深圳大学研究生课程：模式识别理论与方法

课程作业实验报告

实验名称：感知器算法

实验编号：Proj05-01

签　　名：

姓　　名：夏荣杰

学　　号：2170269107

截止提交日期：2018 年 5 月 25 日

摘要： 感知器算法是通过模拟人类神经元，学习两类已标记的样本，建立一个线性分类器。感知器学习的目标是求得一个能够将线性可分数据集的正负两类实例点完全正确分开的分离超平面，通过定义准则函数并将准则函数极小化来确定感知器模型参数，而极小化问题常用梯度下降法来解决。本实验的目的是学习和掌握两种感知器算法：批处理感知器算法和批处理裕量松弛算法。本实验给出了基于梯度下降法的这两种感知器算法，介绍了原理并编程实现，实验一通过设计基于梯度下降法的批处理感知器算法对两类数据进行训练以学习感知机模型的参数，并通过改变学习率进行重复实验。实验结果表明，随着迭代次数的增加，准则函数不断收敛，收敛速度取决于权值向量的初始值和学习率。实验二通过设计基于梯度下降法的批处理裕量松弛算法对两类数据进行训练来学习感知机模型的参数。实验过程中，采用不同的裕量间隔和学习率进行实验。实验结果表明，裕量间隔的大小决定了准则函数的初始值的大小，而学习率的大小决定了收敛速度的快慢。最后，对这两种感知器算法的共同点和不同点进行比较和分析。

一、 背景技术 或 基本原理

1. 两类线性可分的情况

假设有 n 个样本 $y_1, y_2, ..., y_n$，分为两类 ω_1, ω_2。

利用这些样本来确定下面判别函数的权向量 **a**：

$$g(x) = \sum_{i=0}^{d} a_i y_i(x) = \mathbf{a}^t \mathbf{y} \qquad (1)$$

若存在一个矢量 $\mathbf{a}^t$，在代入 g(**x**)后可以将所有不同类的样本分开，我们称该问题是线性可分的。

若 y_i 属于 ω_1 类时有 $\mathbf{a}^t \mathbf{y}_i > 0$，或 y_i 属于 ω_2 类时 $\mathbf{a}^t \mathbf{y}_i < 0$，则称 y_i 被正确分类了。

若对所有属于 ω_2 类的 y_i 乘以一个负号，则只需要找出使所有样本的 $\mathbf{a}^t \mathbf{y}_i > 0$ 的矢量 $\mathbf{a}^t$。该矢量 $\mathbf{a}^t$ 就是解矢量。

所以，关于线性分类决策面的求解转换为求解满足线性不等式方程 $\mathbf{a}^t \mathbf{y}_i > 0$ 的解，该问题的解可以使用梯度下降技术求解。

2. 梯度下降法

梯度下降法的基本思想及步骤：

① 定义一个准则函数（能量函数、误差函数）J(**a**)；

② 令 k = 1；

③ 选取任意的初始解矢量 **a** 值，记为 **a**(k)；

④ 计算 J(**a**(k))的梯度：$\nabla J(\mathbf{a}(k))$；

⑤ 沿梯度的负方向计算 **a**(k+1)；（梯度的负方向是梯度最陡下降的方向）

一般地，

$$\mathbf{a}(k+1) = \mathbf{a}(k) - \eta(k) \nabla J(\mathbf{a}(k)) \qquad (2)$$

其中，η 称为学习速率。

3. 感知器模型

现在考虑构造求解不等式 $\mathbf{a}^t \mathbf{y}_i > 0$ 的可行的准则函数，可以选择被 a 错分的样本数。

感知器准则函数为：

$$J_p(\mathbf{a}) = \sum_{\mathbf{y} \in y} (-\mathbf{a}^t \mathbf{y}) \qquad (3)$$

其中，$y(\mathbf{a})$ 是被 **a** 错分的样本集。$J_p(\mathbf{a})$ 是对所有被矢量 **a** 误分的样本求和，即与错分样本到判决边界距离之和成正比的。

$J_p(\mathbf{a})$对 **a** 求导，有

$$\nabla J_p(\mathbf{a}) = \sum_{\mathbf{y}\in y}(-\mathbf{y}) \tag{4}$$

因此，权矢量的更新规则变为：

$$\mathbf{a}(k+1) = \mathbf{a}(k) + \eta(k)\sum_{\mathbf{y}\in y}\mathbf{y} \tag{5}$$

其中，$y(\mathbf{a})$是被 a 错分的样本集。

因此，寻找解向量的批处理感知器算法：新的权向量=当前权向量+被当前权矢量误分的所有样本之和×学习因子。

4. 感知器算法的松弛过程

前面的准则函数 J(.)定义为：

$$J_p(\mathbf{a}) = \sum_{\mathbf{y}\in y}(-\mathbf{a}^t\mathbf{y}) \tag{6}$$

该准则函数存在的问题为：

- 该函数过于平滑，解矢量很可能正好落在边界上；
- 函数的解在很大程度上受到模值大的样本的支配。

因此，感知器规则可以被推广为“松弛过程”。

感知器松弛算法是对下面的准则函数进行求解：

$$J_r(\mathbf{a}) = \frac{1}{2}\sum_{\mathbf{y}\in y}\frac{(\mathbf{a}^t\mathbf{y}-b)}{\|\mathbf{y}\|^2} \tag{7}$$

这里的$y(\mathbf{a})$是满足$\mathbf{a}^t\mathbf{y} \le b$的样本集。上面的准则函数总是非负，当且仅当在所有$\mathbf{a}^t\mathbf{y} > b$时，函数值为 0。

- **松弛算法（梯度、迭代公式）：**

上面准则函数$J_r(\mathbf{a})$的梯度为：

$$\nabla J_r = \sum_{\mathbf{y}\in y}\frac{\mathbf{a}^t\mathbf{y}-b}{\|\mathbf{y}\|^2}\mathbf{y} \tag{8}$$

批处理裕量迭代公式：

$$\mathbf{a}(k+1) = \mathbf{a}(k) + \eta(k)\sum_{\mathbf{y}\in y}\frac{\mathbf{a}^t\mathbf{y}-b}{\|\mathbf{y}\|^2}\mathbf{y} \tag{9}$$

