華東理工大學

模式识别大作业

题　　目　Otto Group 商品识别

学　　院　信息科学与工程学院

专　　业　控制科学与工程

组　　员　徐逸峰、莫泽坤、徐泽铭、朱雪婷、张鑫婷

指导教师　赵海涛

完成日期： 2018 年 12 月 15 日

目录

Otto Group 商品识别

从本质上来讲，识别问题是对待识别的物体进行特征分类，大自然中存在的任何一类物体，即使是人造的，都有其区别于其他物体的一个或多个特征，而识别问题就是突出这些能够使一类物体区别于其他物体的特征，将其表述为数学形式，通过一些算法将这些特征划分开来，从而达到将物体合理分类，即识别某类物体的目的。

本次实验将针对识别问题，选定一个具体的问题——Otto Group 商品识别问题来探究物体识别的具体方法，并采用不同的方法解决该问题，且对不同方法进行比较，提出思考。

1 问题描述

Otto Group 是世界上最大的电子商务公司之一，在全世界范围内，它每天会卖出数百万件商品。这些商品总体来说可以分为 9 类，即每件商品所属的类别分别是 Class_1～ Class_9。对于这家公司来说，货物供给和需求分析是非常重要的信息，因此，获取每类商品的售出情况信息有助于该公司优化自己的销售策略。现假设某商品的所属类别是未知的，仅知道一些商品的多个特征，故本问题的任务就是设计一个算法模型来判断一个商品所属的类别。

很明显，本问题是一个典型的多分类问题，且已提供每个商品的多项特征和类别，本问题中，提供了 49502 个商品，且在这些商品中，共有 93 个特征，每类商品都有几个特定的特征。本问题需要先使用提供的商品特征和类别进行模型训练，然后使用训练好的模型预测商品所属的类别，共需要预测 12376 个商品所属的类别。

1.1 BP算法原理

BP 神经网络因其学习算法采用反向误差传播算法（BP 算法）而得名。BP 算法的学习过程由信息正向传播和误差反向传播两部分组成。在正向传播过程中，计算各层神经元的状态。输入信息从输入层经各隐含层逐层处理，并传向输出层，每层神经元（节点）的状态只影响下一层节点的状态。如果输出层的状态与期望的输出不一致，则转入反向传播过程，将误差信号沿原来的连接通路返回，同时修改各层神经元的权值，权值的不断调整会使网络误差越来越小。神经网络一般形式如图 1-1 所示，参数如表 1-1 所示。

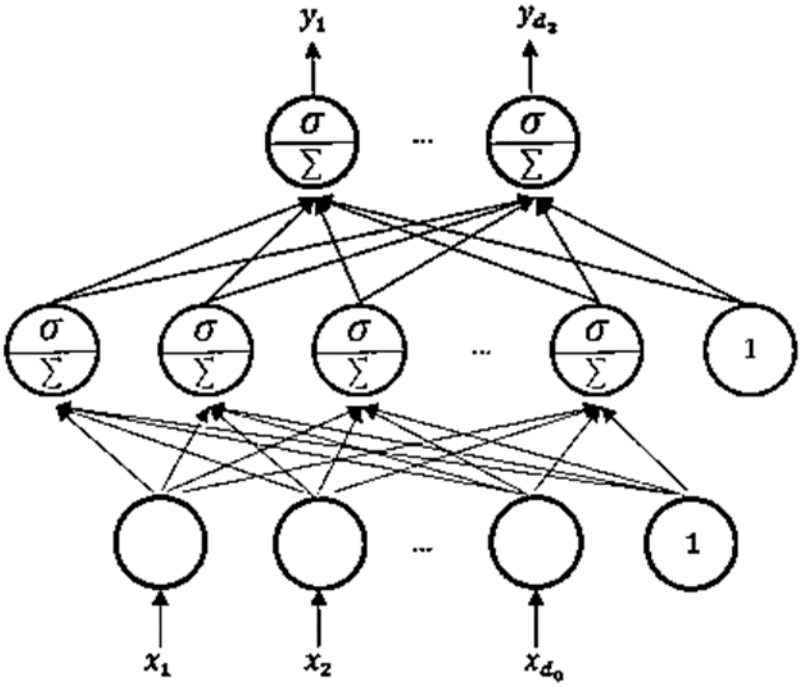

图 1-1 三层神经神经网络结构示意

表 1-1 神经网络参数

层名	符号	约定
输入层（d_0个神经元）	$x_1, x_2, \dots x_{d_0}$	输入数据
中间层（d_1个神经元）	$w_1^{(1)}, w_2^{(1)}, \dots w_{d_1}^{(1)}$	权值
	$b_1^{(1)}, b_2^{(1)}, \dots b_{d_1}^{(1)}$	阈值
	$z_1^{(1)}, z_2^{(1)}, \dots z_{d_1}^{(1)}$	求和输出
	$h_1, h_2, \dots h_{d_1}$	激活函数变换输出
输出层（d_2个神经元）	$w_1^{(2)}, w_2^{(2)}, \dots w_{d_2}^{(2)}$	权值
	$b_1^{(2)}, b_2^{(2)}, \dots b_{d_2}^{(2)}$	阈值
	$z_1^{(2)}, z_2^{(2)}, \dots z_{d_2}^{(2)}$	求和输出
	$y_1, y_2, \dots y_{d_2}$	激活函数变换输出
	$t_1, t_2, \dots t_{d_2}$	真实值

在正向传播过程中，隐含层或输出层任一神经元将来自前一神经元传来的信息进行加权整合，通常还会在整合过的信息中添加一个阈值，这主要是模仿生物学中神经元必须达到一定的阈值才会触发的原理。再经激活函数变换后传到下一

