

1.实验内容

1.1 实验内容

以实验文件夹中的 “ex2data1.txt” 、 “ex2data2.txt” 文件中的数据为实验样本，使用不同类型的分类器对其进行分类。

- 1) 将样本分为训练集和测试集，按不同的分类规则设计分类器，画出两类样本概率分布的决策边界，并计算训练样本的错误率。
- 2) 对比不同分类规则设计的分类器，分析其性能优劣并找出最优分类规则。

1.2 实验平台

本实验是在 Windows 10 系统下使用 MATLAB 语言在 MATLAB2015a 平台上编程实现的。

1.3 实验数据

如图 1-3-1 所示，实验样本为二维样本，文件中数据矩阵的每一行为一个样本，矩阵的第一列为 x_1 ，第二列为 x_2 ，第三列为实际类别 y 。

	x_1	x_2	类别
第 1 个样本	$x_1(1)$	$x_2(1)$	$y(1)$
.....
第 m 个样本	$x_1(m)$	$x_2(m)$	$y(m)$

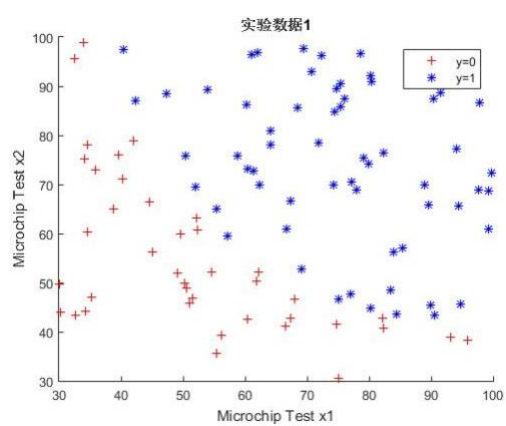
图 1-3-1 实验数据

如图 1-3-2 所示，文件 “ex2data1.txt” 中的样本是二维线性可分的，文件 “ex2data2.txt” 中的样本是二维线性不可分的。

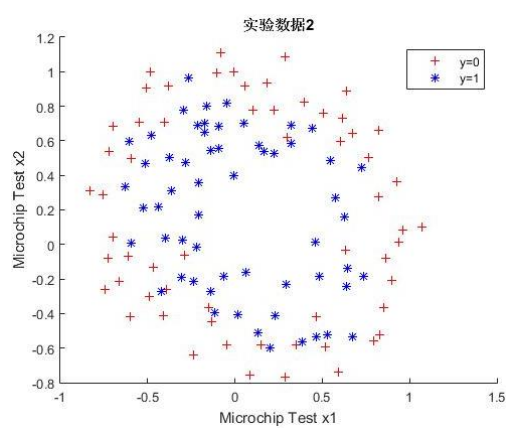
其中，如图 1-3-2(a)所示，实验数据 1 由 100 个实验样本组成，实际类别分为 0,1 两个类别，label=0 的样本 40 个，label=1 的样本 60 个。

如图 1-3-2(b)所示数据 2 由 118 个实验样本组成，实际类别分为 0,1 两个类

别，label=0 的样本 60 个， label=1 的样本 58 个。



(a)



(b)

图 1-1-2 实验数据分布

本实验选取实验数据样本的 70%作为训练样本，剩余 30%作为测试样本。

2.代码实现

2.1 贝叶斯分类器

2.1.1 贝叶斯分类器实验原理

对于二分类问题，它们的先验概率分别为 $P(\omega_1)$ 和 $P(\omega_2)$ ，根据最小错误率准则，决策规则为

$$\begin{cases} x \in \omega_1, & P(\omega_1) > P(\omega_2) \\ x \in \omega_2, & \text{else} \end{cases} \quad (2-1-1)$$

x 的类条件概率密度为 $p(x|w_i)(i=1,2)$ ，称为似然函数。根据贝叶斯公式，在模式 x 出现的条件下，两类的后验概率为：

$$p(w_i | x) = \frac{p(x|w_i)P(w_i)}{p(x)} = \frac{p(x|w_i)P(w_i)}{\sum_{i=1}^2 p(x|w_i)P(w_i)} \quad (2-1-2)$$

观察上式，要比较两类的概率，只需比较分子即可，因此，决策规则为

$$\begin{cases} x \in \omega_1, & P(x|\omega_1)P(\omega_1) > P(x|\omega_2)P(\omega_2) \\ x \in \omega_2, & \text{else} \end{cases} \quad (2-1-3)$$

则有判别函数

$$g_i(x) = P(x|\omega_i)P(\omega_i) \quad (2-1-4)$$

假设 $P(x|\omega_i)$ 服从正态分布，即当 $P(x|\omega_i) \sim N(\mu, \Sigma)$ 时，判别函数为

$$g_i(x) = \frac{P(\omega_i)}{(2\pi)^{n/2}|\Sigma_i|^{1/2}} \exp\left(-\frac{1}{2}(x - \mu_i)^T \Sigma_i^{-1}(x - \mu_i)\right) \quad (2-1-5)$$

取上式的对数，去掉无关项，判别函数化简为

$$g_i(x) = -\frac{1}{2}(x - \mu_i)^T \Sigma_i^{-1}(x - \mu_i) - \frac{d}{2} \ln |\Sigma_i| + P(\omega_i) \quad (2-1-6)$$

对于二分类问题，决策面方程为

$$g_1(x) = g_2(x) \quad (2-1-7)$$

2.1.2 算法流程与代码实现

1.代码实现步骤:

- | | |
|-------|---|
| 步骤 1: | 将实验数据分为训练集与测试集。
提取实验数据 <code>data</code> 的 70%作为训练集 <code>train_0</code> 与 <code>train_1</code> ，其他
剩余实验数据 <code>data</code> 的 30%作为训练集 <code>test</code> 。 |
| 步骤 2: | 计算先验概率。
根据样本分布情况计算训练集的先验概率 <code>p_w_0</code> 和 <code>p_w_1</code> 。 |
| 步骤 3: | 计算均值。
使用均值函数 <code>mean</code> 计算训练样本的均值向量 <code>mean_train_0</code> 和
<code>mean_train_1</code> 。 |
| 步骤 4: | 计算方差。
使用方差函数 <code>var</code> 计算训练样本的协方差矩阵 <code>var_train_0</code> 和
<code>var_train_1</code> 。 |
| 步骤 5: | 确定判别函数。
根据式(2-1-6)计算判别函数 $g_i(x)$ ，并对训练集 <code>test</code> 中的每一个
样本进行判决，得到判决结果 <code>label_test</code> 。 |
| 步骤 6: | 计算错误率。
比较计算出的 <code>label_test</code> 与测试集 <code>test</code> 实际的类别，得到错误
率。 |
| 步骤 7: | 根据式(2-1-7)，使用 <code>contour</code> 函数画出决策面曲线。 |

图 2-1-1 贝叶斯分类器实现步骤

2.主要代码（完整代码见 5.2 节）

（1）使用均值函数 `mean` 计算每类训练样本的均值：

```
train_0_mean = mean(train_0);    train_1_mean = mean(train_1);
```

（2）使用方差函数 `var` 计算每类训练样本的方差：

```
train_0_var = cov(train_0_x, train_0_y);    train_1_var = cov(train_1_x, train_1_y);
```

（3）由步骤 5，根据式(2-1-6)计算判别函数 $g_i(x)$ ：

`%inv` 矩阵求逆；`det` 计算方阵的行列式

```

g0    =    (-1/2)*((x-train_0_mean))*inv(train_0_var)*((x-train_0_mean)') -
(1/2)*(log(det(train_0_var))) + log(p_w_0);

```

```

g1    =    (-1/2)*((x-train_1_mean))*inv(train_1_var)*((x-train_1_mean)') -
(1/2)*(log(det(train_1_var))) + log(p_w_1);

```

（4）使用 `contour` 函数画出决策曲线：

```

***画出决策曲线

```

```

***获取数据范围，并选取数据点计算 label***%

```

```

xmin = min(data_x); xmax = max(data_x);  ymin = min(data_y); ymax = max(data_y);
xvals = linspace(xmin,xmax,1000);        yvals = linspace(ymin,ymax,1000);
[gridx,gridy] = meshgrid(xvals, yvals);

```

```

***计算矩阵 Z 用于画图***%

```

```

Z = zeros(size(gridx));

```

```

for i = 1:size(xvals,2)

```

```

    for j = 1:size(yvals,2)

```

```

        x = [xvals(i),yvals(j)];

```

```

***计算判别函数***%

```

```

g0    =    (-1/2)*((x-train_0_mean))*inv(train_0_var)*((x-train_0_mean)') -
(1/2)*(log(det(train_0_var))) + log(p_w_0);

```

```

g1    =    (-1/2)*((x-train_1_mean))*inv(train_1_var)*((x-train_1_mean)') -
(1/2)*(log(det(train_1_var))) + log(p_w_1);

```

```

    if (g0>g1)

```

```

        Z(i,j) = 0;

```

```

    else

```

```

        Z(i,j) = 1;

```

```

    end

```

```

end

```

```

end

```

2.1.3 实验结果与分析

由 2.1.2 节所述，将实验数据的 70%作为训练集，30%作为训练集，设计贝叶斯分类器对其进行分类。

如图 2-1-2 所示的决策面曲线可以看到，贝叶斯分类器对于实验数据样本 1 的分类效果明显好于样本 2，从图中可以看出，相对于样本 2，样本 1 的分布更接近于二维高斯分布，因此相对于样本 2，样本 1 更适合使用贝叶斯分类器进行分类。

对于实验数据 ex2data1.txt 中的样本，使用贝叶斯分类器的错误率为 0.0667。

对于实验数据 ex2data2.txt 中的样本，使用贝叶斯分类器的错误率为 0.6111。

分类结果也表明样本 1 更适合使用贝叶斯分类器。

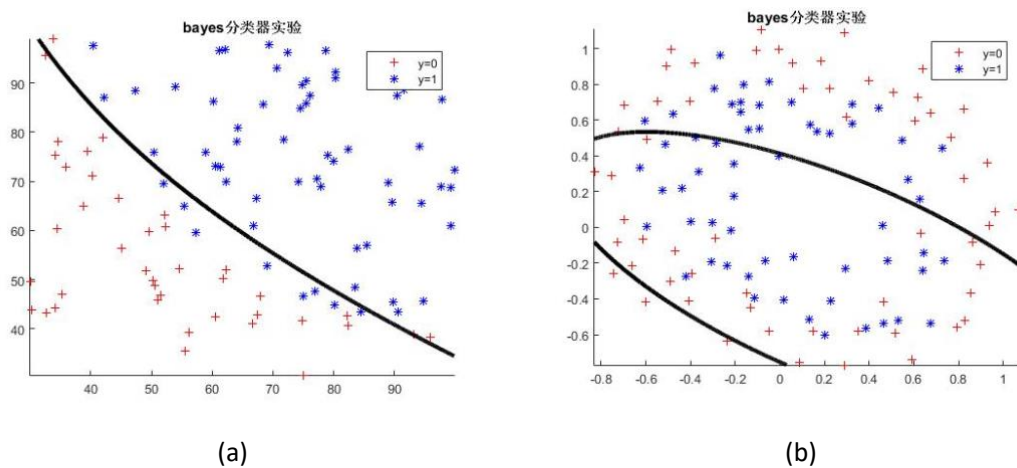


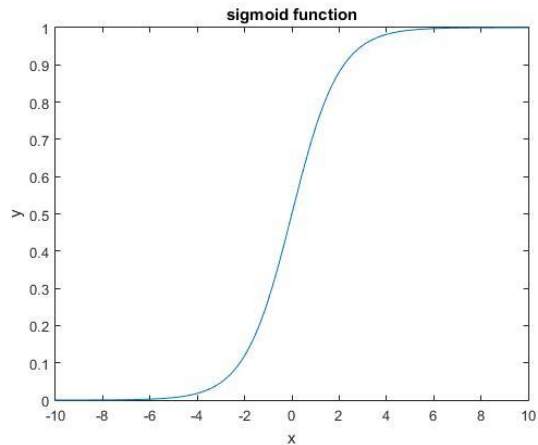
图 2-1-2

2.2 logistic 回归

2.2.1logistics 回归实验原理

(1) sigmoid 函数:

$$s(x) = \frac{1}{1 + e^{-x}} \quad (2-2-1)$$



sigmoid 函数

如上图所示，sigmoid 函数也称为 logistic 函数，用于隐层神经元输出，其取值范围为 $\{0,1\}$ ，它可以将一个实数映射到 $\{0,1\}$ 的区间，可以用来做二分类。在特征相差较复杂或是相差不是特别大时效果比较好，因此，sigmoid 函数常被用作神经网络的激活函数。