单样本裕量迭代公式为：

$$\mathbf{a}(k+1) = \mathbf{a}(k) + \eta(k)\frac{\mathbf{a}^t\mathbf{y}(k)-b}{\|\mathbf{y}(k)\|^2}\mathbf{y}(k) \tag{10}$$

- **松弛算法的实质：**

对上面公式（8）—（10）进行变换，如下：

$$\nabla J_r = \sum_{\mathbf{y}\in y} \frac{\mathbf{a}^t\mathbf{y}-b}{\|\mathbf{y}\|^2}\mathbf{y} = \sum_{\mathbf{y}\in y}\left(\frac{\mathbf{a}^t\mathbf{y}-b}{\|\mathbf{y}\|}\cdot\frac{\mathbf{y}}{\|\mathbf{y}\|}\right) \tag{11}$$

$$\mathbf{a}(k+1) = \mathbf{a}(k) + \eta(k)\sum_{\mathbf{y}\in y}\left(\frac{\mathbf{a}^t\mathbf{y}-b}{\|\mathbf{y}\|}\cdot\frac{\mathbf{y}}{\|\mathbf{y}\|}\right) \tag{12}$$

$$\mathbf{a}(k+1) = \mathbf{a}(k) + \eta(k)\frac{\mathbf{a}^t\mathbf{y}(k)-b}{\|\mathbf{y}(k)\|}\cdot\frac{\mathbf{y}(k)}{\|\mathbf{y}(k)\|} \tag{13}$$

由此可见，松弛算法的本质是使用规范化样本进行学习，即学习中主要利用样本在特征空间中的方向信息进行学习。

二、 实验方法 或 算法流程步骤

实验用到的方法和步骤如下：

1. 数据预处理

为了方便，实验中需要对数据进行预处理。对于一个样本 y_i，如果有 $a^t y_i > 0$ 就标记为 ω_1，如果 $a^t y_i < 0$ 就标记为 ω_2。这样我们可以用一种规范化操作来简化两类样本的训练过程，也就是说对属于 ω_2 的样本，用负号表示而不是标记 ω_2。有了规范化，我们就可以忘掉这些标记，而寻找一个对所有样本都有 $a^t y_i > 0$ 的权向量 a。

为了简化数据处理，并且为了使得分类判别函数通过原点，我们需要对所有的样本进行规范化处理，由 d 维增加到 $d+1$ 维，即对每个样本第一维加 1，如 $y_i = [1, x_1, x_2, ..., x_d]^{\mathrm{T}}$。这样做将在二维空间中不过原点的直线映射为在三维空间中过原点的平面，从而将寻找权向量 w 和权阈值 w_0 的问题简化为寻找一个权向量 $a = [w_0, w]^{\mathrm{T}}$。

2. 批处理感知器算法

➢ 批处理感知器算法的实现步骤：

① 初始化：权向量 **a**，学习步长 $\eta(\cdot)$，误差阈值θ，令 $k = 0$

② 将是被 **a** 错分的样本找出，形成样本集 y_k。

③ do $k = k + 1$

④ $\mathbf{a}(k+1) = \mathbf{a}(k) + \eta(k)\sum_{\mathbf{y}\in y_k}\mathbf{y}$

⑤ until $\left|\eta(k)\sum_{\mathbf{y}\in y_k}\mathbf{y}\right| > \theta$

⑥ return **a**

⑦ end

将上述的批处理感知器算法应用在 ω_1 和 ω_2 训练数据上，设阈值 $\theta = 0.5$，初始化权向量为零向量 $a(1) = 0$，取学习率 $\eta = 0.2, 0.4, 0.6$ 进行训练。并绘出在不同学习率情况下，$J_p(a)$ 的值随迭代次数 k 变化的曲线。

重复实验：将上述的批处理感知器算法应用到 ω_2 和 ω_3 训练数据上，设阈值 $\theta=0.5$，初始化权向量为零向量 $a(1)=0$，取学习率 $\eta=0.3,0.5,0.7$ 进行重新训练，并绘出在不同学习率情况下，$J_p(a)$ 的值随迭代次数 *k* 变化的曲线。

3. 批处理裕量松弛算法

- 批处理裕量松弛算法的实现步骤：

① 初始化：权向量 **a**，学习步长 $\eta(\cdot)$，边沿裕量 *b*，令 *k* = 0

② do $k = (k+1) \bmod n$

③ $y_k = \{\ \}$

④ $j = 0$

⑤ do $j = (j+1)$

⑥ if $\mathbf{a}^t\mathbf{y}(j) \le b$ then 把 $y(j)$ 加进 y_k

⑦ until $j = n$

⑧ $$\mathbf{a}(k+1)=\mathbf{a}(k)+\eta(k)\sum_{\mathbf{y}\in y_k}\frac{\mathbf{a}^t\mathbf{y}(k)-b}{\|\mathbf{y}(k)\|^2}\mathbf{y}(k)$$

⑨ until $y_k = \{\ \}$

⑩ return **a**

⑪ end

将上述的批处理裕量松弛算法应用在 ω_1 和 ω_3 训练数据上，设间隔 $b=0.1$，初始化权向量为零向量 $a(1)=0$，取学习率 $\eta=0.01,0.02$ 进行训练。并绘出在不同学习率情况下，$J_r(a)$ 的值对于训练回合数 *k* 的函数曲线。

重复实验：将上述的批处理裕量松弛算法应用在 ω_1 和 ω_3 训练数据上，设间隔 $b=0.5$，初始化权向量为零向量 $a(1)=0$，取学习率 $\eta=0.01,0.02$ 进行训练。并绘出在不同学习率情况下，$J_r(a)$ 的值对于训练回合数 *k* 的函数曲线。