层神经元。

（1）从输入层到中间层

$$z_i^{(1)} = \boldsymbol{w_i^{(1)}}\boldsymbol{x} = \sum_{j=1}^{d_0} w_{ij}^{(1)} x_j + b_i^{(1)}$$

$$h_i = \sigma\left(z_i^{(1)}\right) = \sigma\left(\sum_{j=1}^{d_0} w_{ij}^{(1)} x_j + b_i^{(1)}\right)$$

其中：$i = 1,2 \ldots d_1$，$\boldsymbol{x} = [x_1, x_2, \ldots, x_{d_0}]^T$，$\boldsymbol{w_i^{(1)}} = [w_{i1}^{(1)}, w_{i2}^{(1)}, \ldots, w_{id_0}^{(1)}]$

（2）从中间层到输出层

$$z_i^{(2)} = \boldsymbol{w_i^{(2)}}\boldsymbol{h} = \sum_{j=1}^{d_1} w_{ij}^{(2)} h_j + b_i^{(2)}$$

$$y_i = \sigma\left(z_i^{(2)}\right) = \sigma\left(\sum_{j=1}^{d_1} w_{ij}^{(2)} h_j + b_i^{(2)}\right)$$

其中：$i = 1,2 \ldots d_2$，$\boldsymbol{h} = [h_1, h_2, \ldots, h_{d_1}]^T$，$\boldsymbol{w_i^{(2)}} = [w_{i1}^{(2)}, w_{i2}^{(2)}, \ldots, w_{id_0}^{(2)}]$

在反向传播过程中，构造关于输出值与真实值的目标函数，按照减少输出值与真实值之间误差的方向，依托梯度下降法，从输出层反向经过各中间层回到输入层，逐步修正各连接权值与阈值。

构造目标函数（目标函数应为多个样本的和，下面为简化的方式）：

$$J = \frac{1}{2}\sum_{i=1}^{d_2}(t_i - y_i)^2$$

（1）从输出层到中间层

$$\frac{\partial J}{\partial z_i^{(2)}} = \frac{\partial J}{\partial y_i}\frac{\partial y_i}{\partial z_i^{(2)}} = -(t_i - y_i)\sigma'(z_i^{(2)})$$

其中：$i = 1,2, \ldots d_2$，为下面书写方便，令：$\delta_i^{(2)} = -(t_i - y_i)\sigma'\left(z_i^{(2)}\right)$

$$\frac{\partial J}{\partial w_{ij}^{(2)}} = \frac{\partial J}{\partial z_i^{(2)}}\frac{\partial z_i^{(2)}}{\partial w_{ij}^{(2)}} = \delta_i^{(2)} h_j$$

$$w_{ij}^{(2)} \leftarrow w_{ij}^{(2)} - \alpha\delta_i^{(2)} h_j$$

$$\frac{\partial J}{\partial b_i^{(2)}} = \frac{\partial J}{\partial z_i^{(2)}}\frac{\partial z_i^{(2)}}{\partial b_i^{(2)}} = \delta_i^{(2)}$$

$$b_i^{(2)} \leftarrow b_i^{(2)} - \alpha\delta_i^{(2)}$$

其中：$i = 1,2,\dots d_2$，$j = 1,2,\dots d_1$

（2）从中间层到输入层

因为$z_i^{(1)} z_k^{(2)}(\ i = 1,2,\dots d_2)$ 都有关系，因此求导时要对每个$z_k^{(2)}$进行。

$$\frac{\partial J}{\partial z_i^{(1)}} = \sum_{k=1}^{d_2} \frac{\partial J}{\partial z_k^{(2)}} \frac{\partial z_k^{(2)}}{\partial z_i^{(2)}} = \sum_{k=1}^{d_2} \frac{\partial J}{\partial z_k^{(2)}} \frac{\partial z_k^{(2)}}{\partial h_i} \frac{\partial h_i}{\partial z_i^{(1)}} = \sum_{k=1}^{d_2} \delta_k^{(2)} w_{k_i}^{(2)} \sigma'(z_i^{(1)})$$

其中：$i = 1,2,\dots d_1$，为下面书写方便，令：$\delta_i^{(1)} = \sum_{k=1}^{d_2} \delta_k^{(2)} w_{k_i}^{(2)} \sigma'(z_i^{(1)})$

$$\frac{\partial J}{\partial w_{ij}^{(1)}} = \frac{\partial J}{\partial z_i^{(1)}} \frac{\partial z_i^{(1)}}{\partial w_{ij}^{(1)}} = \delta_i^{(1)} x_j$$

$$w_{ij}^{(1)} \leftarrow w_{ij}^{(1)} - \alpha \delta_i^{(1)} x_j$$

$$\frac{\partial J}{\partial b_i^{(1)}} = \frac{\partial J}{\partial z_i^{(1)}} \frac{\partial z_i^{(1)}}{\partial b_i^{(1)}} = \delta_i^{(1)}$$

$$b_i^{(1)} \leftarrow b_i^{(1)} - \alpha \delta_i^{(1)}$$

其中：$i = 1,2,\dots d_1$，$j = 1,2,\dots d_0$

1.2 SVM算法原理

对于分类问题，对于某个维度内的数据，分类的超平面可以表示为

$$g(\boldsymbol{x}) = \boldsymbol{\omega}^T \boldsymbol{x} + \boldsymbol{b} = 0$$