(2) Logistic Regression:

在线性回归问题中，通过线性回归方程对一个或多个自变量和因变量之间关系进行建模，并根据训练样本求解模型参数，最终得到自变量 x 与因变量 y 之间的函数关系。

二分类问题与线性回归问题相似，只是我们现在想要预测的 y 值只能取两个值，0 和 1。我们可以忽略 y 是等于 0 或 1 的离散值的事实来处理分类问题，然后用我们以前的线性回归算法来预测。

首先需要找一个合适的预测函数 (hypothesis)，一般表示为 $h_{\theta}(x)$ 函数，该函数就是我们需要找的分类函数，它用来预测输入数据的判断结果。

构造一个 cost 函数 (损失函数)，记为 $J(\theta)$ ，表示所有训练数据预测值与实际类别的偏差。由于预测值只取 0 和 1，整理代价函数为：

$$J(\theta) = -\frac{1}{m} \left[\sum_{i=1}^m y^{(i)} \log h_{\theta}(x^{(i)}) + (1 - y^{(i)}) \log (1 - h_{\theta}(x^{(i)})) \right] \quad (2-2-2)$$

$J(\theta)$ 函数的值越小表示预测函数越准确 (即 $h_{\theta}(x)$ 函数越准确)，所以需要找到 $J(\theta)$ 函数的最小值。找函数的最小值有不同的方法，使用梯度下降法 (Gradient Descent) 寻找该最小值。

根据最大似然理论，简化模型更新方程为：

$$\theta_j = \theta_j + \alpha(y^{(i)} - h_{\theta}(x^{(i)}))x_j^{(i)} \quad (2-2-3)$$

其中 $h_{\theta}(x)$ 为 logistic 函数。

2.2.2 算法流程与代码实现

1. 代码实现步骤：

步骤 1:	初始化 设置初始参数：模型参数 θ , 迭代次数 ite , 学习率 α 将训练数据映射到高维，扩展后的数据存储在 $train_mat$
步骤 2:	计算预测函数 $h_{\theta}(x)$ 根据式(2-2-1)计算预测值 h_theta_x
步骤 3:	计算代价函数 计算代价函数 $cost$
步骤 4:	更新模型参数 根据式(2-2-3)更新 θ

图 2-2-1 logistic 回归算法步骤

2. 主要代码（完整代码见 5.3 节）

```

***计算 logistic 函数***%
h_theta_x = 1./(exp(-(train_mat*theta)) +1 );

***计算更新方程及代价函数***%

***创建存储代价函数的向量以观察收敛过程（损失下降过程）***%
cost = zeros(ite,1);

for i = 1:ite

    ***sigmoid function***%
    h_theta_x = 1./(exp(-(train_mat*theta)) +1 );

    ***使用梯度下降法更新模型系数***%
    theta(1,1) = theta(1,1) - (alpha) * sum( h_theta_x - train_y );
    theta(2,1) = theta(2,1) - (alpha) * sum( (h_theta_x - train_y).*train_mat(:,2) );
    theta(3,1) = theta(3,1) - (alpha) * sum( (h_theta_x - train_y).*train_mat(:,3) );
    theta(4,1) = theta(4,1) - (alpha) * sum( (h_theta_x - train_y).*train_mat(:,4) );

```



```

theta(5,1) = theta(5,1) - (alpha) * sum( (h_theta_x - train_y).*(train_mat(:,5) ));

theta(6,1) = theta(6,1) - (alpha) * sum( (h_theta_x - train_y).*(train_mat(:,6) ));

***计算 cost***%

cost(i,1) = (1/2/train_num) * sum((h_theta_x - train_y).*(h_theta_x - train_y));

end

```

2.2.3 实验结果与分析

由 2.1.2 节所述,将实验数据的 70%作为训练集,30%作为训练集,利用 logistic 回归对其分类并计算错误率,画出决策曲线。

1.对实验样本 ex2data1 进行实验:

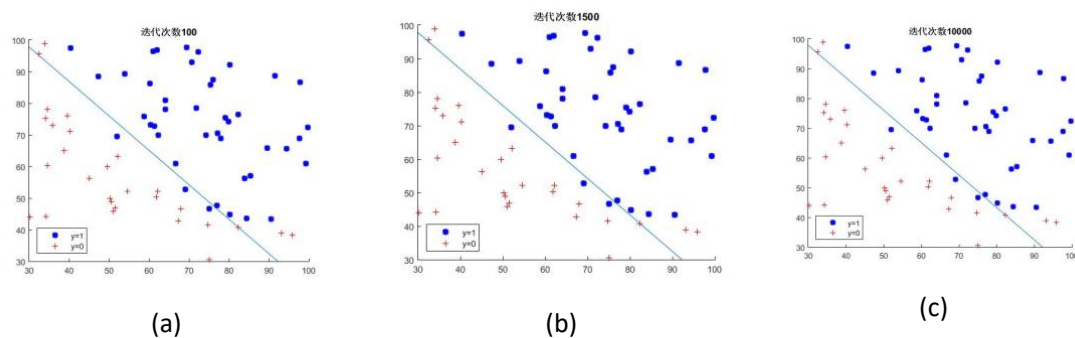


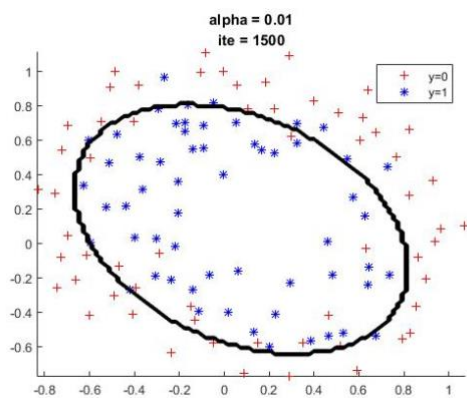
图 2-2-2 样本 ‘ex2data1’ 分类

如图 2-2-2(a)所示,使用 logistic 回归对样本 ex2data1 分类,使用梯度下降法拟合参数,迭代次数设为 100 次,最终代价函数下降为 $\text{cost} = 0.2035$,对测试集分类的错误率为 0.2000。

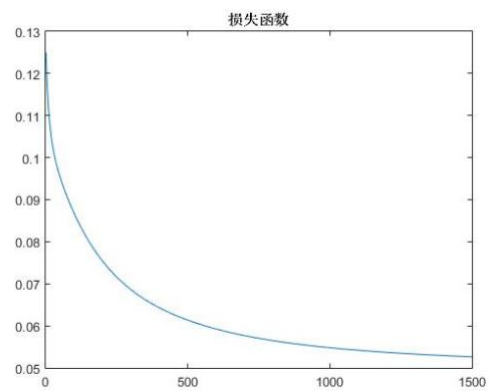
如图 2-2-2(b)所示,使用 logistic 回归对样本 ex2data1 分类,使用梯度下降法拟合参数,迭代次数设为 1500 次,最终代价函数下降为 $\text{cost} = 0.2035$ 对测试集分类的错误率为 0.2000。

如图 2-2-2(c)所示,使用 logistic 回归对样本 ex2data1 分类,使用梯度下降法拟合参数,迭代次数设为 10000 次,最终代价函数下降为 $\text{cost} = 0.2035$,对测试集分类的错误率为 0.2000。

2.对实验样本 ex2data1 进行实验:

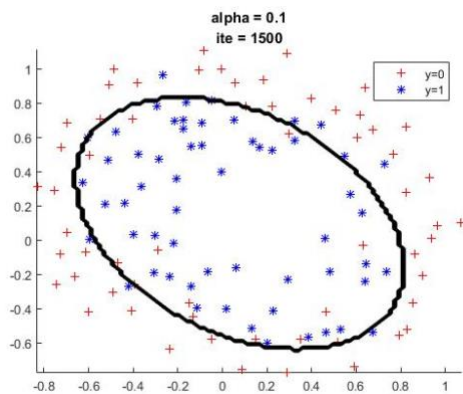


决策面

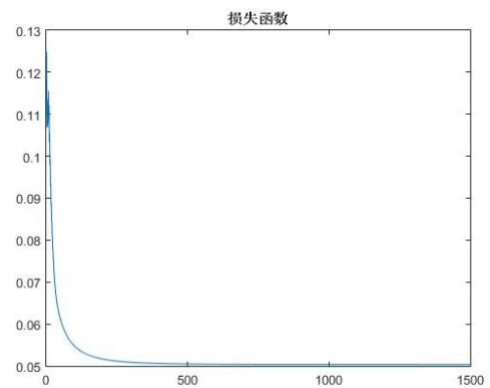


损失函数

(a) 学习率 0.01 迭代次数 1500

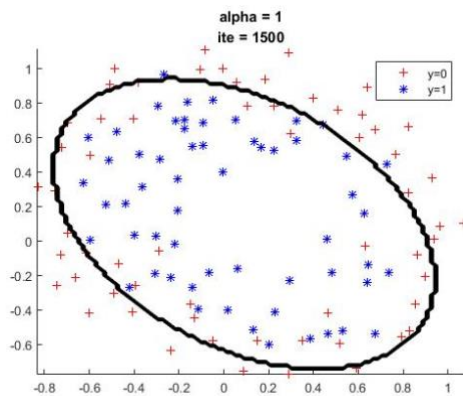


决策面

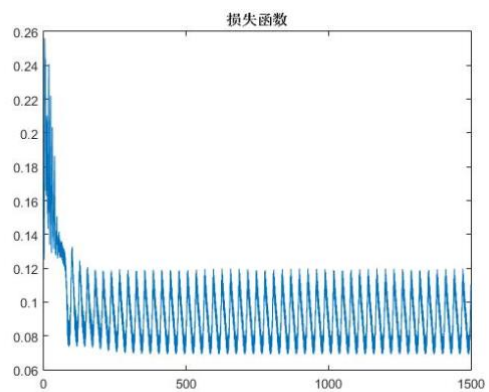


损失函数

(b) 学习率 0.1 迭代次数 1500

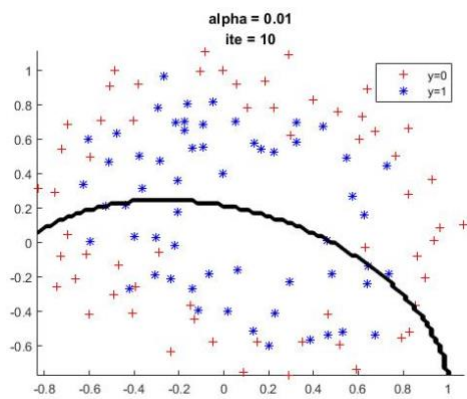


决策面

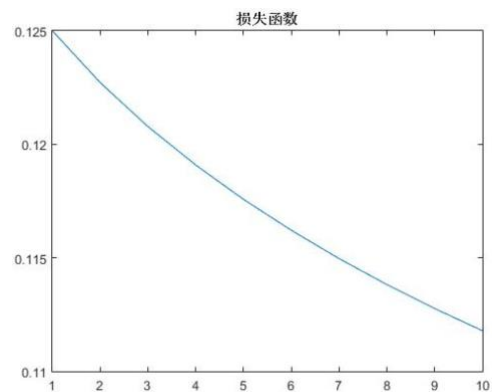


损失函数

(c) 学习率 1 迭代次数 1500

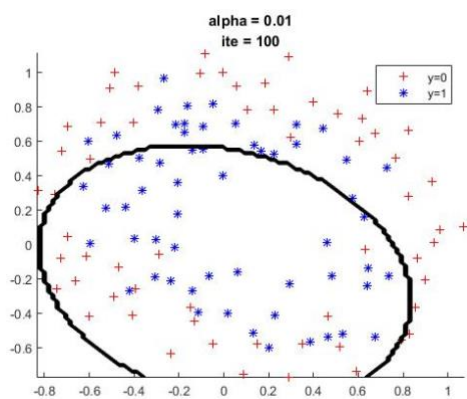


决策面

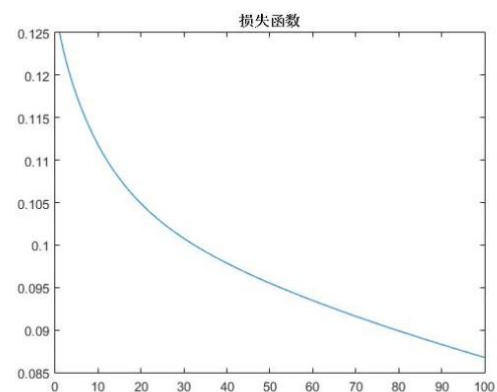


损失函数

(d) 学习率 0.01 迭代次数 10

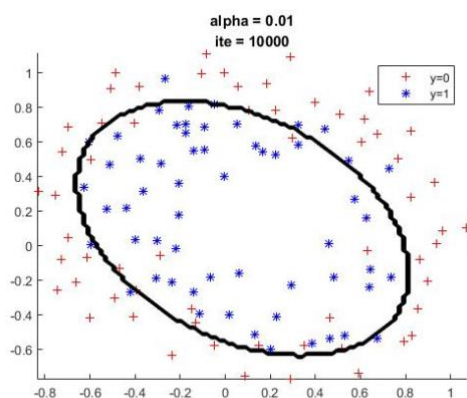


决策面

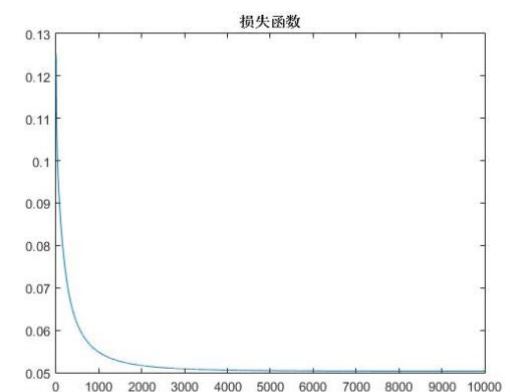


损失函数

(e) 学习率 0.01 迭代次数 100



决策面



损失函数

(f) 学习率 0.01 迭代次数 10000

图 2-2-3 样本 'ex2data2' 分类

使用 logistic 回归对样本 ex2data2 分类需要将二维样本映射到 6 维，即映射为 $\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1^2 + \theta_4 x_2^2 + \theta_5 x_1 x_2$ 。

如图 2-2-3(a)所示,使用 logistic 回归对样本 ex2data2 分类,使用梯度下降法拟合参数,学习率设置为 $\alpha=0.01$,迭代次数设为 1500 次,最终代价函数下降为 0.0526,对训练集进行测试,错误率为 $e=0.2222$ 。

如图 2-2-3(b)所示,使用 logistic 回归对样本 ex2data2 分类,使用梯度下降法拟合参数,学习率设置为 $\alpha=0.1$,迭代次数设为 1500 次,最终代价函数下降为 $\text{cost}=0.0503$,对训练集进行测试,错误率为 $e=0.2222$ 。

如图 2-2-3(c)所示,使用 logistic 回归对样本 ex2data2 分类,使用梯度下降法拟合参数,学习率设置为 $\alpha=1$,迭代次数设为 1500 次,最终代价函数下降为 $\text{cost}=0.0691$,对训练集进行测试,错误率为 $e=0.2500$ 。

如图 2-2-3(d)所示,使用 logistic 回归对样本 ex2data2 分类,使用梯度下降法拟合参数,学习率设置为 $\alpha=0.01$,迭代次数设为 10 次,最终代价函数下降为 $\text{cost}=0.1118$,对训练集进行测试,错误率为 $e=0.6667$ 。

如图 2-2-3(e)所示,使用 logistic 回归对样本 ex2data2 分类,使用梯度下降法拟合参数,学习率设置为 $\alpha=0.01$,迭代次数设为 100 次,最终代价函数下降为 $\text{cost}=0.0868$,对训练集进行测试,错误率为 $e=0.5278$ 。

如图 2-2-3(f)所示,使用 logistic 回归对样本 ex2data2 分类,使用梯度下降法拟合参数,学习率设置为 $\alpha=0.001$,迭代次数设为 10000 次,最终代价函数下降为 $\text{cost}=0.0503$,对训练集进行测试,错误率为 $e=0.2222$ 。

可以分析到 2-2-3(d)所示,学习率设置为 $\alpha=0.01$,迭代次数设为 10 次时,由于迭代次数过少,存在欠拟合现象,所以分类错误率较高。

2.3 SVM

2.3.1 SVM 支持向量机原理

1. 基本原理

SVM 的基本思想是寻找一个最优分类超平面,并基于核定理,通过非线性映射,把样本空间映射到高维空间,使在特征空间中解决样本空间中的非线性分类问题。

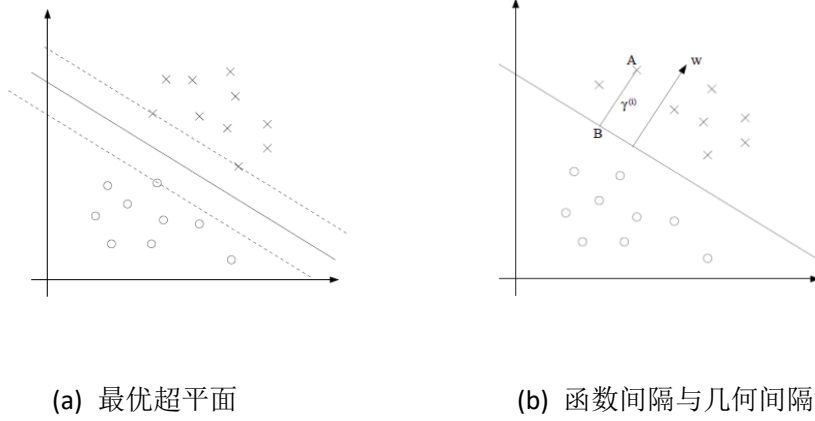


图 2-3-1 SVM 分类器图示

假设训练样本集为 $\{(a_1, y_1), (a_2, y_2), \dots, (a_m, y_m)\}$ ，其中 $a_i \in R^f$ 表示输入变量， $y_i \in \{-1, 1\}$ 为与输入变量相对应的样本标检，其中 f 为数据样本的特征数，即维数， m 为数据样本总数。设分类超平面的方程为：

$$\varpi a - \gamma = 0 \quad (2-3-1)$$

其中， γ 为超平面的偏移量，且 $\gamma \in R$ ， ϖ 为超平面的法向量，且 $\varpi \in R^f$ 。

通过以下公式来计算函数间隔为样本点到超平面的距离，设为 $j^{(i)}$

$$|a^T \varpi - r| \div \sqrt{\varpi^T \varpi} = \sqrt{a^T \varpi - r} \div \|\varpi\|_2 = y^{(i)} (a^T \varpi - r) \div \|\varpi\|_2 \quad (2-3-2)$$

并令 $j = \min_{j=1,2,\dots,m} j^{(i)}$ ，我们要做的就是找到最小函数间隔，并最大化该函数间

隔（确保分类准确性）。综上所述 SVM 优化问题可表述为：

$$\begin{aligned} \max_{\varpi, r} & 2 \frac{1}{\|\varpi\|_2} \\ \text{s.t.} & y^{(i)} (a^{(i)T} \varpi - r) \geq 1, i = 1, 2, \dots, m \end{aligned} \quad (2-3-3)$$

可以进一步将上式改写为：

$$\begin{aligned} \max_{\varpi, r} & \frac{1}{2} \|\varpi\|_2^2 \\ \text{s.t.} & D(A\varpi - re) \geq e \end{aligned} \quad (2-3-4)$$

其中， $A \in R^{M \times j}$ 为样本矩阵， $D \in R^{M \times M}$ 是对角矩阵，该矩阵对角线上的元素为样本 a 所对应的标签， $e = (1, 1, \dots, 1)^T$ 。通过求解上式就可以找到一个 margin

最大的分类超平面，与超平面平行的另外两条线到超平面的距离都等于 $\frac{1}{\|\omega\|_2}$ ，SVM 就是在相应条件约束下最大化这个距离。

2.常用核函数

线性核函数：
$$K(x_i, y_j) = x_i^T x_j \tag{2-3-5}$$

多项式核函数：
$$K(x_i, y_j) = ((x_i^T x_j) + 1)^d \tag{2-3-6}$$

径向基函数：
$$K(x_i, y_j) = \exp(-\gamma^2 \|x_i - y_j\|^2) \tag{2-3-7}$$

sigmoid 函数：
$$K(x_i, y_j) = \tanh(v^T (x_i^T x_j) + r) \tag{2-3-8}$$

2.3.2 参数选择与代码实现

1.MATLAB 自带 svm 相关函数参数分析：

[svm_struct, svIndex] = svmtrain(training, groupnames, varargin);

outclass = svmclassify(svmStruct,sample, varargin);

(1) svmtrain 函数参数：

表 2-3-1 svmtrain 函数参数

name	参数可选项及其含义
kernel_function	表示可选的核函数 <div> <div>'linear'</div> <div>选择使用线性核</div> </div> <div> <div>'rbf'</div> <div>选择使用径向基核，惩罚因子默认值为 1</div> </div> <div> <div>'polynomial'</div> <div>选择使用多项式核，默认为 3 阶</div> </div> <div> <div>'mlp'</div> <div>选择使用多层感知器内核，默认权重 1，默认偏差-1</div> </div>

(2) svmclassify 函数参数：

表 2-3-2 svmclassify 函数参数

name	参数可选项及其含义
svmStruct	创建好的分类器结构，即 svmtrain 函数的训练结果。
sample	测试数据，注 test 样本必须与训练样本具有相同的列数（即相同维数）

3. 主要代码（完整代码见 5.4 节）：

根据表 2-3-1 所述的函数参数，选择不同的核函数，对数据样本进行训练与

分类，并计算分类错误率，画出决策曲线。

(1) 使用线性核函数：

```
svmtrain(data, label, 'kernel_function', 'linear', 'showplot', true);
```

(2) 使用多项式核函数：

```
svmtrain(data, label, 'kernel_function', 'polynomial', 'polyorder', 2, 'showplot', true);
```

其中参数 'polyorder' 为指定多项式的阶数

(3) 使用径向基核函数：

```
svmtrain(data, label, 'kernel_function', 'rbf', 'rbf_sigma', 0.5, 'boxconstraint', C, 'showplot', true);
```