三、 实验结果

1. 批处理感知器算法

- **将批处理感知器算法应用在 ω_1 和 ω_2 训练数据上：**

取学习率 $\eta=0.2,0.4,0.6$ 进行训练。并绘出在不同学习率情况下，$J_p(a)$ 的值随迭代次数 *k* 变化的曲线，如图 1-1 所示。

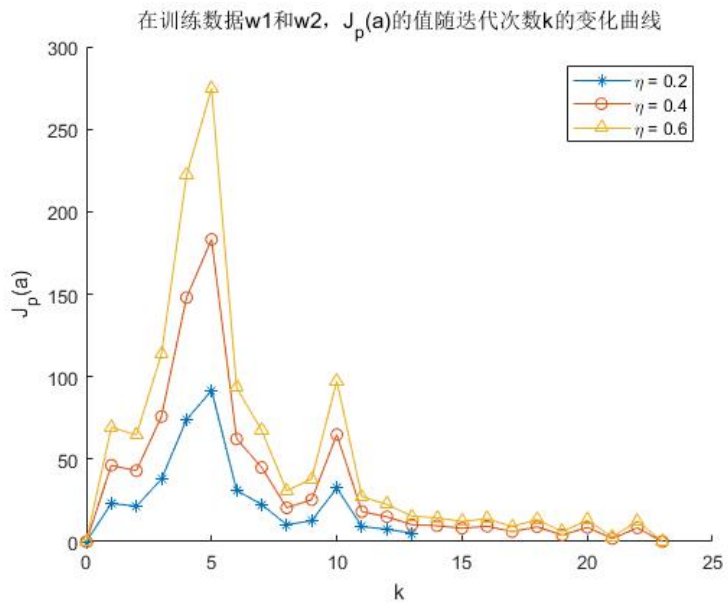

图 1-1. 在不同学习率情况下， $J_p(a)$ 的值随迭代次数 k 变化的曲线

分类结果：利用学习得到的权值向量 **a** = [20.40, -18.24, 20.46]（这里取学习率 $\eta = 0.6$）计算得到决策面，并对 ω_1 和 ω_2 训练数据进行分类，分类结果如图 1-2 所示。

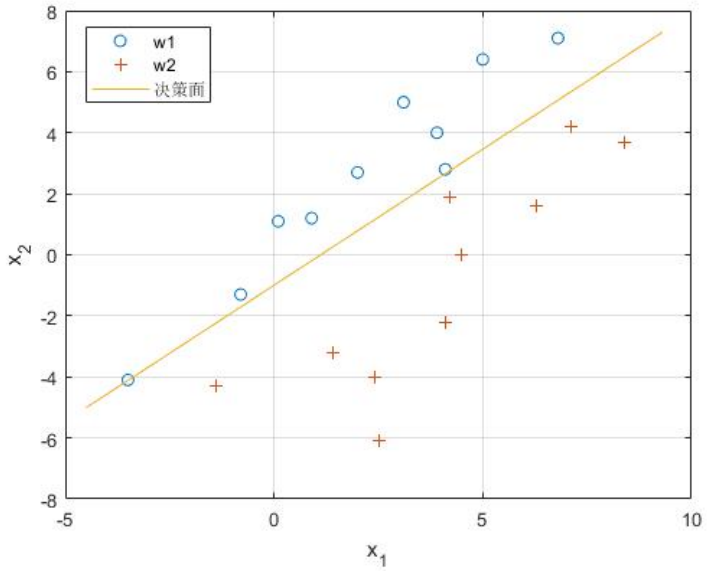

图 1-2. 学习得到的决策面对 ω_1 和 ω_2 训练数据的分类结果

- **将批处理感知器算法应用在 ω_2 和 ω_3 训练数据上：**

取学习率 $\eta = 0.3, 0.5, 0.7$ 进行重新训练，并绘出在不同学习率情况下， $J_p(a)$ 的值随迭代次数 k 变化的曲线，如图 1-3 所示。

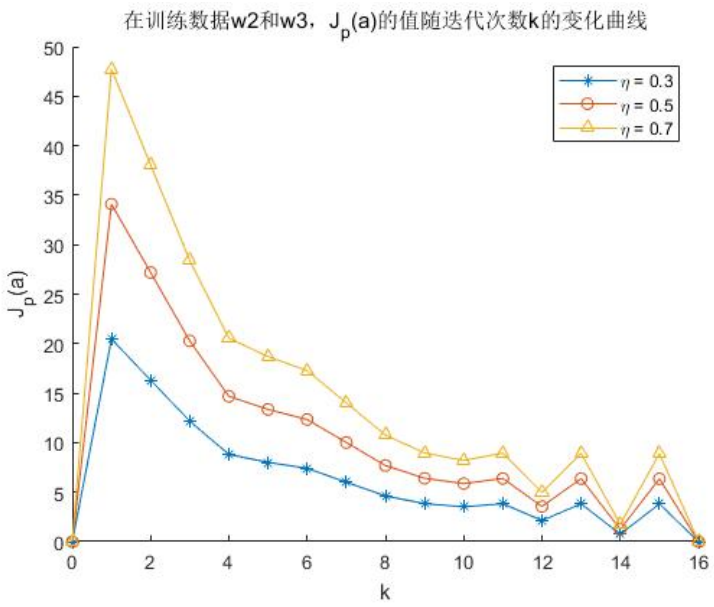

图 1-3. 在不同学习率情况下， $J_p(a)$ 的值随迭代次数 k 变化的曲线

分类结果：利用学习得到的权值向量 **a** = [-13.30, 28.95, -34.02]（这里取学习率 $\eta = 0.7$）计算得到决策面，并对 ω_2 和 ω_3 训练数据进行分类，分类结果如图 1-4 所示。