分类问题划分为线性可分和线性不可分。对于线性可分的情况，对数据做作如下处理：

假定对于一个点x，令其垂直投影到超平面上的对应点为x_0，ω是垂直于超平面的一个向量，分隔面的法向量为

$$\omega_0 = \frac{\omega}{\|\omega\|}\gamma$$

γ为样本x到超平面的距离，根据几何知识，可得

$$x = x_0 + \gamma \frac{\omega}{\|\omega\|}$$

又因为x_0是超平面上一点，代入超平面方程可得

$$\omega^T x_0 + b = 0$$

在式$x = x_0 + \gamma \frac{\omega}{\|\omega\|}$左右同时乘以$\omega^T$，得

$$\omega^T x = \omega^T x_0 + \gamma \frac{\omega^T \omega}{\|\omega\|}$$

整理可得

$$\gamma = \frac{\omega^T x + b}{\|\omega\|} = \frac{|g(x)|}{\|\omega\|}$$

即样本到超平面的距离。

如图所示，即为要寻找的最优超平面（Optimal Hyper Plane），其到两条边界的距离相等，这个距离便是几何间隔$\tilde{\gamma} = y\gamma$，两条边界间隔边界之间的距离等于$2\tilde{\gamma}$，而间隔边界上的点就是支持向量，示意图如图 1-2 所示。

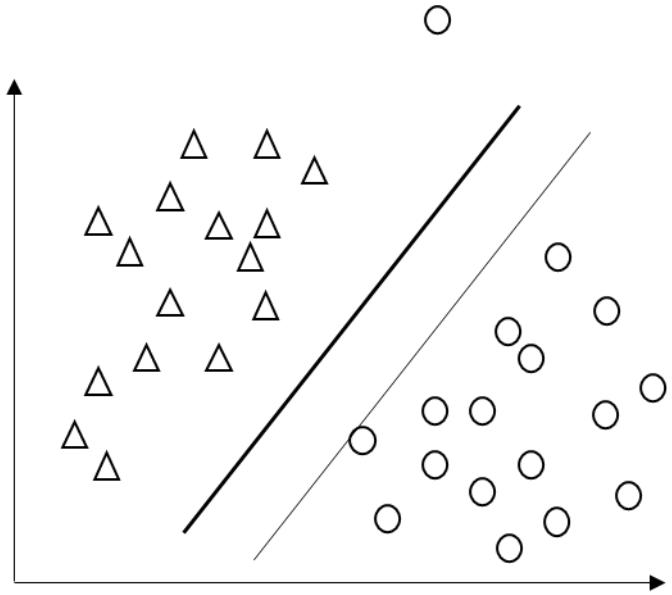

图 1-2 SVM 分类示意图

即目标函数为$\text{argmax}(\min(\text{y}(w^T\text{x} + \text{b})) * \frac{1}{\|w\|})$

注意到 w 和 b 等比例放大时，d 的结果是不会改变的。因此我们可以让所有支持向量的范数为 1，这样一来，目标函数及约束条件可以简化为：

$$\begin{cases} argmax(\dfrac{1}{\|w\|}) \\ s.t.\ y(w^T x + b) - 1 \geq 0 \end{cases}$$

为了便于之后的求导计算，将目标函数简化为$\min \frac{1}{2}\|w\|^2$

SVM 其实主要是在用对偶理论求解一个二次凸优化问题，通常使用拉格朗日乘子法求解。

令$\text{L}(\text{w}, \text{b}, \alpha) = \frac{1}{2} * \|w\|^2 - \sum_{i=1}^{n} \alpha_i (y_i(wx + b) - 1)$，对 w，b 求偏导并化简，

得到$\begin{cases} \underset{\alpha}{max} \left\{ \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{m} \sum_{j=1}^{m} \alpha_i \alpha_j y_i y_j x_i x_j \right\} \\ \text{s.t.} \quad \sum_{i=1}^{m} \alpha_i y_i = 0 \\ \alpha_i \geq 0 \quad i = 1,2,\cdots m \end{cases}$

求得最终结果为：

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

$$b^* = y_j - \sum_{i=1}^{N} \alpha_i^* y_i (x_i \cdot x_j)$$

其中α_i^*表示任选的一个支持向量

很多样本数据都不能够用一个超平面把数据完全分开。如果数据集中存在噪点的话，那么在求超平的时候就会出现很大问题。如果某些极少数数据点偏差过大落入对立类别数据域，计算支持向量就会小很多，分类效果就会变差。

因此引入一个松弛变量ξ来允许一些数据分割错误。这是新的目标函数变为：

$$min \frac{1}{2} * \|w\|^2 + C \sum_{i=1}^{n} \xi_j$$

其中 C 用于控制“最大化间隔”，根据实际问题进行调节。目标函数使用拉格朗日法进行化简计算后得到和原来一样的目标函数，但增加了$\alpha_i \leq C$的条件。

当然这是线性可分的情况，那么如果问题本身是线性不可分的情况呢，那就是先扩维后再计算，计算形式是一样的，示意图如图 1-3 所示：

$$\max_{\alpha} \left\{ \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{m} \sum_{j=1}^{m} \alpha_i \alpha_j y_i y_j \Phi(x_i) \Phi(x_j) \right\}$$