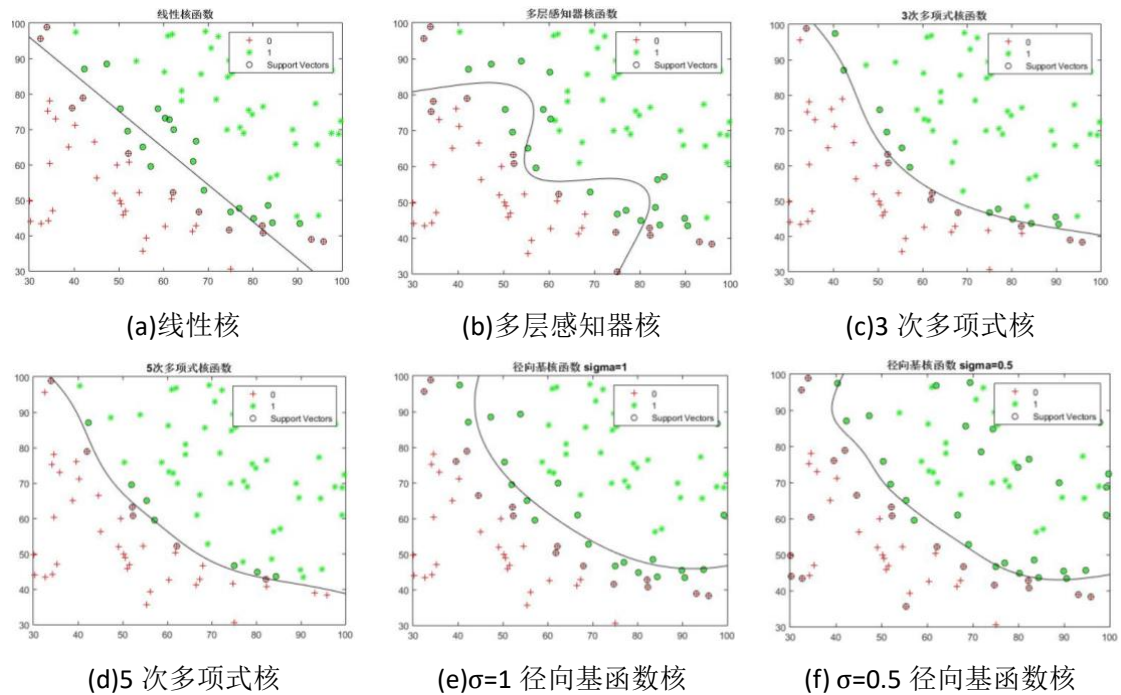
其中第参数 'sigma' 为径向基函数核中指定比例因子的正数参数。

(4) 使用多层感知器核函数：

```
svmtrain(data, label, 'kernel_function', 'mlp', 'showplot', true);
```

2.3.3 实验结果与分析

由 2.1.2 节所述，将实验数据的 70% 作为训练集，30% 作为训练集，使用 SVM 对其进行分类，并画出分类超平面。由如图 2-3-2、2-3-3 所示的决策面曲线观察分类效果。



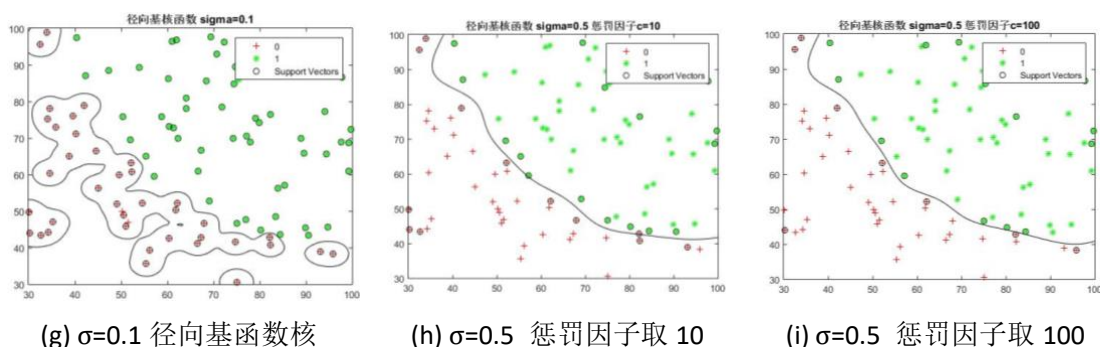


图 2-3-2 样本 ‘ex2data1’ SVM 分类结果

对于实验数据 ex2data1.txt 中的样本，使用 SVM 进行分类：

如图 2-3-2(a)所示，选用线性核的错误率为 0.0333。

如图 2-3-2(b)所示，选用多层感知器核的错误率为 0.0011。

如图 2-3-2(c)所示，选用 3 次多项式核的错误率为 0.0333。

如图 2-3-2(d)所示，选用 5 次多项式核的错误率为 0.0333。

如图 2-3-2(e)所示，选用参数 $\sigma=1$ 的径向基函数核的错误率为 0.1667。

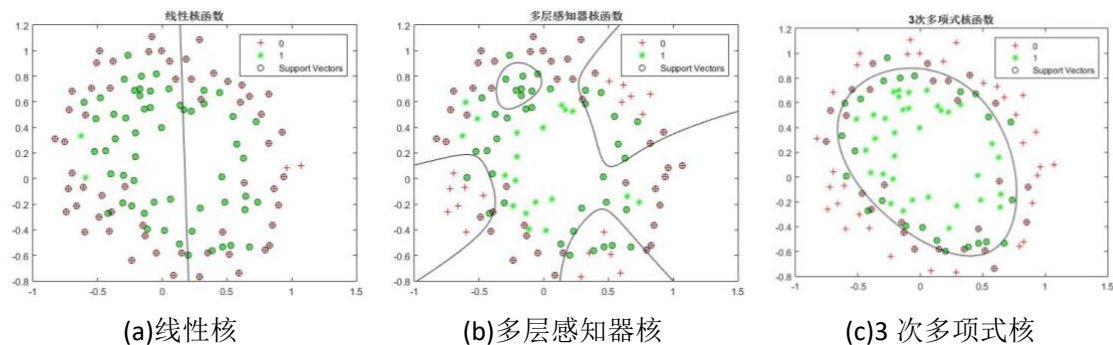
如图 2-3-2(f)所示，选用参数 $\sigma=0.5$ 的径向基函数核的错误率为 0.0667。

如图 2-3-2(g)所示，选用参数 $\sigma=0.1$ 的径向基函数核的错误率为 0.2333。

如图 2-3-2(h)所示，选用参数 $\sigma=0.5$ ，惩罚因子 c 取 10 的径向基函数核的错误率为 0.0333。

如图 2-3-2(i)所示，选用参数 $\sigma=0.5$ ，惩罚因子 c 取 100 的径向基函数核的错误率为 0.0333。

由此可以得出，对于线性可分的数据，使用线性核足以得到较好的分类结果。尤其注意到如图 2-3-2(g)所示，虽然从分类超平面看，分类效果最好，但对测试样本集进行分类，错误率反而较高，说明对于训练样本集产生了过拟合。



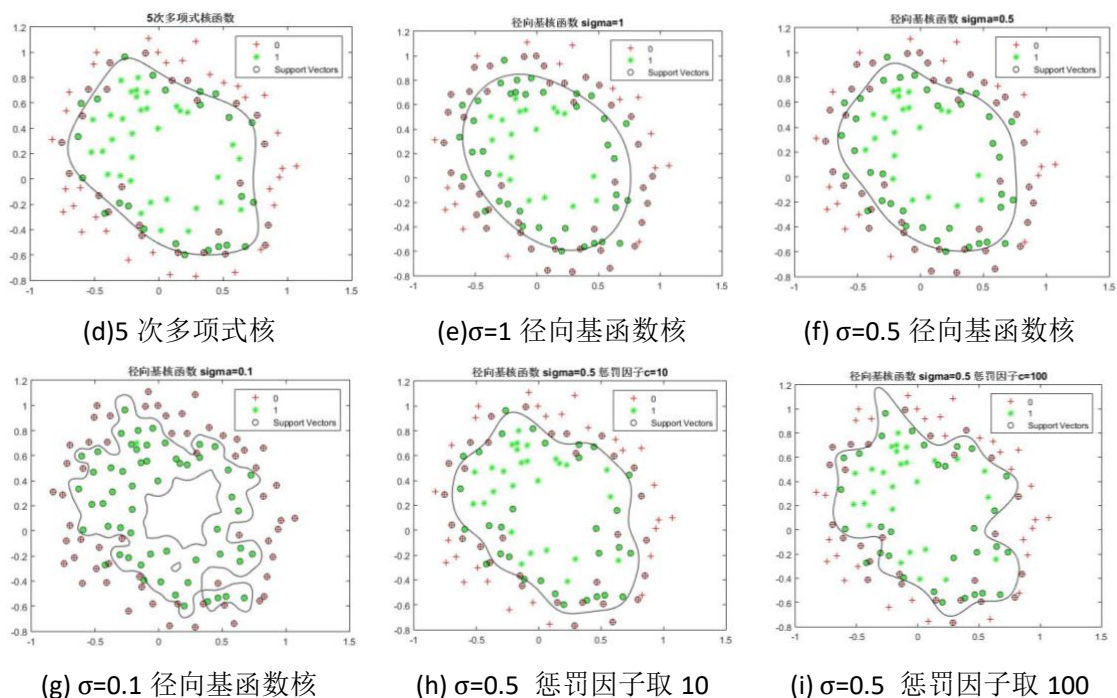


图 2-3-3 样本 ‘ex2data2’ SVM 分类结果

对于实验数据 `ex2data2.txt` 中的样本，使用 SVM 进行分类：

如图 2-3-3(a)所示，选用线性核的错误率为 0.1111。

如图 2-3-3(b)所示，选用多层感知器核的错误率为 0.1944。

如图 2-3-3(c)所示，选用 3 次多项式核的错误率为 0.0556。

如图 2-3-3(d)所示，选用 5 次多项式核的错误率为 0.0556。

如图 2-3-3(e)所示，选用参数 $\sigma=1$ 的径向基函数核的错误率为 0.0556。

如图 2-3-3(f)所示，选用参数 $\sigma=0.5$ 的径向基函数核的错误率为 0.0010。

如图 2-3-3(g)所示，选用参数 $\sigma=0.1$ 的径向基函数核的错误率为 0.1667。

如图 2-3-3(h)所示，选用参数 $\sigma=0.5$ ，惩罚因子 c 取 10 的径向基函数核的错误率为 0.1389。

如图 2-3-2(i)所示，选用参数 $\sigma=0.5$ ，惩罚因子 c 取 100 的径向基函数核的错误率为 0.1111。

可以看到，对于数据样本 2，选用参数 $\sigma=0.5$ 的径向基函数核分类效果最好，观察其分类超平面，也较为合理。其中，对于使用径向基核函数的分类，对于同一个数据样本使用，使用更小的 σ ，不一定分类效果更好，这跟样本集本身的分类有关。

2.4 神经网络分类器

2.4.1 BP 神经网络原理

1. BP 神经网络：

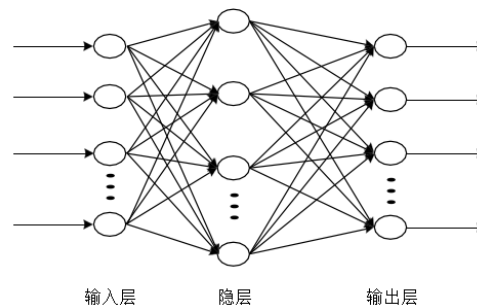


图 2-4-1 BP 神经网络结构图

BP 神经网络的基本结构如图 2-4-1 所示，其模型拓扑结构包括输入层(input)、隐层(hidden layer)和输出层(output layer)三层结构。

输入层各神经元负责接收来自外界的输入信息，并传递给中间层各神经元，中间层是内部信息处理层，负责信息变换，根据信息变化能力的需求。中间层可以设计为单隐层或者多隐层结构，最后一个隐层传递到输出层各神经元的信息，经进一步处理后，完成一次学习的正向传播处理过程，由输出层向外界输出信息处理结果。隐层节点一般采用 Sigmoid 型函数，输入和输出节点可以采用 Sigmoid 型函数或者线性函数。

2. MATLAB 自带神经网络函数及参数分析（完整代码见 5.5 节）：

（1）BP 神经网络 matlab 函数：

newff ()函数：

- 功能：建立一个前向 BP 网络
- 格式：net = newff(PR , [S1 S2... SN1], {TF1 TF2... BTF BLF PF})
- net 为新创建的 BP 神经网络；PR 为网络输入取向量取值范围的矩阵
[S1 S2... SN1] 表示网络隐含层和输出层神经元的个数
{TF1 TF2... TFN1} 表示网络隐含层和输出层的传输函数，默认为'tansig'
BTF 表示网络的训练函数，默认为'trainlm'
BLF 表示网络的权值学习函数，默认为'learngdm'
PF 表示性能数，默认为'mse'

注意，`newff` 要求每一列为一个样本，所以需要对数据样本进行转置
`train()` 函数：

- Train neural network 训练神经网络
- `[net, tr, Y, E, Pf, Af] = train(net, P, T, Pi, Ai)`