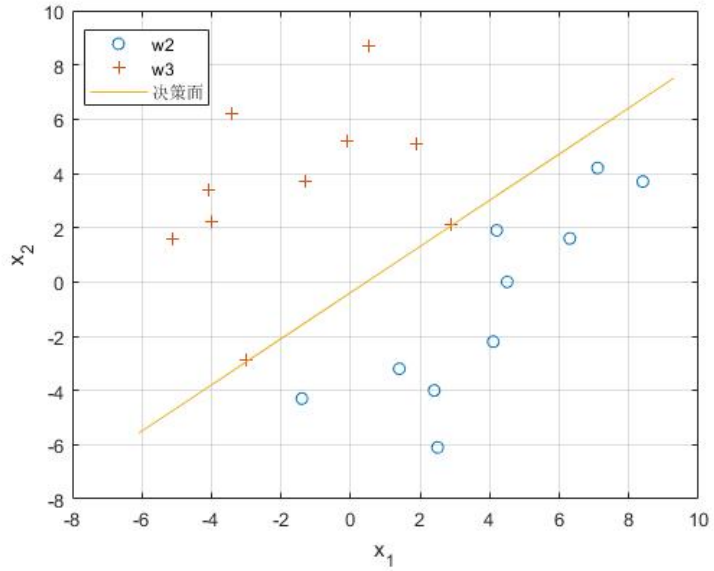

图 1-4. 学习得到的决策面对 ω_2 和 ω_3 训练数据的分类结果

2. 批处理裕量松弛算法

- **设裕量间隔** $b=0.1$：

取学习率 $\eta=0.01$ 进行训练。绘出 $J_r(a)$ 的值对于训练回合数 k 的函数曲线，如图 2-1 所示。

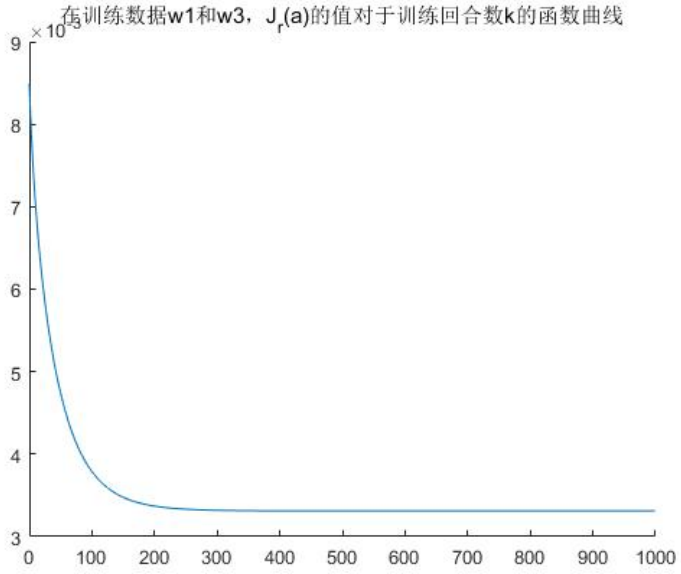

图 2-1. $J_r(a)$ 的值对于训练回合数 k 的函数曲线（$b=0.1$，$\eta=0.01$）

改变学习率 $\eta=0.02$ 进行训练。绘出 $J_r(a)$ 的值对于训练回合数 k 的函数曲线，如图 2-2 所示。

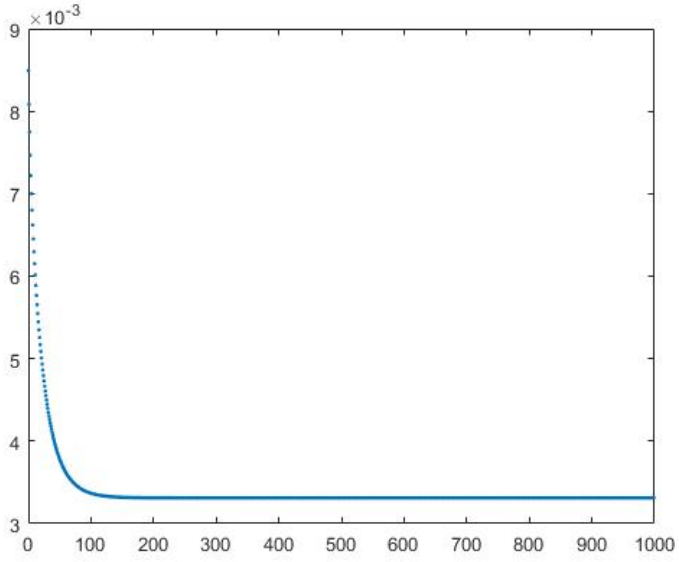

图 2-2. $J_r(a)$ 的值对于训练回合数 k 的函数曲线（$b=0.1$，$\eta=0.02$）

- **设裕量间隔** $b=0.5$：

取学习率 $\eta=0.01$ 进行训练。绘出 $J_r(a)$ 的值对于训练回合数 k 的函数曲线，如图 2-3 所示。

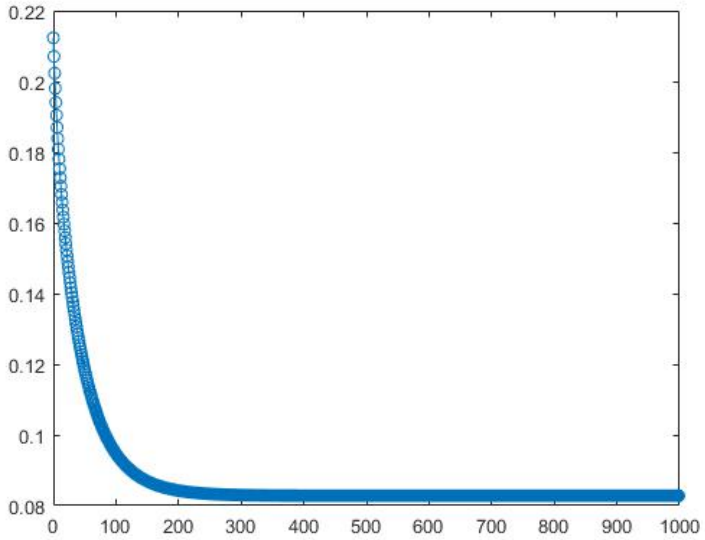

图 2-3. $J_r(a)$ 的值对于训练回合数 k 的函数曲线（$b=0.5$，$\eta=0.01$）

改变学习率 $\eta=0.02$ 进行训练。绘出 $J_r(a)$ 的值对于训练回合数 k 的函数曲线，如图 2-4 所示。

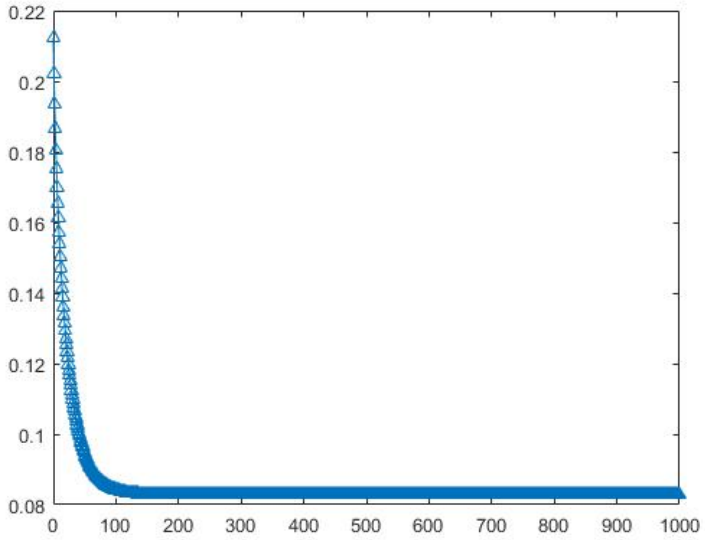

图 2-4. $J_r(a)$ 的值对于训练回合数 k 的函数曲线（$b=0.5$，$\eta=0.02$）

- **分类结果：**

利用学习得到的权值向量（这里取间隔 $b=0.5$，学习率 $\eta=0.02$）计算得到决策面，并对 ω_1 和 ω_3 训练数据进行分类，分类结果如图 2-5 所示。

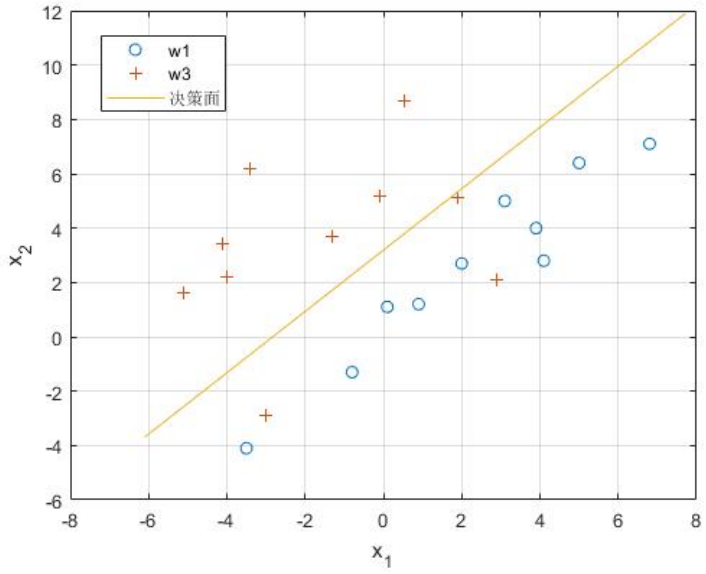

图 2-5. 学习得到的决策面对 ω_1 和 ω_3 训练数据的分类结果

四、 讨论与分析

1. 批处理感知器算法

➢ **将批处理感知器算法应用在 ω_1 和 ω_2 训练数据上：**