$$\text{s.t.} \quad \sum_{i=1}^{m} \alpha_i y_i = 0$$

$$\alpha_i \geq 0 \qquad i = 1,2,\cdots m$$

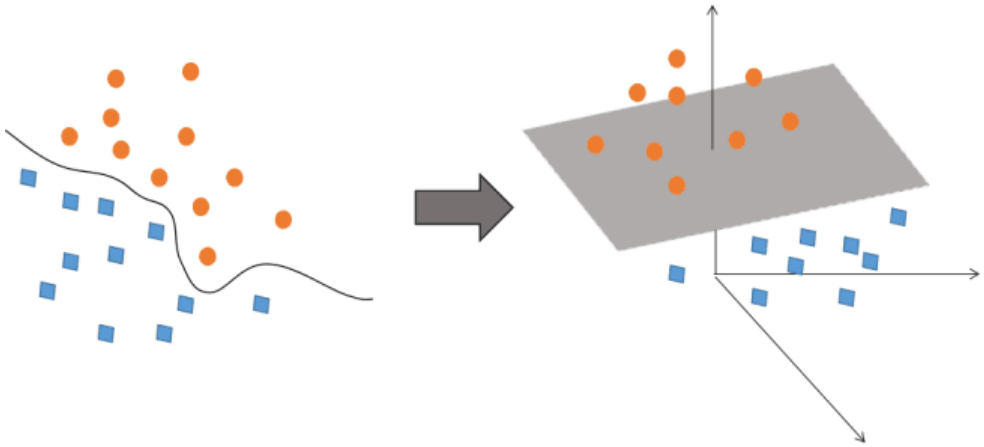

图 1-3 SVM 线性不可分情况下的扩维

其中$\Phi(x_i)$表示原来的样本扩维后的坐标。

从扩维前后的所求最大值问题的数学描述中可以明显看出，不管是不扩维的求解还是扩维的求解，在求解对偶问题的过程中都会用到各样本点的内积的结果，那么这时候问题来了，在很多情况下，扩维可能会吧原数据扩到很高维(甚至无

穷维)，这时候直接求内积是非常困难的，我们为了避免做这样的事就提出了核函数的概念。

核函数就是任意两个样本点在扩维后的空间的内积，如果等于这两个样本点在原来空间经过一个函数后的输出，那么这个函数就叫核函数。我们的目的就在于能够找到一个函数能够表达两个向量在映射后空间中的内积。

核函数种类很多，相同种类但是不同参数的核函数效果又有所不同，需要说明的是并不是说所有的核函数都能显示的写出隐含的从低维到高维的扩维细节。成为核函数有满足的条件 Mercer's condition。一般用得比较多比较成熟的核函数有线性核函数、多项式核函数、高斯核函数、拉普拉斯核函数和 sigmod 核函数等。

本次实验中使用线性核函数和高斯核函数两种核函数：

线性核函数如下所示：

$$\mathrm{k}(x_1, x_2) = \langle x_1, x_2 \rangle$$

高斯核函数如下所示：

$$\mathrm{k}(x_1, x_2) = \exp(-\frac{\|x_1 - x_2\|^2}{2\sigma^2})$$

1.2.1 SVM 直接法

直接法直接在目标函数上进行修改，将多个分类面的参数求解合并到一个最优化问题中，通过求解该最优化问题"一次性"实现多类分类。这种方法看似简单，但是在最优化问题求解过程中的变量过多，训练速度慢，而且在分类精度上也不占优势，当训练样本数非常大时，这一问题更加突出。其计算复杂度比较高，实现起来比较困难，只适合用于小型问题中。

1.2.2 SVM 间接法

间接法主要是通过组合多个二分类器来实现多分类器的构造，常见的方法有 one-versus-one 和 one-versus-rest 两种。

（1）"一对多"（one-versus-rest）

"一对多"SVM 思想是将多个类别转化成进行多次两个类别的分类来实现。在训练时，对于k个类别的样本数据，需要训练k个 SVM 二类分类器，第i个 SVM 子分类的样本数据标记为正类，其他不属于i类别的样本数据标记为负类。测试时，对测试数据分别计算各判别函数值，如果只有一个分类器输出正值，那么可直接判决结果为相应分类器编号，否则选取判别函数值最大所对应的类别为测试数据的类别。

这种方法的优点在于训练k个分类器，个数较少，其分类速度相对较快。但也有不足之处：（1）可能出现分类重叠现象以及不可分类现象。即所有分类器同时输出正值或同时输出负值。分类重叠可以根据到超平面的距离的远近来分类，

不可分类比较难解决，只能额外多划分出来一类。（2）当有新的类别加进来时，需要对所有的模型进行重新训练。（3）每个分类器的训练都是将全部的样本作为训练样本，这样在求解二次规划问题时，训练速度会随着训练样本的数量的增加而急剧减慢；同时由于负类样本的数据要远远大于正类样本的数据，从而出现样本不对称的情况，且这种情况随着训练数据的增加而趋向严重。