`sim()` 函数：

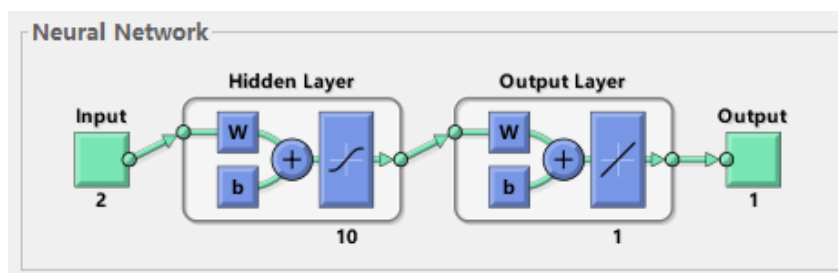
- Simulate neural network 模拟预测
- `[Y, Pf, Af, E, perf] = sim(net, P, Pi, Ai, T)` Y 即最终预测的输出

(2) 网络训练参数 `net.trainParam` 参数含义及默认值：

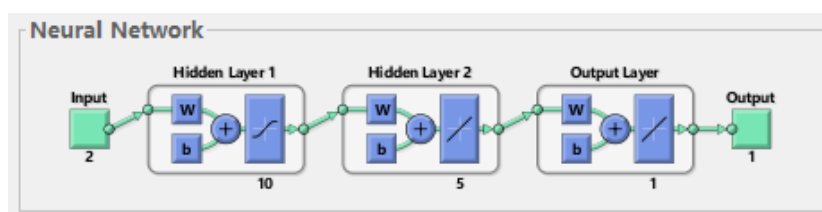
表 2-4-1 MATLABBP 神经网络函数参数

BP 神经网络参数名称	参数具体含义	系统默认值设置
<code>net.trainParam.epochs</code>	训练次数	100
<code>net.trainParam.min_grad</code>	最小梯度要求	1e-6
<code>net.trainParam.show</code>	显示训练迭代过程	25
<code>net.trainParam.time</code>	最大训练时间	Inf

2.4.2 参数选择与代码实现



(a)



(b)

图 2-4-2

如图 2-4-2 所示，设置不同的节点数进行测试，隐层节点数分别设置为 10+5

个节点（隐层 2 层）和 10 个节点（隐层 1 层）。

迭代次数 `net.trainParam.epochs` 设置为 1500 次，收敛误差 `net.trainParam.goal` 设置为 0.001，学习率 `net.trainParam.lr` 设置为 0.01。

2.4.3 实验结果与分析

由 2.1.2 节所述，将实验数据的 70% 作为训练集，30% 作为训练集。

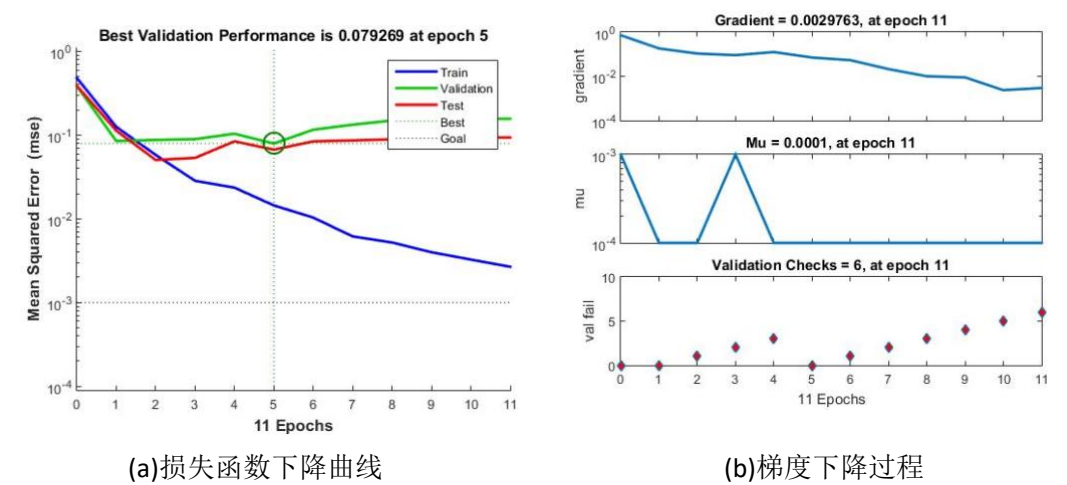


图 2-4-3 样本 ex2data1 隐层 1 层

使用如图 2-4-2(a) 所示的隐层只有 1 层的网络对实验数据 `ex2data1.txt` 中的样本进行 3 次实验，错误率分别为 0.1667，0.1000，0.0667。

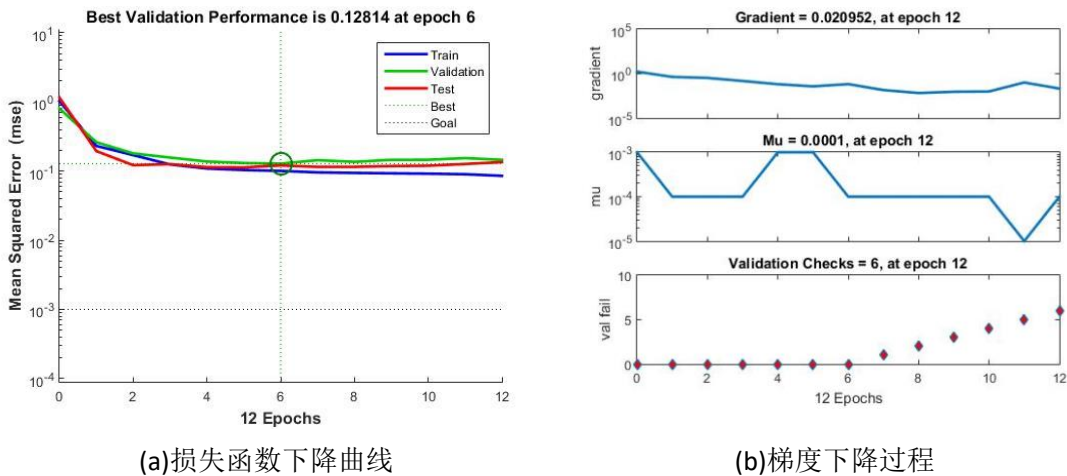


图 2-4-4 样本 ex2data2 隐层 1 层

使用如图 2-4-2(a) 所示的隐层只有 1 层的网络对实验数据 `ex2data2.txt` 中的样

本进行 3 次实验，错误率分别为 0.3056，0.3611，0.2778。

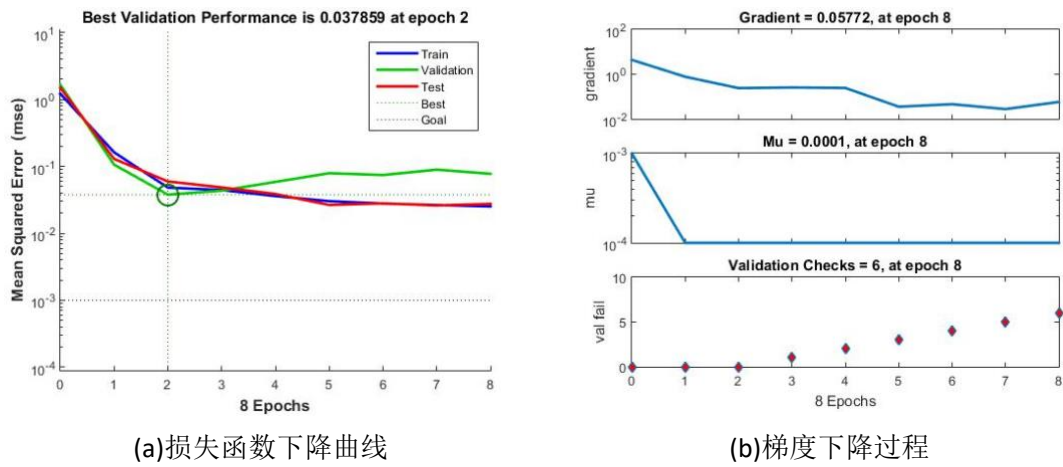


图 2-4-5 样本 ex2data1 隐层 2 层

使用如图 2-4-2(b)所示的隐层 2 层的网络对实验数据 ex2data1.txt 中的样本进行 3 次实验，错误率分别为 0.1000，0.1667，0.1000。

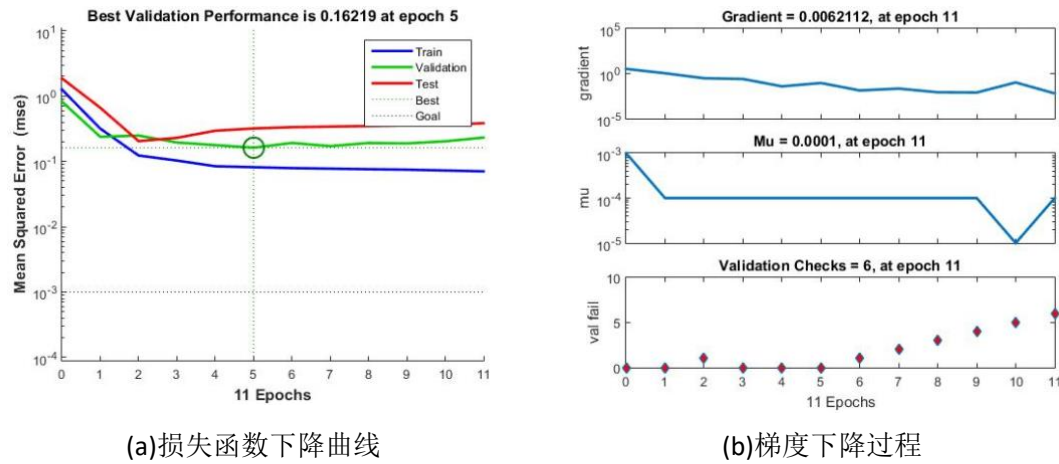
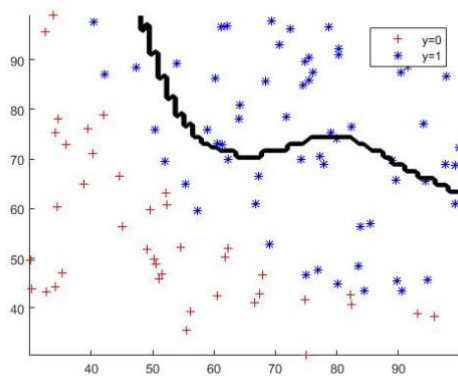


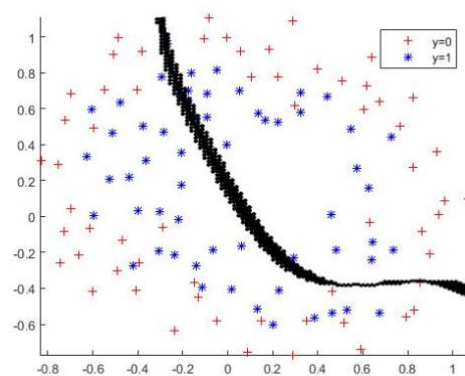
图 2-4-6 样本 ex2data2 隐层 2 层

使用如图 2-4-2(b)所示的隐层 2 层的网络对实验数据 ex2data2.txt 中的样本进行 3 次实验，错误率分别为 0.1944，0.2222，0.3056。

