问题，但是使用的方法不一样，效果也不一样！在实验过程中我发现，对于我手动实现的这个 SVM 解法用时相当的长，我将我的代码计算时长和 sklearn 库进行了对比，得到的结果如下表所示：

可以看到 sklearn 的计算速度比我的代码快了二十多倍，代表我的代码能优化的地方还是很多的！

	计算时长 (单位：秒)
MyCode	219.0131516456604
sklearn	9.697283267974854

1.6 参考资料

[1] http://t.csdnimg.cn/39MUM

[2] http://t.csdnimg.cn/5H1J7

[3] http://t.csdnimg.cn/sSvha

[4] https://www.bilibili.com/video/BV1jt4y1E7BQ?t=869.1&p=10