该分类将原有的分类问题同样分解成了一系列的两类分类问题，其中两个子类间的分类函数采用 SVM。如图 1-4 表示：

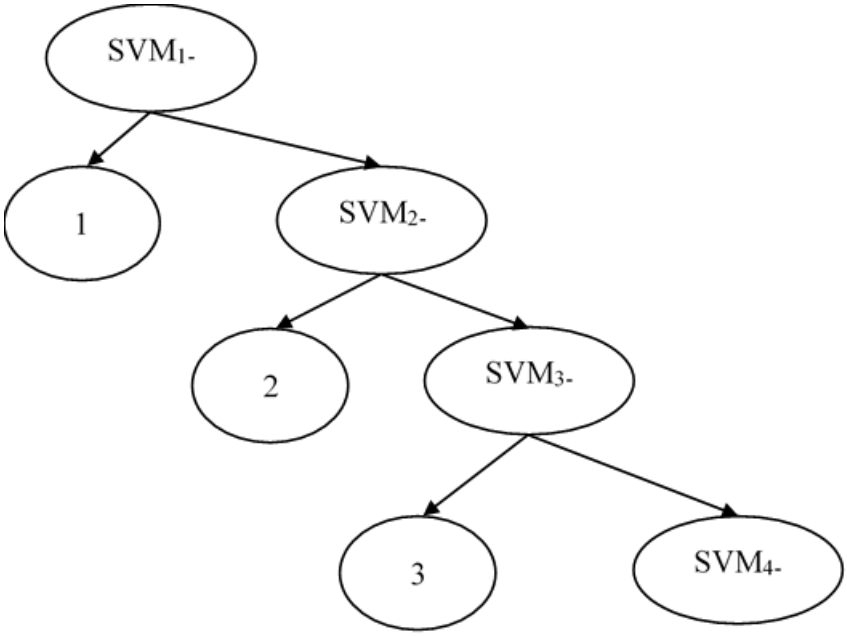

图 1-4 一对多 SVM 示意

（2）“一对一”（one-versus-one）

“一对一”SVM 分别选取两个不同类别构成一个 SVM 子分类器，思想是在分类过程中，两个类别依次进行判别分类。实现这样对于k个类别来说，共有$(k*(k-1)/2)$个分类器。在构造i和j的分类器时，可以将类别i的训练样本置为 1，j的样本置为-1 来进行训练。在进行测试的时候，使用最多的就是 Friedman 提出的投票策略：将测试数据x对所有的分类器分别进行测试，若由

$$y = sgn\left(W_{ij}^{T}\emptyset(x) + b_{ij}\right)$$

得到x属于第i类，则第i类加 1，属于j类，则第j类投票加 1。累计各类别的得分，选择得分最高者所对应的类别为测试数据的类别。

这种方式的优点是：增加测试样本，不需要重新训练所有的 SVM，只需要重新训练和增加训练样本相关的分类器，且在训练单个模型时，速度相对较快。缺点在于所需构造和测试的二类分类器的数量关于k呈二次函数增长，总训练时间和测试时间相对较慢。

从“一对一”的方式出发，出现了有向无环图（Directed Acyclic Graph）的分

类方法，训练过程和“一对一”方法类似，分类器的组成方式如图 1-5 所示：

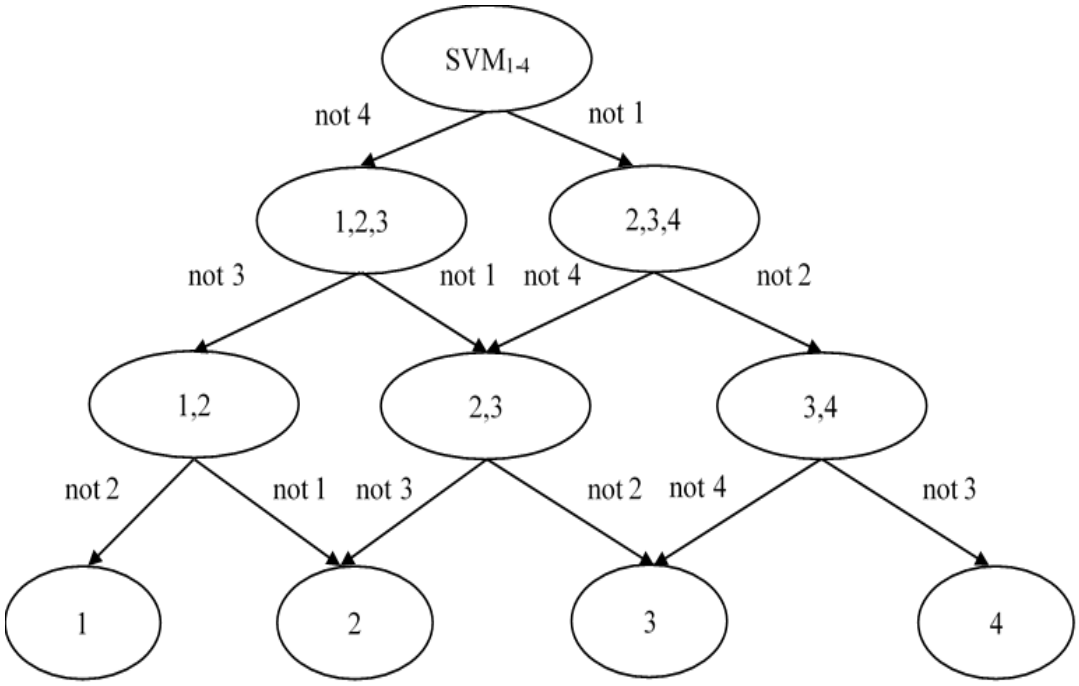

图 1-5 一对一 SVM 示意

这种方法的优势在于当类别数是k，只调用了$k-1$个分类器，减小了测试的数据量，提高了分类速度，且没有分类重叠和不可分类现象，但也会发生无法补救的错误，出现错误向下累积的现象。如果一开始的分类器回答错误，即初始分类错误，那么后面的分类器是无论如何也无法纠正它的错误的。但 DAG 的误差上限有理论证明，不会无限增加。而一对多和一对一方法中，尽管每一个两类分类器的泛化误差限是知道的，但是合起来进行多类分类的时候，误差上界还没有准确定论。

2 实验设计方案

本题使用 multi-class logarithmic loss 作为最后评判标准，公式如下：

$$\text{logloss} = -\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{M} y_{ij}\log(p_{ij})$$

其中 N 代表测试数据集中的商品数量，log 使用自然对数，y_{ij}表示商品 i 是否属于 j，如果是则y_{ij} =1，否则为 0。p_{ij}代表预测商品 i 属于$class_j$的概率。