决策面曲线如下图 2-4-7 所示。



(a) 样本 ex2data1



(b) 样本 ex2data2

图 2-4-7 决策面曲线

3.结课分析

(1) 不同算法性能分析

如表 3-1 所示，对比不同算法对于测试集分类，错误率最小的参数。

表 3-1 ex2data1 分类性能对比

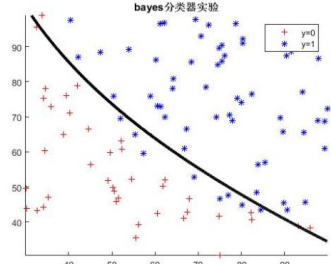
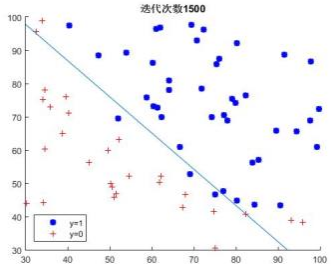
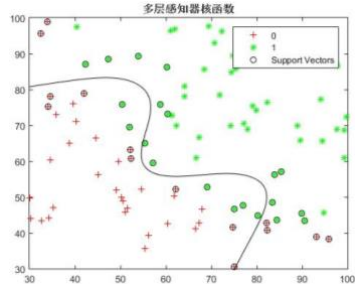
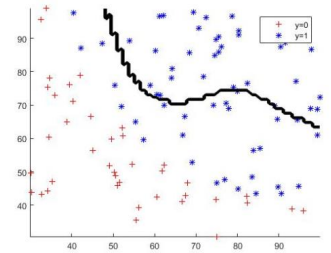
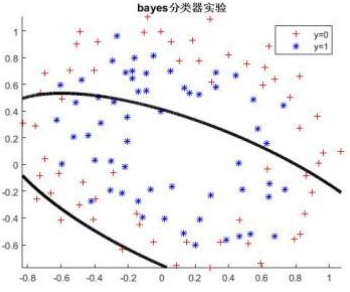
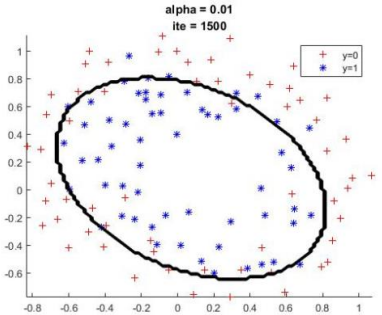
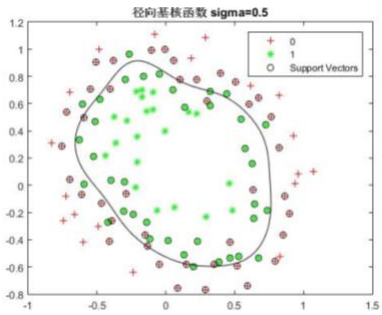
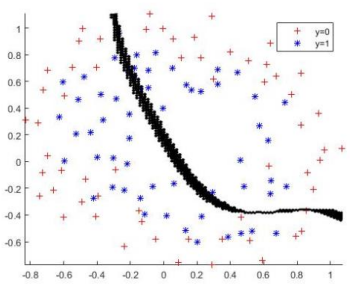
算法	分类结果分析	
贝叶斯分类器	最优参数	 <p>决策面曲线</p>
	最小错误率	
	0.0667	
Logistic 回归	最优参数	 <p>决策面曲线</p>
	ite=1500	
	最小错误率	
	0.2000	
SVM	最优参数	 <p>决策面曲线</p>
	多层感知器核	
	最小错误率	
	0.0011	
BP 神经网络	最优参数	 <p>决策面曲线</p>
	最小错误率	
	0.0667	

表 3-2 ex2data2 分类性能对比

算法	分类结果分析	
贝叶斯分类器	最优参数	 <p>决策面曲线</p>
	最小错误率	
	0.6111	
Logistic 回归	最优参数	 <p>决策面曲线</p>
	学习率 $\alpha=0.01$ ite=1500	
	最小错误率	
	0.2222	
SVM	最优参数	 <p>决策面曲线</p>
	径向基函数核 参数 $\sigma=0.5$	
	最小错误率	
	0.0010	
BP 神经网络	最优参数	 <p>决策面曲线</p>
	最小错误率	
	0.1944	

(2) 最优分类器分析

如上表 3-1 所示, 对于数据样本 ex2data1 进行分类, 最优的是使用多层感知

器核的 SVM 分类器，但是在实际应用中，由于该样本是线性可分的样本，实际不需要使用这么复杂的分类器。

如上表 3-2 所示，对于数据样本 `ex2data2` 进行分类，最优的是使用径向基函数核且取参数 $\sigma=0.5$ 的 SVM 分类器。

可以看到，我们预估的最优分类器 BP 神经网络并没有取得最好的结果，这是因为实验样本过少，网络不能得到很好的训练导致的，因此，在实际应用中，应该根据实际情况选择不同的分类器。

4.参数优化——非参数估计

对于 2.1 节所述的贝叶斯分类器设计过程，当参数未知时，可以选用非参数估计的方法对其进行估计，常用的估计数据概率密度的方法有 Parzen 窗和 Kn 近邻法，以“ex2data.txt”文件夹中的一维实验样本为例进行实验。

1.实验原理

(1) Kn 近邻法

预先确定样本总数 n 的某个函数 Kn ，然后再 x 点周围选择一个区域，调整区域体积大小，直至 Kn 个样本落入区域中。这些样本被称为点 x 的 Kn 个最近邻。如果 x 点附近的密度比较高，则 V 的体积自然就相对较小，从而可以提升分辨力；如果 x 点附近的密度比较低，则 V 的体积就较大，但一进入高密度区就会停止增长。固定样本数 Kn ，在 x 附近选取与之最近的 Kn 个样本，计算该 Kn 个样本的最小体积 V 。在 x 处的概率密度估计值为：

$$\widehat{p(x)} = \frac{Kn}{nV} \quad (4-1-1)$$

Kn 近邻估计中，估计效果取决于近邻数。

(2) Parzen 窗法

Parzen窗法可以看作是用核函数对样本在取值空间中进行插值。假设 $x \in \mathbf{R}$ ， d 为 d 维的向量，并假设每一个区域是一个超立方体，它在每一维上的棱长都是 h ，则小区域的体积为： $V=h^d$ 。

定义如下 d 维单位窗函数：

$$\varphi(u) = \begin{cases} 1, & |u_j| \leq \frac{1}{2}, j=1,2,\dots,d \\ 0, & \text{其他} \end{cases} \quad (4-1-2)$$

并使 $\varphi(u)$ 满足条件 $\varphi(u) \geq 0$ ，且 $\int \varphi(u) du = 1$ ，则落入体积 V 中的样本数为

$$k_N = \sum_{i=1}^N \varphi\left(\frac{x-x_i}{h}\right) \quad (4-1-3)$$

则此处概率密度的估计值是：

$$\hat{p}(x) = \frac{1}{N} \sum_{i=1}^N \frac{1}{V} \varphi\left(\frac{x-x_i}{h}\right) \quad (4-1-4)$$

$\frac{1}{V}\varphi(u)$ 称为窗函数或核函数，一般可以选择的窗函数有方窗、高斯窗等。

方窗的核函数为：

$$k(x, x_i) = \begin{cases} \frac{1}{h^d} & \text{若 } |x^j - x_i^j| \leq \frac{h}{2}, j=1, 2, \dots, d \\ 0 & \text{其他} \end{cases} \quad (4-1-5)$$

其中, h 为超立方体的棱长, 当样本信号是一维的高斯信号时, d 的取值为1。

正态窗函数为

$$k(x, x_i) = \frac{1}{V} \varphi(u) = \frac{1}{V} \frac{1}{\sqrt{2\pi}} \exp\left\{-\frac{1}{2}u^2\right\} \quad (4-1-6)$$

概率密度的估计式为

$$\hat{p}_N(x) = \frac{1}{N \cdot h_1} \sum_{i=1}^N \frac{1}{\sqrt{2\pi}} \exp\left[-\frac{1}{2}\left(\frac{x-x_i}{h_1}\right)^2\right] \quad (4-1-7)$$

2.实验实现（代码见 5.6 节）

（1）Kn 近邻法

步骤 1:	确定小仓的体积 kn，初始化 data_plot_t 用于统计落入小仓样本个数
步骤 2:	data_plot_t 与 kn 对比，当 data_plot_t>kn 时使用这些样本进行计算
步骤 3:	使用步骤 2 的选取样本估计 data_plot_kn
步骤 4:	重复步骤 2、3，直到遍历所有样本

图 4-2-1 Kn 近邻法实现步骤

（2）Parzen 窗法（以使用高斯窗为例）

步骤 1:	初始化小仓棱长 h1 及高斯函数方差 var
步骤 2:	根据式（4-1-2）计算窗函数，筛选计算样本
步骤 3:	根据式（4-1-6）计算核函数的值 kxx
步骤 4:	根据式（4-1-7）估计概率密度 data_plot_y_parzen
步骤 5:	重复步骤 2、3、4，直到遍历所有样本

图 4-2-2 Parzen 窗法实现步骤

3.非参数估计结果

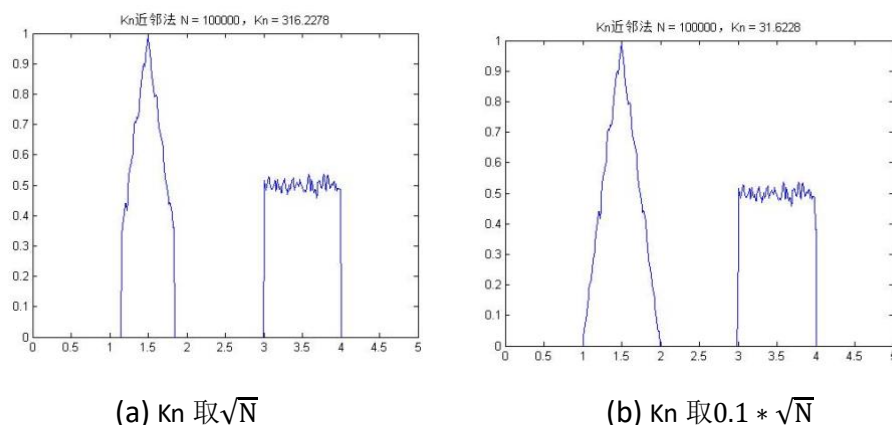


图 4-3-1 kn 近邻法实验结果

如图 4-3-1 所示的 K_n 参数取值不同的 K_n 近邻法非参数估计实验。对图(a)(b)可以看出, K_n 取值越小, 估计结果的毛刺更细, 分辨率更高, 估计更细致, K_n 近邻法的估计结果越接近真实的概率密度。由此可以验证, 样本总数一定时, 当 n 值为有限值时(即 K_n 越大), K_n 近邻估计十分粗糙, 当 n 值增大(即 K_n 越小), K_n 近邻估计的结果变得精细。

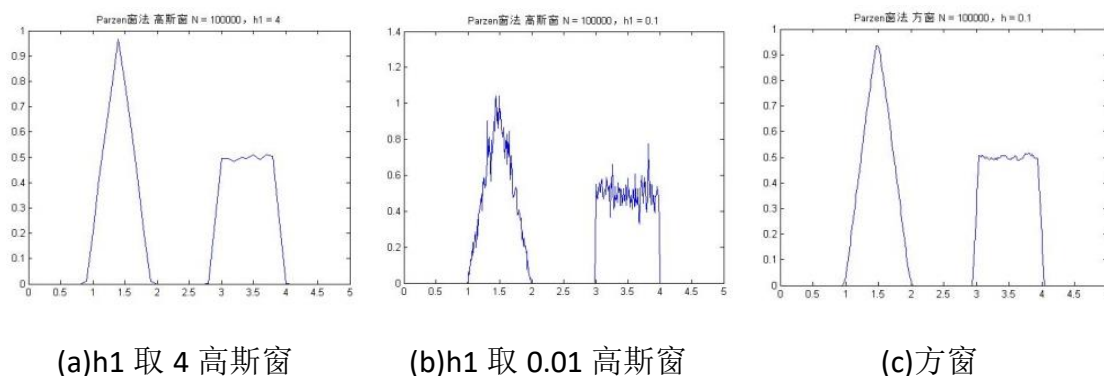


图 4-3-2 Parzen 窗法实验结果

如图 4-3-2 所示, 从方窗和正态高斯窗 2 个 Parzen 窗仿真实验可以看出, 估计的概率密度函数与窗函数的选择以及 h 的取值大小有密切的关系。

如图 4-3-2(a)所示, 当 $h=4$ 时, 使用高斯窗进行非参数估计, 曲线起伏很大, 估计较粗糙, 与真实的概率密度函数曲线相差较大。如图 4-3-2(b)所示, 当 $h=0.1$ 时, 估计较为精细, 不过噪声依旧明显。可以验证 Parzen 窗的实验原理, 随着参数 h 的增大, 估计曲线逐渐逼近实际的概率密度曲线。对比图 4-3-2 方窗与正态窗的两个 Parzen 窗参数估计, 可以看出, 在取同样的步长和参数时, 正态窗函数的估计结果要好于方窗。