图 1-1 是批处理感知器算法利用 ω_1 和 ω_2 作为训练数据，由图 1-1，在不同学习率情况下，$J_p(a)$ 的值随迭代次数 k 变化的曲线，可以看出：经过有限次迭代运算，算法必定收敛，随着迭代次数的增加，准则函数 $J_p(a)$ 不断收敛。

改变学习率，$\eta = 0.2, 0.4, 0.6$，由图 1-1 可以看出，$\eta = 0.2$ 的准则函数 $J_p(a)$ 变化曲线最低，曲线变化较为平缓；$\eta = 0.4$ 次之；$\eta = 0.6$ 的准则函数 $J_p(a)$ 变化曲线最高，曲线变化剧烈。即，学习率较低，其对应的准则函数 $J_p(a)$ 值小，变化较平缓。

➢ **将批处理感知器算法应用在 ω_2 和 ω_3 训练数据上：**

图 1-2 是批处理感知器算法利用 ω_2 和 ω_3 作为训练数据，由图 1-2，在不同学习率情况下，$J_p(a)$ 的值随迭代次数 k 变化的曲线，同样可以得出结论：

① **经过有限次迭代运算，算法收敛；**

② **随着迭代次数的增加，准则函数 $J_p(a)$ 不断收敛；**

③ **收敛速度取决于权向量的初始值和学习率：学习率较低，其对应的准则函数**

$J_p(a)$ **值小，变化较平缓。**

- **结果对比：**

对比图 1-1 和图 1-2，可以发现，后者所需要的迭代次数比前者少，学习率也不同。因此，对于不同的训练样本，使得准则函数 $J_p(a)$ 收敛所需要的迭代次数不同，学习率也需要随之做调整。

此外可以发现，图 1-1 和图 1-2 的准则函数都可以收敛到 0，说明训练样本都是线性可分的，这可以通过图 1-3 和图 1-4 加以验证：利用学习得到的权值向量 a 计算得到分类决策面，这些决策面能够对训练样本进行分类。**因此，只要训练样本集是线性可分的，对于任意的初始权值矢量，经过有限次迭代运算，算法必定收敛。**

2. 批处理裕量松弛算法

图 2-1 和图 2-2 是批处理裕量松弛算法利用 ω_1 和 ω_3 作为训练数据，间隔 $b=0.1$，学习率分别为 $\eta=0.01$ 和 $\eta=0.02$；图 2-3 和图 2-4 是批处理裕量松弛算法利用 ω_1 和 ω_3 作为训练数据，间隔 $b=0.5$，学习率分别为 $\eta=0.01$ 和 $\eta=0.02$。