2.1 BP算法设计方案

（1）初始化权值。把所有的权值和神经元节点的阈值设置成较小的随机数。

（2）正向计算网络实际输出。逐层计算中间层及输出层各节点的输出。

（3）反向递推更新参数。利用递推算法，先从输出层开始，然后从输出端向输入端调整各层权值。

（4）重复（2）-（3）过程，直到迭代次数足够或者误差达到规定指标以下。流程如图 1-6 所示。

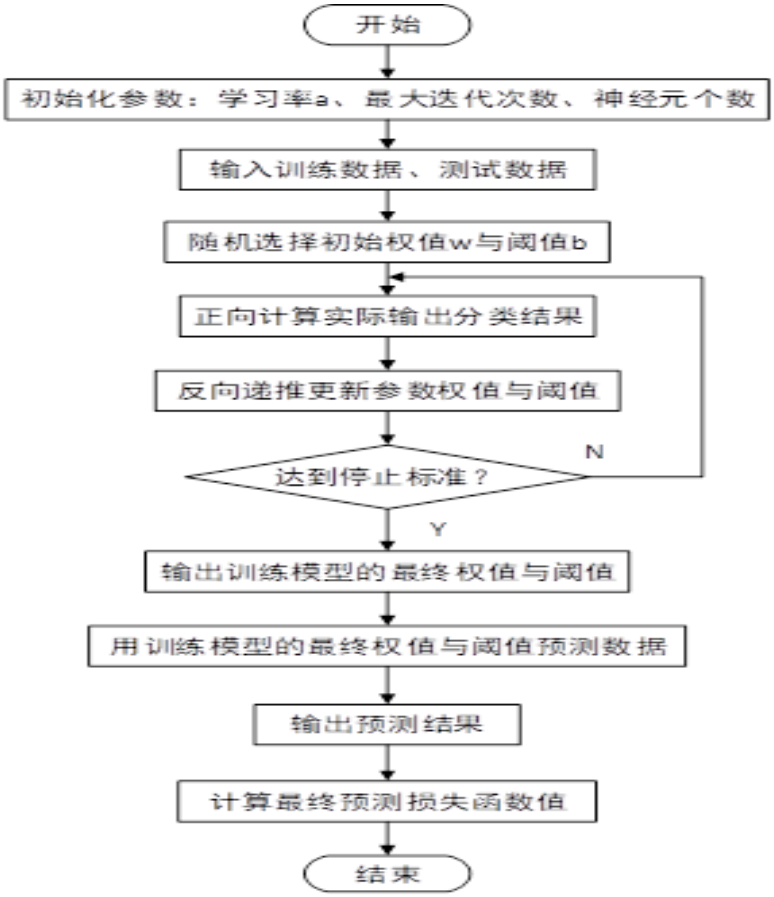

图 1-6 BP 算法方案流程图

2.2 SVM设计方案

2.2.1 one versus rest

（1）实验采用 5 折交叉验证：将所有训练数据分为 5 组，每次取一组作为验证集，4 组作为训练集。

（2）训练 SVM 模型并验证。

（a）对于当前类别，将 4 组训练数据中的第 i 类(当前类别)的 label 记为 1，其余类别的记为-1。

（b）采用二分类 SVM，使用 fitcsvm 函数训练区分 i 类与其他类，得到第 i 类验证集的当前交叉验证的 SVM 模型。

（c）使用 predict 函数预测第 i 组预测数据的分类结果，得到验证集的 predictlabel。

（d）对比 predictlabel 和 testlabel 比较预测分类结果与真实分类结果，计算正确率并保存。

（3）对于每个当前类别 i 重复步骤（2）5 次进行交叉验证。

（4）以上过程分出了训练数据的第 i 类数据，对每个类别都行进交叉验证(对类别进行循环)，找到对于每个类别而言的最高正确率对应的模型。

（5）输入需要预测的测试集数据，每个商品都用以上训练好的 9 个预测模型预测得到该商品被分到每一类的概率，取最大概率作为该商品分类结果。

（6）重复步骤（5），预测所有商品，输出分类结果。

（7）提交结果。

具体流程如图 1-7 所示。

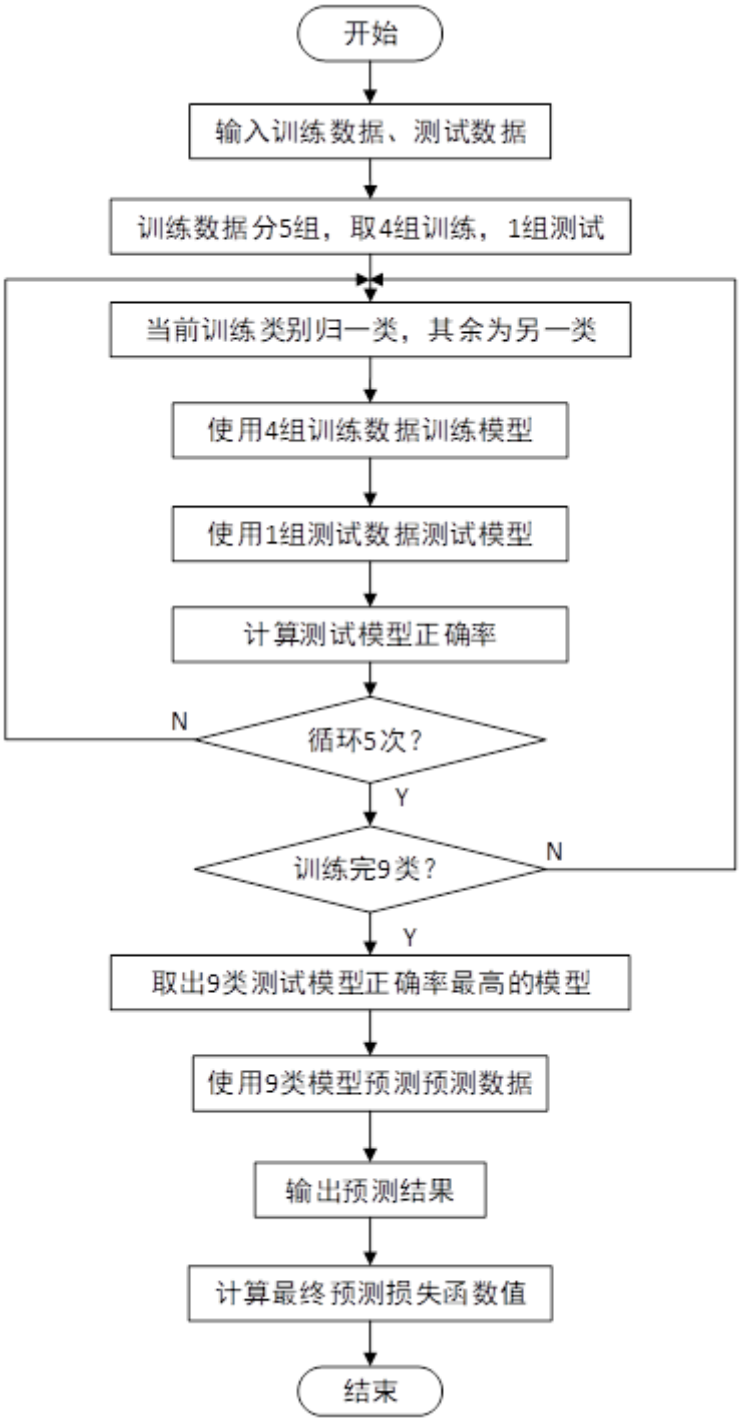

图 1-7 one vs rest 流程图

2.2.2 one versus one

one-versus-one 的主要思想为：对于 9 类数据，每次取 2 类做基本 SVM 二分类，可取的两类有：1 类与 2 类、1 类与 3 类、…、8 类与 9 类共 36 种组合，这 36 个组合可以构造 36 个二分类模型，这 36 个模型中有 4 个模型会包含一种类别。用这 36 个二分类模型预测商品所属类别，某商品在这 36 个模型中属于某类别的概率和最大，则该商品属于该类别。