5.实验代码

5.1 数据可视化代码

```
% *****实验数据 ex2data1.txt 可视化***** %

ex2data1 = load('ex2data1.txt');

%提取数据点的横、纵坐标，分别存入向量 ex2data1_x、ex2data1_y

ex2data1_x = ex2data1(1:size(ex2data1,1),1);

ex2data1_y = ex2data1(1:size(ex2data1,1),2);

%根据样本的实际类别将训练集分类

ex2data1_0 = zeros(size(ex2data1)); ex2data1_1 = zeros(size(ex2data1));

%将 label 为 0 的训练样本存入向量 ex2data1_0 中，将 label 为 1 的训练样本存入向量 ex2data1_1 中

for i = 1:size(ex2data1,1)

    if (ex2data1(i,3)==0)

        ex2data1_0(i,:) = ex2data1(i,:);

    else

        ex2data1_1(i,:) = ex2data1(i,:);

    end

end

ex2data1_0(all(ex2data1_0==0,2),:)=[]; ex2data1_1(all(ex2data1_1==0,2),:)=[]; %去掉向

量中没有数据的点

figure;

%画图，用红色+号表示 label 为 0 的数据点，用蓝色*号表示 label 为 1 的数据点

scatter(ex2data1_0(:,1),ex2data1_0(:,2),'r+');

hold on

scatter(ex2data1_1(:,1),ex2data1_1(:,2),'b*');

hold off

%在坐标区上添加图例

legend('y=0','y=1');
```

```

%添加横纵坐标

xlabel('Microchip Test x1');ylabel('Microchip Test x2');

%添加图注

title('实验数据 1');

```

5.2 贝叶斯分类器实现代码

```

%***导入实验数据***%

data = load('ex2data1.txt');

%***提取数据点的横、纵坐标***%

data_x = data(1:size(data,1),1); data_y = data(1:size(data,1),2);

%***根据样本的实际类别将数据集分类***%

data_0 = zeros(size(data)); data_1 = zeros(size(data));

for i = 1:size(data,1)

    if (data(i,3)==0)

        data_0(i,:) = data(i,:);

    else

        data_1(i,:) = data(i,:);

    end

end

data_0(all(data_0==0,2),:)=[]; data_1(all(data_1==0,2),:)=[];

%***所有 label 为 0 的数据存储在数组 data_0 中，每一行为一个样本***%

%***提取 data_0 的 70%作为训练样本 train_0***%

extra_0 = floor(0.7*size(data_0,1));

% size(数组, 1) 1 表示计算行数 2 表示计算列数

% 数组(提取起始行:提取终止行,提取起始列:提取终止列)

train_0 = data_0(1:extra_0,1:2); train_0_x = train_0(:,1); train_0_y = train_0(:,2);

%***提取 data_1 的 70%作为训练样本 train_1 ***%

extra_1 = floor(0.7*size(data_1,1));

train_1 = data_1(1:extra_1,1:2); train_1_x = train_1(:,1); train_1_y = train_1(:,2);

```

```

***剩余其他样本作为测试样本***%

test_0 = data_0((extra_0+1):(size(data_0,1)),:);

test_1 = data_1((extra_1+1):(size(data_1,1)),:);

test    = cat(1,test_0,test_1);    test_x = test(:,1);    test_y = test(:,2);

%        cat(dim,数组,数组) 1表示纵向串联数组

***计算先验概率***%

p_w_0 = size(data_0,1)/size(data,1);

p_w_1 = 1 - p_w_0;

***计算训练样本的均值和协方差矩阵***%

train_0_mean = mean(train_0);          train_1_mean = mean(train_1);

train_0_var = cov(train_0_x , train_0_y);    train_1_var = cov(train_1_x , train_1_y);

***创建向量 label_test 存储分类结果***%

label_test = zeros(size(test,1),1);

***判别函数***%

for i = 1:size(test,1)

    x = test(i,1:2);

    g0      =      (-1/2)*((x-train_0_mean))*inv(train_0_var)*((x-train_0_mean)')      -
(1/2)*(log(det(train_0_var))) + log(p_w_0);

    g1      =      (-1/2)*((x-train_1_mean))*inv(train_1_var)*((x-train_1_mean)')      -
(1/2)*(log(det(train_1_var))) + log(p_w_1);

    if (g0>g1)

        label_test(i,1) = 0;

    else

        label_test(i,1) = 1;

    end

end

*** 计算错误率 ***%

label_real = test(:,3);

***计算正确识别个数***%

```

```

t = 0;  f = 0;

for i=1:size(test,1)

    if (label_test(i,1) == label_real(i))

        t = t+1;

    else

        f = f+1;

    end

end

e = f/size(test,1)

***画出决策曲线***%

xmin = min(data_x); xmax = max(data_x);  ymin = min(data_y); ymax = max(data_y);

xvals = linspace(xmin,xmax,1000);          yvals = linspace(ymin,ymax,1000);

[gridx,gridy] = meshgrid(xvals, yvals);

***计算矩阵 Z 用于画图***%

Z = zeros(size(gridx));

for i = 1:size(xvals,2)

    for j = 1:size(yvals,2)

        x = [xvals(i),yvals(j)];

        g0      =      (-1/2)*((x-train_0_mean))*inv(train_0_var)*((x-train_0_mean)')      -
(1/2)*(log(det(train_0_var))) + log(p_w_0);

        g1      =      (-1/2)*((x-train_1_mean))*inv(train_1_var)*((x-train_1_mean)')      -
(1/2)*(log(det(train_1_var))) + log(p_w_1);

        if (g0>g1)

            Z(i,j) = 0;

        else

            Z(i,j) = 1;

        end

    end

end

end

```



```

figure;

scatter(data_0(:,1),data_0(:,2),'r+');

hold on

scatter(data_1(:,1),data_1(:,2),'b*');

hold on

contour(gridx,gridy,z, 'k', 'Linewidth',3); title('bayes 分类器实验')

```

5.3 logistic 回归实现代码

```

data = load('ex2data2.txt');

%***提取数据点的横、纵坐标***%

data_x1 = data(1:size(data,1),1);

data_x2 = data(1:size(data,1),2);

%***根据样本的实际类别将数据集分类***%

data_0 = zeros(size(data)); data_1 = zeros(size(data));

for i = 1:size(data,1)

    if (data(i,3)==0)

        data_0(i,:) = data(i,:);

    else

        data_1(i,:) = data(i,:);

    end

end

data_0(all(data_0==0,2),:)=[]; data_1(all(data_1==0,2),:)=[];

%***所有 label 为 0 的数据存储在数组 data_0 中，每一行为一个样本***%

%***所有 label 为 1 的数据存储在数组 data_1 中，每一行为一个样本***%

%***提取 data_0 的 70%作为训练样本 train_0***%

extra_num_0 = floor(0.7*size(data_0,1));

train_0 = data_0(1:extra_num_0,:); train_0_x1 = train_0(:,1); train_0_x2 =

train_0(:,2);

%***提取 data_1 的 70%作为训练样本 train_1 ***%

```

```

extra_num_1 = floor(0.7*size(data_1,1));

train_1 = data_1(1:extra_num_1,:);      train_1_x1 = train_1(:,1);   train_1_x2 =
train_1(:,2);

***剩余其他样本作为测试样本***%

test_0 = data_0((extra_num_0+1):(size(data_0,1)),:);
test_1 = data_1((extra_num_1+1):(size(data_1,1)),:);

test      = cat(1,test_0,test_1);      test_x1 = test(:,1);   test_x2 = test(:,2);
test_y = test(:,3);

***合并训练样本***%

train      = cat(1,train_0,train_1);   train_x1 = train(:,1);   train_x2 = train(:,2);
train_y = train(:,3);

***计算训练样本总个数***%

[train_num,train_c] = size(train);

ite  = 1500;   %迭代次数

alpha = 0.01;   %学习率

*** theta 为 n+1 维向量***%

theta = zeros(6,1);

***调整训练集 train, 扩展训练数据为 n+1 维, 存在新的数组 train_mat 中进行计算***%

train_mat = ones(size(train,1),6);

train_mat(:,2) = train_x1;

train_mat(:,3) = train_x2;

train_mat(:,4) = train_x1.*train_x1;

train_mat(:,5) = train_x2.*train_x2;

train_mat(:,6) = train_x1.*train_x2;

***计算更新方程及代价函数***%

***创建存储代价函数的向量以观察收敛过程（损失下降过程）***%

cost = zeros(ite,1);

```

```

for i = 1:ite

    %***sigmoid function***%

    h_theta_x = 1./(exp(-(train_mat*theta)) +1 );

    %***使用梯度下降法更新模型系数***%

    theta(1,1) = theta(1,1) - (alpha) * sum( h_theta_x - train_y );

    theta(2,1) = theta(2,1) - (alpha) * sum( (h_theta_x - train_y).*train_mat(:,2) );

    theta(3,1) = theta(3,1) - (alpha) * sum( (h_theta_x - train_y).*train_mat(:,3) );

    theta(4,1) = theta(4,1) - (alpha) * sum( (h_theta_x - train_y).*train_mat(:,4) );

    theta(5,1) = theta(5,1) - (alpha) * sum( (h_theta_x - train_y).*train_mat(:,5) );

    theta(6,1) = theta(6,1) - (alpha) * sum( (h_theta_x - train_y).*train_mat(:,6) );

    %***计算 cost***%

    cost(i,1) = (1/2/train_num) * sum((h_theta_x - train_y).*(h_theta_x - train_y));

    %    cost(i,1) =

end

%***使用训练集测试模型参数***%

label_test_logistic = zeros(size(test,1),1);

%***计算正确识别个数***%

t = 0;    f = 0;

for i=1:size(test,1)

    h_t = 1/(1+exp(-([1,test(i,1),test(i,2),test(i,1)^2,test(i,2)^2,test(i,1)*test(i,2)]*theta))) );

    if h_t>=0.5

        label_test_logistic(i,1) = 1;

    else

        label_test_logistic(i,1) = 0;

    end

    if (label_test_logistic(i,1) == test_y(i))

        t = t+1;

```

```

else

    f = f+1;

end

end

e = f/size(test,1)

***画出决策曲线***%

x1min = min(data_x1); x1max = max(data_x1);
x2min = min(data_x2); x2max = max(data_x2);
x1vals = linspace(x1min,x1max,100);
x2vals = linspace(x2min,x2max,100);
[gridx1,gridx2] = meshgrid(x1vals, x2vals);
Z = zeros(size(gridx1));
for i = 1:size(x1vals,2)
    for j = 1:size(x2vals,2)
        x = [x1vals(i),x2vals(j)];
        h_t_plot = 1/(1+
exp(-([1,x1vals(i),x2vals(j),x1vals(i)^2,x2vals(j)^2,x1vals(i)*x2vals(j)]*theta)) ));
        if h_t_plot>=0.5
            Z(i,j) = 1;
        else
            Z(i,j) = 0;
        end
    end
    % Z(i,j) = (1+ exp(-([1,x1vals(i),x2vals(j)]*theta)) )^(-1);
end
end

figure;
scatter(data_0(:,1),data_0(:,2),'r+');
hold on

```

```

scatter(data_1(:,1),data_1(:,2),'b*');

hold on

legend('y=0','y=1');

contour(gridx1,gridx2,Z, 'k', 'Linewidth',1); title(' ');

***观察损失下降过程***

figure;

plot(cost); title('损失函数');