- **不同的间隔对准则函数曲线收敛率的影响：**

将图 2-1 和图 2-3 进行合并，得到学习率为 $\eta=0.01$ 的情况下，$b=0.1$ 和 $b=0.5$ 的收敛曲线图，如图 4-1 所示。

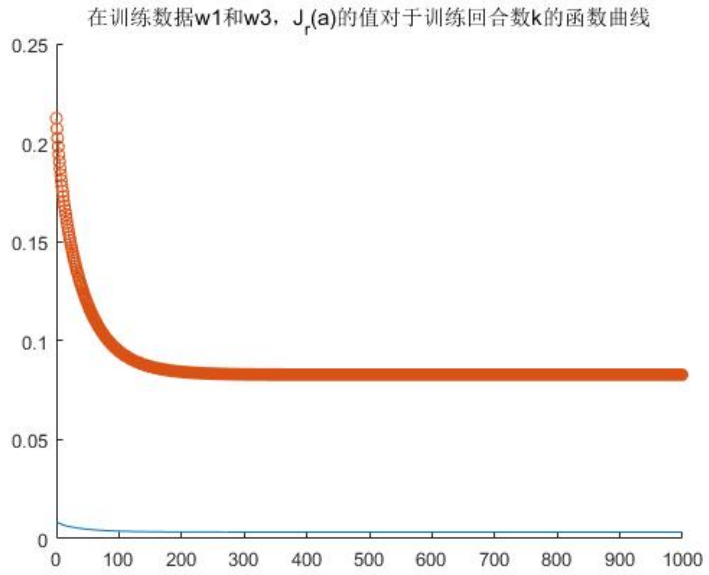

图 4-1. $J_r(a)$ 的值对于训练回合数 k 的函数曲线

（$\eta=0.01$，上面 $b=0.5$、下面 $b=0.1$）

将图 2-2 和图 2-4 进行合并，得到学习率为 $\eta=0.02$ 的情况下，$b=0.1$ 和 $b=0.5$ 的

收敛曲线图，如图 4-2 所示。

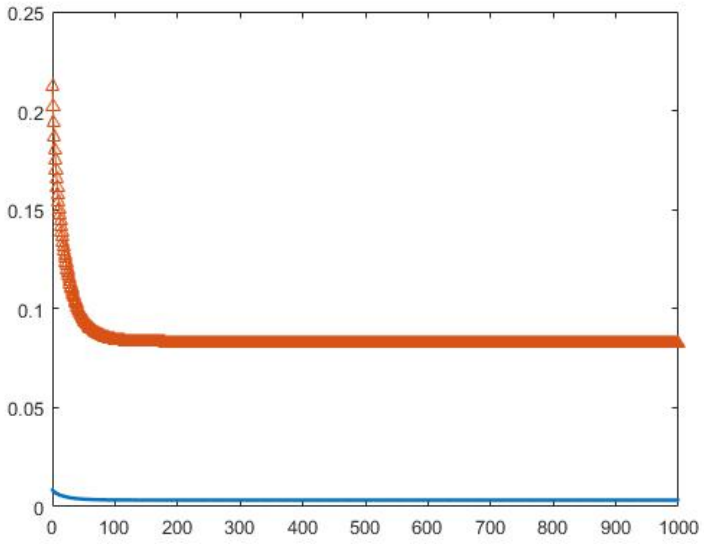

图 4-2. $J_r(a)$ 的值对于训练回合数 k 的函数曲线

（$\eta = 0.02$，上面 $b = 0.5$、下面 $b = 0.1$）

由图 4-1 和图 4-2 可以知道：间隔 b 不同时，准则函数 $J_r(a)$ 的收敛速率不同，b 大时，$J_r(a)$ 曲线的初始值较大；b 小时，$J_r(a)$ 曲线的初始值较小。这是因为：由上面的原理可知，感知器算法和松弛算法都是错误驱动的，它们的准则函数只与错分样本有关。由上面公式（7）可知，松弛算法的准则函数为 $J_r(\mathbf{a}) = \frac{1}{2}\sum_{\mathbf{y}\in y}\frac{(\mathbf{a}^t\mathbf{y} - b)}{\|\mathbf{y}\|^2}$，$y(\mathbf{a})$ 是满足 $\mathbf{a}^t\mathbf{y} \le b$ 的样本集。当 b 大时，容许更多的错分样本，因此 $J_r(a)$ 大。

- **不同的间隔和学习率对准则函数曲线收敛率的影响：**

将图 2-1、图 2-2、图 2-3 和图 2-4 进行合并（取 300 次迭代），得到不同间隔和不同学习率情况下的收敛曲线图，如图 4-3 所示。

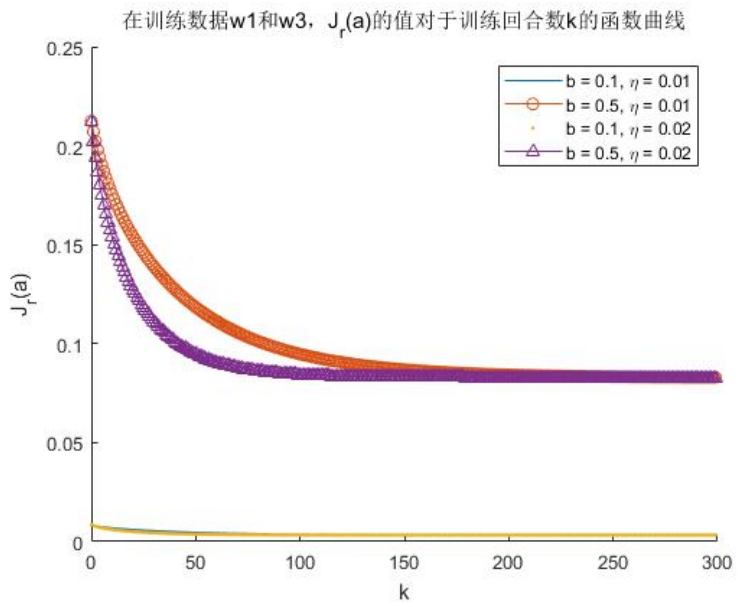

图 4-3. 在不同的间隔和学习率下， $J_r(a)$ 的值对于训练回合数 k 的函数曲线

由图 4-3 可知，**间隔 *b* 的大小决定了准则函数的初始值的大小**：间隔 *b* 大的准则函数的初始值大；而**学习率的大小决定了收敛速度的快慢**：学习率高的收敛速度快。

图 4-3 的准则函数曲线都无法收敛到 0，这是因为样本线性不可分，所以算法无法收敛，这可以通过图 2-5 加以验证：利用学习得到的权值向量 a 计算得到分类决策面，不存在某一个决策面能够对训练样本进行线性分类。