（1）取两类数据，进行 SVM 二分类。

（2）重复步骤（1），获取 36 个二分类模型。

（3）输入需要预测的数据，对每个商品，分别用 36 个模型进行预测，计算该商品被分到每一类的概率和。

（4）取该商品被分类的最大概率和，将该商品分到该类。

（5）重复步骤（3）～（4），直至所有预测商品分类完，输出预测结果。

（6）计算损失函数值。

具体流程如图 1-8 所示。

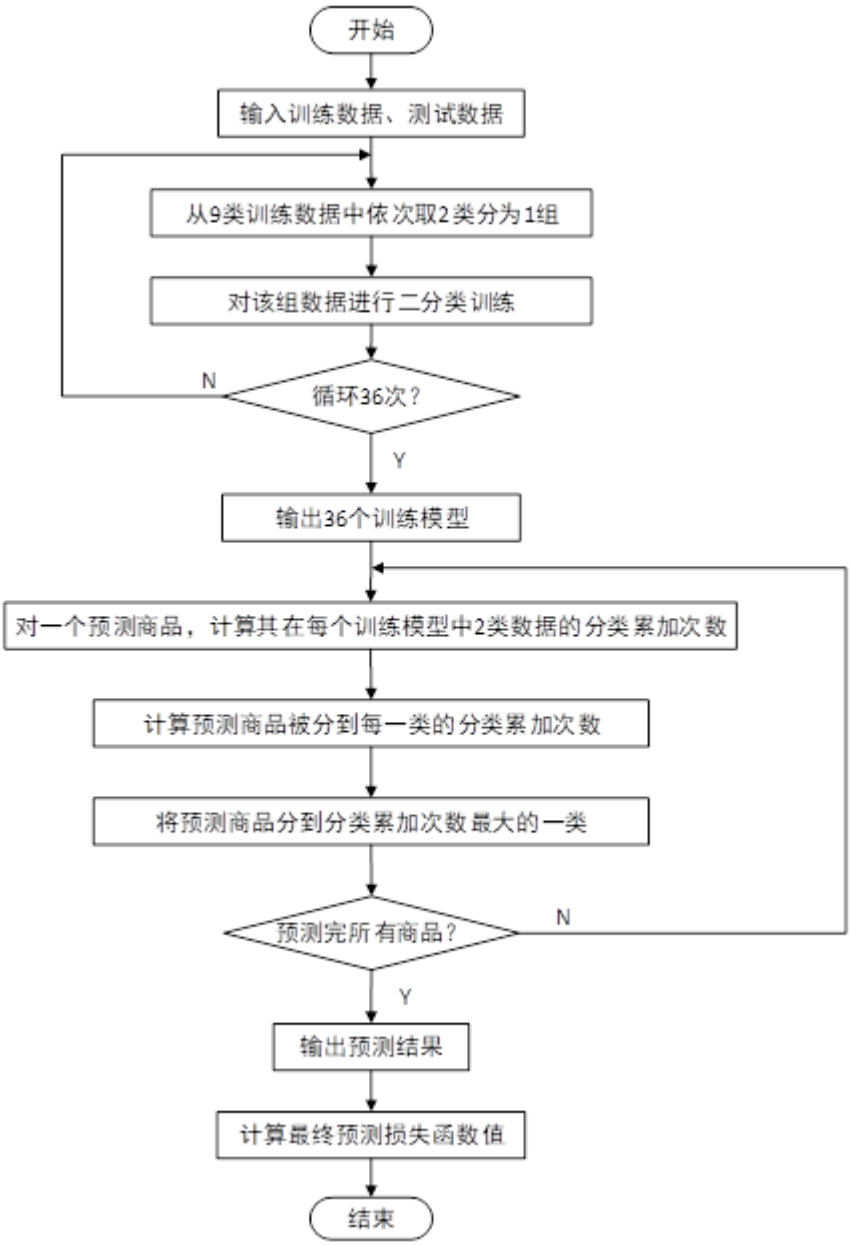

图 1-8 one vs one 流程图

2.3 参数调节

在使用 libsvm 并采用高斯核函数时，为了找到最适合本问题的参数，需要进行参数调节。

高斯核函数有两个非常重要的参数 c 与 g。其中 c 是惩罚系数，即对误差的宽容度。c 越高，说明越不能容忍出现误差，容易过拟合。c 越小，容易欠拟合(c 过大或过小，泛化能力变差，因此尽量保证在 g 最优的情况下取最小的 c 值)，g 是选择高斯函数作为核函数后，该函数自带的一个参数。隐含地决定了数据映射到新的特征空间后的分布，g 越大，支持向量越少，g 值越小，支持向量越多。c 和 g 的取值都会影响到最后的预测精度。