```

5.4SVM 实验代码

```

data = load('ex2data1.txt');

***提取数据点的横、纵坐标***

data_x = data(1:size(data,1),1);

data_y = data(1:size(data,1),2);

***根据样本的实际类别将数据集分类***

data_0 = zeros(size(data)); data_1 = zeros(size(data));

for i = 1:size(data,1)

    if (data(i,3)==0)

        data_0(i,:) = data(i,:);

    else

        data_1(i,:) = data(i,:);

    end

end

data_0(all(data_0==0,2),:)=[]; data_1(all(data_1==0,2),:)=[];

***所有 label 为 0 的数据存储在数组 data_0 中，每一行为一个样本***

***提取 data_0 的 70%作为训练样本 train_0***

extra_0 = floor(0.7*size(data_0,1)); train_0 = data_0(1:extra_0,:);

***提取 data_1 的 70%作为训练样本 train_1 ***

extra_1 = floor(0.7*size(data_1,1)); train_1 = data_1(1:extra_1,:);

```

```

***整个训练集***%

train = cat(1,train_0,train_1);

***剩余其他样本作为测试样本***%

test_0 = data_0((extra_0+1):(size(data_0,1)),:);
test_1 = data_1((extra_1+1):(size(data_1,1)),:);
test = cat(1,test_0,test_1); test_num = size(test,1);

***使用线性核***%

***训练分类器并画出决策曲线***%

class = svmtrain(train(:,1:2),train(:,3),'kernel_function','linear','showplot',true);

title('线性核函数');

***测试分类器并计算错误率***%

test_label = svmclassify(class,test(:,1:2));

e = abs(sum(test_label - test(:,3)))/test_num

***使用 sigmoid***%

***训练分类器并画出决策曲线***%

class = svmtrain(train(:,1:2),train(:,3),'kernel_function','mlp','showplot',true);

title('多层感知器核函数');

***测试分类器并计算错误率***%

test_label = svmclassify(class,test(:,1:2));

e = abs(sum(test_label - test(:,3)))/test_num

***使用 3 次多项式核函数***%

***训练分类器并画出决策曲线***%

class =
svmtrain(train(:,1:2),train(:,3),'kernel_function','polynomial','polyorder','polynomial
','showplot',true); title('3 次多项式核函数');

***测试分类器并计算错误率***%

test_label = svmclassify(class,test(:,1:2));

e = abs(sum(test_label - test(:,3)))/test_num

***使用 6 次多项式核函数***%

```

```

***训练分类器并画出决策曲线***%

class
=

svmtrain(train(:,1:2),train(:,3),'kernel_function','polynomial','polyorder',5,'showplot',true); title('5 次多项式核函数');

***测试分类器并计算错误率***%

test_label = svmclassify(class,test(:,1:2));

e = abs(sum(test_label - test(:,3)))/test_num

***使用径向基核函数***%

***训练分类器并画出决策曲线***%

class
=

svmtrain(train(:,1:2),train(:,3),'kernel_function','rbf','rbf_sigma',1,'showplot',true);

title('径向基核函数 sigma=1');

***测试分类器并计算错误率***%

test_label = svmclassify(class,test(:,1:2));

e = abs(sum(test_label - test(:,3)))/test_num

***训练分类器并画出决策曲线***%

class
=

svmtrain(train(:,1:2),train(:,3),'kernel_function','rbf','rbf_sigma',0.5,'showplot',true); title('径向基核函数 sigma=0.5');

***测试分类器并计算错误率***%

test_label = svmclassify(class,test(:,1:2));

e = abs(sum(test_label - test(:,3)))/test_num

***使用径向基核函数***%

***训练分类器并画出决策曲线***%

class
=

svmtrain(train(:,1:2),train(:,3),'kernel_function','rbf','rbf_sigma',0.1,'showplot',true); title('径向基核函数 sigma=0.1');

***测试分类器并计算错误率***%

test_label = svmclassify(class,test(:,1:2));

```

```

e = abs(sum(test_label - test(:,3)))/test_num

***使用径向基核函数,添加惩罚因子***%

***训练分类器并画出决策曲线***%

class
=

svmtrain(train(:,1:2),train(:,3),'kernel_function','rbf','rbf_sigma',0.5,'boxconstraint',10,'showplot',true); title('径向基核函数 sigma=0.5 惩罚因子 c=10');

***测试分类器并计算错误率***%

test_label = svmclassify(class,test(:,1:2));

e = abs(sum(test_label - test(:,3)))/test_num

***使用径向基核函数,添加惩罚因子***%

***训练分类器并画出决策曲线***%

class
=

svmtrain(train(:,1:2),train(:,3),'kernel_function','rbf','rbf_sigma',0.5,'boxconstraint',100,'showplot',true); title('径向基核函数 sigma=0.5 惩罚因子 c=100');

***测试分类器并计算错误率***%

test_label = svmclassify(class,test(:,1:2));

e = abs(sum(test_label - test(:,3)))/test_num

```

5.5BP 神经网络分类器实验代码

```

data = load('ex2data2.txt');

***提取数据点的横、纵坐标***%

data_x = data(1:size(data,1),1);

data_y = data(1:size(data,1),2);

***根据样本的实际类别将数据集分类***%

data_0 = zeros(size(data)); data_1 = zeros(size(data));

for i = 1:size(data,1)

    if (data(i,3)==0)

        data_0(i,:) = data(i,:);

    else

```



```

        data_1(i,:) = data(i,:);

    end

end

data_0(all(data_0==0,2),:)=[];    data_1(all(data_1==0,2),:)=[];

***所有 label 为 0 的数据存储在数组 data_0 中，每一行为一个样本***%

***提取 data_0 的 70%作为训练样本 train_0***%

extra_0 = floor(0.7*size(data_0,1));    train_0 = data_0(1:extra_0,:);

***提取 data_1 的 70%作为训练样本 train_1 ***%

extra_1 = floor(0.7*size(data_1,1));    train_1 = data_1(1:extra_1,:);

***剩余其他样本作为测试样本***%

test_0 = data_0((extra_0+1):(size(data_0,1)),:);

test_1 = data_1((extra_1+1):(size(data_1,1)),:);

in = cat(1,train_0,train_1);    train_in = in(:,1:2)';    train_out = in(:,3)';

test = cat(1,test_0,test_1);    test_in = test(:,1:2)';    test_out = test(:,3)';

test_num = size(test,1);

%输入数据归一化处理

[inputn,inputps]=mapminmax(train_in);

%创建网络

net=newff(inputn,train_out,[10,5],{'tansig','purelin'});

%设置训练次数

net.trainParam.epochs = 1500;

%设置收敛误差

net.trainParam.goal=0.001;

% 学习速率

net.trainParam.lr=0.01;

%训练网络

net=train(net,inputn,train_out);

%测试数据归一化

inputn_test=mapminmax('apply',test_in,inputps);

```

```

%net 预测标签输出

y=sim(net,inputn_test);%y=sim(net,x);net 表示已训练好的网络，x 表示输入数据，y 表示网络预测数据。

%根据网络输出获得预测类别的种类

y(find(y<0.5))=0;

y(find(y>=0.5))=1;

*** 计算错误率 ***%

t = 0;  f = 0;

for i=1:test_num

    if (y(i) == test_out(i))

        t = t+1;

    else

        f = f+1;

    end

end

e = f/test_num

% ***画出决策曲线***%

% xmin = min(data_x); xmax = max(data_x);

% ymin = min(data_y); ymax = max(data_y);

% xval = linspace(xmin,xmax,100);

% yval = linspace(ymin,ymax,100);

% xvals = mapminmax('apply',xval,inputps);

% yvals = mapminmax('apply',yval,inputps);

% [gridX,gridY] = meshgrid(xval, yval);

% Z = zeros(size(gridX));

% for i = 1:size(xvals,2)

%     for j = 1:size(yvals,2)

%         x = [xvals(i),yvals(j)]';

%         tmp = sim(net,x);

%         if (tmp<0.5)

```

```

%          z(i,j) = 0;

%          else

%          z(i,j) = 1;

%          end

%      end

% end

% figure;

% scatter(data_0(:,1),data_0(:,2),'r+');

% hold on

% scatter(data_1(:,1),data_1(:,2),'b*');

% hold on

% legend('y=0','y=1');

% contour(gridX,gridY,Z, 'k', 'Linewidth',1);

```

5.6 非参数估计实验代码

```

clc;close all;clear all;

[data] = textread('ex2data.txt', ...

    '%f');

%*****画数据的直方图*****%

figure(1);

xbins = 0:0.1:5;  [data_plot_y,data_plot_x] = hist(data,xbins);

data_plot_y_1 = data_plot_y/(size(data,1));

plot(data_plot_x,data_plot_y_1);

%*****kn 近邻法*****%

%*****kn = 0.1*sqrt(样本总数)*****%

figure(3);

xbins = 0:0.01:5;  [data_plot_y_kn,data_plot_x] = hist(data,xbins);

kn = 0.1*sqrt((size(data,1)));

data_plot_t = 0;  t = 1;  data_plot_v = 0;  data_plot_k = data_plot_y;

```

```

for i = 1:size(data_plot_y_kn,2)

    data_plot_t = data_plot_t + data_plot_y_kn(1,i);

    if (data_plot_t > kn)

        for m = t : i

            data_plot_v = data_plot_v + data_plot_y_kn(1,m);

        end

        for n = t : i

            data_plot_kn(1,n) = data_plot_v/(size(data,1)*(i-t+1)*0.01);

        end

    else

        data_plot_kn(1,i) = 0;

    end

    data_plot_t = 0; data_plot_v = 0;

    t = i;

end

plot(data_plot_x,data_plot_kn);

title(['Kn 近邻法 N = ',num2str(size(data,1)),', Kn = ',num2str(kn)]);

%*****Parzen 窗法*****%

%*****核函数使用高斯窗函数*****%

figure(5);

xbins = 0:0.1:5; [data_plot_y_p,data_plot_x] = hist(data,xbins);

data_plot_y_parzen = zeros(size(data_plot_y_p));

h1 = 4;

var = h1/(sqrt(size(data,1)));

% kxx = zeros(size(data,1));

for i = 1:(size(data_plot_x,2)-1)

    for j = 1:size(data,1)

        kxx(j) =

(1/((sqrt(2*pi))*var))*exp(-( ( data(j,1)-(data_plot_x(i+1)) )^2)/(2*((var)^2)));

```

```

end

data_plot_y_parzen(i) = (1/size(data,1))*(sum(kxx(:)));

end

plot(data_plot_x,data_plot_y_parzen);

title(['Parzen 窗法 高斯窗 N = ',num2str(size(data,1)),' h1 = ',num2str(h1)]);

%*****核函数使用方窗*****%

figure(7);

h = 0.1;

xbins = 0:0.01:5; [data_plot_y_p,data_plot_x] = hist(data,xbins);

data_plot_y_parzen = zeros(size(data_plot_y_p));

for i = 1:(size(data_plot_x,2)-1)

    for j = 1:size(data,1)

        if abs(data(j,1)-(data_plot_x(i+1)))<h/2

            kxx(j) = 1/h;

        else

            kxx(j) = 0;

        end

    end

end

data_plot_y_parzen(i) = (1/size(data,1))*(sum(kxx(:)));

end

plot(data_plot_x,data_plot_y_parzen);

title(['Parzen 窗法 方窗 N = ',num2str(size(data,1)),' h = ',num2str(h)]);

```

目 录

1.实验内容.....	1
1.1 实验内容	1
1.2 实验平台	1
1.3 实验数据	1
2.代码实现.....	3
2.1 贝叶斯分类器	3
2.1.1 贝叶斯分类器实验原理	3
2.1.2 算法流程与代码实现	4
2.1.3 实验结果与分析	6
2.2 logistic 回归	6
2.2.1logistics 回归实验原理.....	6
2.2.2 算法流程与代码实现	8
2.2.3 实验结果与分析	9
2.3 SVM	12
2.3.1SVM 支持向量机原理	12
2.3.2 参数选择与代码实现	14
2.3.3 实验结果与分析	15
2.4 神经网络分类器	18
2.4.1BP 神经网络原理	18
2.4.2 参数选择与代码实现	19
2.4.3 实验结果与分析	20
3.结课分析.....	23
4.参数优化——非参数估计	26
5.实验代码.....	29
5.1 数据可视化代码	29
5.2 贝叶斯分类器实现代码	30
5.3logistic 回归实现代码	33
5.4SVM 实验代码	37
5.5BP 神经网络分类器实验代码	40
5.6 非参数估计实验代码	43