3. 批处理感知器算法和批处理裕量松弛算法的比较与分析

由上面的批处理感知器算法和批处理裕量松弛算法的实验结果分析可以知道：

对于线性可分的数据来说松弛算法可以收敛到解向量。与感知器相同，松弛算法对于线性不可分数据是不收敛的。

- **二者之间的共同点：**

① 批处理感知器算法和批处理裕量松弛算法都是只关注被错分的样本，它们的准则函数只与错分样本有关；

② 批处理感知器算法和批处理裕量松弛算法对于线性不可分的数据是不收敛的。对于线性可分的数据，可以在有限步内找到解向量。收敛速度取决于权向量的初始值和学习率。

- **二者之间的不同点：**

① 批处理感知器算法 $J_p(a)$ 的梯度是不连续的（因为 $J_p(a)$ 分段线性），函数过于平滑，收敛较慢，而且其在解区边界很光滑导致可能收敛到解区边界，对泛化性不利；其次，其得到的解向量可能依赖于模值最大的样本向量；

② 批处理裕量松弛算法使用规范化样本进行学习，即学习中主要利用样本在特征空间的方向信息进行学习，可以避免上面①中批处理感知器算法存在的问题，准则函数 $J_r(a)$ 收敛较快，解向量不依赖于模值最大的样本向量。

总结：感知器是最简单的可以“学习”的机器，是解决线性可分的最基本方法。也是很多复杂算法的基础。批处理裕量松弛算法是感知器的算法的一种有效的推广。

附录.

1. 批处理感知器算法

```
%%----------------Proj05-01：感知器算法--------------%%
%%----------Proj05-01-exp1：批处理感知器算法-----------%%
clear; clc;
%%第一类
w1 = [0.1 1.1; 6.8 7.1; -3.5 -4.1; 2.0 2.7; 4.1 2.8;
    3.1 5.0; -0.8 -1.3; 0.9 1.2; 5.0 6.4; 3.9 4.0];
%%第二类
w2 = [7.1 4.2; -1.4 -4.3; 4.5 0.0; 6.3 1.6; 4.2 1.9;
    1.4 -3.2; 2.4 -4.0; 2.5 -6.1; 8.4 3.7; 4.1 -2.2];
%%第三类
w3 = [-3.0 -2.9; 0.54 8.7; 2.9 2.1; -0.1 5.2; -4.0 2.2;
    -1.3 3.7; -3.4 6.2; -4.1 3.4; -5.1 1.6; 1.9 5.1];
%%第四类
w4 = [-2.0 -8.4; -8.9 0.2; -4.2 -7.7; -8.5 -3.2; -6.7 -4.0;
    -0.5 -9.2; -5.3 -6.7; -8.7 -6.4; -7.1 -9.7; -8.0 -6.3];
%%样本数据规范化
Y_w1 = [ones(size(w1, 1), 1), w1];
Y_w2 = [ones(size(w2, 1), 1), w2];
Y_w3 = [ones(size(w3, 1), 1), w3];
Y_w4 = [ones(size(w4, 1), 1), w4];

%% 在 w1 和 w2 的训练数据上实验
figure(1); title('训练数据 w1 和 w2 的分布');%显示训练数据 w1 和 w2 的分布
plot(w1(:, 1), w1(:, 2), 'o');
hold on; grid on; plot(w2(:, 1), w2(:, 2), '+');
xlabel('x_1'); ylabel('x_2');
legend('w1', 'w2');
%初始化
a = [0 0 0]'; eta = [0.1, 0.2, 0.4]; theta = 0.5;
Y = [Y_w1; -Y_w2]';%两类，一类标为+1，另一类为-1

figure(2); hold on; title('在训练数据 w1 和 w2，J_p(a)的值随迭代次数 k 的变化曲线');
for i = 1: size(eta, 2)
    [ak, Jp_a, k] = batch_perceptron(Y, a, eta(i), theta);
    if eta(i) == eta(1)
        plot(0: k-1, Jp_a, '-*');
    end
    if eta(i) == eta(2)
        plot(0: k-1, Jp_a, '-o');
    end
    if eta(i) == eta(3)
```

```
        plot(0: k-1, Jp_a, '-^');
    end
end
xlabel('k'); ylabel('J_p(a)');
legend('\eta = 0.2', '\eta = 0.4', '\eta = 0.6');
%% 分类
figure(3); title('训练数据 w1 和 w2 的分类');%显示训练数据 w1 和 w2 的分布
plot(w1(:, 1), w1(:, 2), 'o');
hold on; grid on; plot(w2(:, 1), w2(:, 2), '+');
xmin = min(min(w1(:,1)),min(w2(:,1)));
xmax = max(max(w1(:,1)),max(w2(:,1)));
xindex = xmin-1: (xmax-xmin)/100: xmax+1;
yindex = -ak(2)*xindex/ak(3) - ak(1)/ak(3);
plot(xindex, yindex);
xlabel('x_1'); ylabel('x_2');
legend('w1', 'w2', '决策面');
%%
%% 在 w2 和 w3 的训练数据上实验
figure(4);  title('训练数据 w2 和 w3 的分布');%显示训练数据 w2 和 w3 的分布
plot(w2(:, 1), w2(:, 2), 'o');
hold on; grid on; plot(w3(:, 1), w3(:, 2), '+');
xlabel('x_1'); ylabel('x_2');
legend('w2', 'w3');
%初始化
a = [0 0 0]'; eta = [0.1, 0.2, 0.4]; theta = 0.5;
Y = [Y_w2; -Y_w3]';%两类，一类标为+1，另一类为-1

figure(5); hold on; title('在训练数据 w2 和 w3，J_p(a)的值随迭代次数 k 的变化曲线');
for i = 1: size(eta, 2)
    [ak, Jp_a, k] = batch_perceptron(Y, a, eta(i), theta);
    if eta(i) == eta(1)
        plot(0: k-1, Jp_a, '-*');
    end
    if eta(i) == eta(2)
        plot(0: k-1, Jp_a, '-o');
    end
    if eta(i) == eta(3)
        plot(0: k-1, Jp_a, '-^');
    end
end
xlabel('k'); ylabel('J_p(a)');
legend('\eta = 0.3', '\eta = 0.5', '\eta = 0.7');
%% 分类
figure(6); title('训练数据 w1 和 w2 的分类');%显示训练数据 w1 和 w2 的分布
```

```
plot(w2(:, 1), w2(:, 2), 'o');
hold on; grid on; plot(w3(:, 1), w3(:, 2), '+');
xmin = min(min(w2(:,1)),min(w3(:,1)));
xmax = max(max(w2(:,1)),max(w3(:,1)));
xindex = xmin-1: (xmax-xmin)/100: xmax+1;
yindex = -ak(2)*xindex/ak(3) - ak(1)/ak(3);
plot(xindex, yindex);
xlabel('x_1'); ylabel('x_2');
legend('w2', 'w3', '决策面');

%% 子函数
function [a, Jp_a, k] = batch_perceptron(Y, a, eta, theta)
%批处理感知器算法
%输入：Y 为样本数据，a 为初始化的权值向量，learning_rate 为学习率，theta 为阈值
%输出：a 为权值向量，Jp_a 为准则函数，k 为收敛时的迭代步数
    k = 0;
    Jp_a = [];%准则函数
    while(1)
        k = k + 1;
        Yk = Y(:, a'*Y <= 0);%找出错分的点，形成集合 Yk
        Jp_a = [Jp_a, sum(- a'*Yk, 2)];%计算准则函数
        d_Jp_a = sum(Yk, 2);
        a = a + eta * d_Jp_a;%更新权值
        if norm(eta * d_Jp_a) < theta
            break;
        end
    end
end
```

2. 批处理裕量松弛算法

```
%%----------------Proj05-01：感知器算法--------------%%
%%----------Proj05-01-exp2：批处理裕量松弛算法-----------%%
clear; clc;
%%第一类
w1 = [0.1 1.1; 6.8 7.1; -3.5 -4.1; 2.0 2.7; 4.1 2.8;
    3.1 5.0; -0.8 -1.3; 0.9 1.2; 5.0 6.4; 3.9 4.0];
%%第二类
w2 = [7.1 4.2; -1.4 -4.3; 4.5 0.0; 6.3 1.6; 4.2 1.9;
    1.4 -3.2; 2.4 -4.0; 2.5 -6.1; 8.4 3.7; 4.1 -2.2];
%%第三类
w3 = [-3.0 -2.9; 0.54 8.7; 2.9 2.1; -0.1 5.2; -4.0 2.2;
    -1.3 3.7; -3.4 6.2; -4.1 3.4; -5.1 1.6; 1.9 5.1];
%%第四类
w4 = [-2.0 -8.4; -8.9 0.2; -4.2 -7.7; -8.5 -3.2; -6.7 -4.0;
```

```
    -0.5 -9.2; -5.3 -6.7; -8.7 -6.4; -7.1 -9.7; -8.0 -6.3];
%%样本数据规范化
Y_w1 = [ones(size(w1, 1), 1), w1];
Y_w2 = [ones(size(w2, 1), 1), w2];
Y_w3 = [ones(size(w3, 1), 1), w3];
Y_w4 = [ones(size(w4, 1), 1), w4];

%% 在 w1 和 w3 的训练数据上实验
figure(1); title('训练数据 w1 和 w3 的分布');%显示训练数据 w1 和 w2 的分布
plot(w1(:, 1), w1(:, 2), 'o');
hold on; grid on; plot(w3(:, 1), w3(:, 2), '+');
xlabel('x_1'); ylabel('x_2');
legend('w1', 'w3');
%初始化
a = [0 0 0]'; eta = [0.01, 0.02]; b = [0.1 0.5];
Y = [Y_w1; -Y_w3]';%两类，一类标为+1，另一类为-1

figure(2); hold on; title('在训练数据 w1 和 w3，J_r(a)的值对于训练回合数 k 的函数曲线');
for i = 1: size(eta, 2)
    for j = 1: size(b, 2)
        [ak, Jr_a, k] = batch_margin_relaxation(Y, a, eta(i), b(j));
        if b(j) == b(1) && eta(i) == eta(1)
            plot(0: k-2, Jr_a, '-'); hold on;
        end
        if b(j) == b(1) && eta(i) == eta(2)
            plot(0: k-2, Jr_a, '.'); hold on;
        end
        if b(j) == b(2) && eta(i) == eta(1)
            plot(0: k-2, Jr_a, '-o');
        end
        if b(j) == b(2) && eta(i) == eta(2)
            plot(0: k-2, Jr_a, '-^');
        end
    end
end
xlabel('k'); ylabel('J_r(a)');
legend('b = 0.1, \eta = 0.01', 'b = 0.5, \eta = 0.01', 'b = 0.1, \eta = 0.02',
'b = 0.5, \eta = 0.02');

%% 分类
figure(3); title('训练数据 w1 和 w3 的分类');%显示训练数据 w1 和 w2 的分布
plot(w1(:, 1), w1(:, 2), 'o');
hold on; grid on; plot(w3(:, 1), w3(:, 2), '+');
xmin = min(min(w1(:,1)),min(w3(:,1)));
```

```
xmax = max(max(w1(:,1)),max(w3(:,1)));
xindex = xmin-1: (xmax-xmin)/100: xmax+1;
yindex = -ak(2)*xindex/ak(3) - ak(1)/ak(3);
plot(xindex, yindex);
xlabel('x_1'); ylabel('x_2');
legend('w1', 'w3', '决策面');

%% 子函数
function [a, Jr_a, k] = batch_margin_relaxation(Y, a, eta, b)
%批处理裕量松弛算法
%输入：Y 为样本数据，a 为初始化的权值向量，learning_rate 为学习率，theta 为阈值
%输出：a 为权值向量，Jp_a 为准则函数，k 为收敛时的迭代步数
    k = 0;
    Jr_a = [];%准则函数
    while(1)
        k = k + 1;
        Yk = Y(:, a'*Y <= b);
        if isempty(Yk) || k > 300  %%判断样本集是否为空
            break;
        end
        sum1 = [0 0 0]';
        sum2 = 0;
        for i = 1: size(Yk, 2)
            sum1 = sum1 + (b - a'*Yk(:, i)) / (norm(Yk(:, i))^2) * Yk(:, i);
            sum2 = sum2 + (a'*Yk(:, i) - b)^2 / (norm(Yk(:, i))^2);
        end
        a = a + eta * sum1;%更新权值
        Jr_a = [Jr_a, 0.5 * sum2];%计算准则函数
    end
end
```