分别设置 c 和 g 变化的范围与步长，每取一组(c,g)通过交叉验证(libsvm 中的参数 v)记录其预测正确率，绘制正确率与参数 c,g 的变化关系图(等高图)，并通过观察重新设置范围与步长，直至搜寻到正确率的最大值。(直接将参数范围设置很广时，会达到 svmtrain 的最大迭代次数，运行时间大幅上升，因此通过重复多次设置较小范围来搜索最优参数)。保存正确率最大值时的 bestc 和 bestg，用于后续训练和测试。

此外，由于每次进行参数寻优都需要产生一个 svm 模型，而每个 svm 的产生时间与数据量有关，为了压缩运行时间，本次参数调节选用 25000 个训练数据(总共 49502 个，大约 50%)，测试结果表明选用的比例不失准确性。

3 实验过程及结果

3.1 BP算法实验

将第二章相关公式转化为 Matlab 程序。本次实验使用三层神经网络，即输入层、一层隐含层、输出层。训练方式为反向传播。

参数设置如下：

以训练一次数据（49502 个）为一次迭代，共迭代 100 次，即训练 4950200 组数据，学习率 α 设置为 0.035，隐含层神经元为 20 个。

迭代过程如图 3-1、图 3-2 所示。

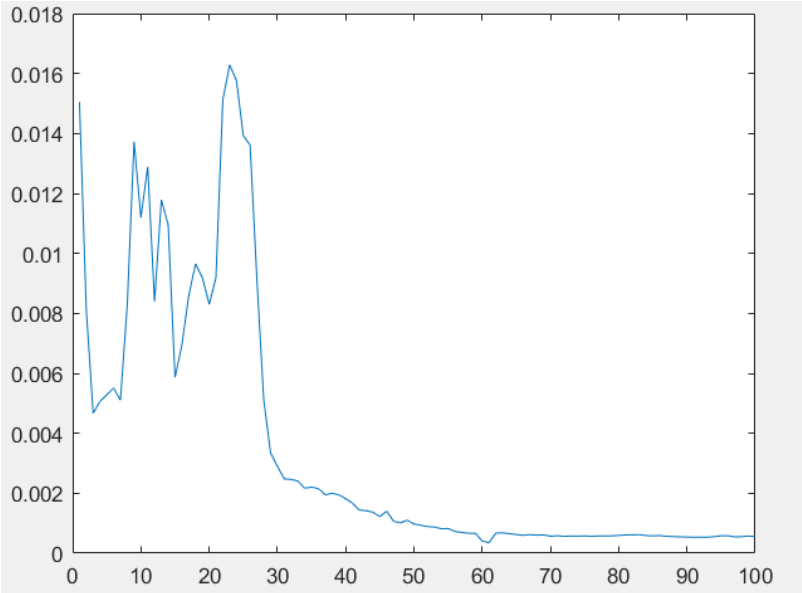

图 3-1 训练时损失函数随迭代的变化

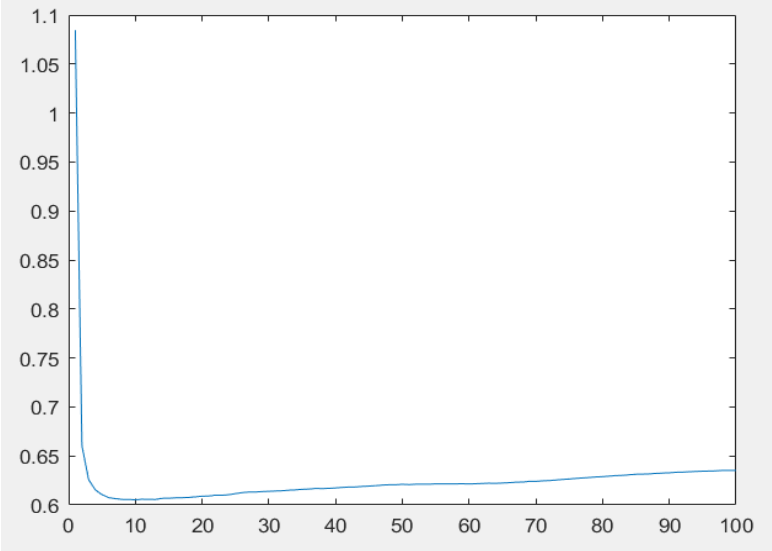

图 3-2 训练时目标函数随迭代的变化

最终的分类结果上传至 Lintcode，得到损失为 0.67134。

从目标、损失函数随迭代的变化可以看出，目标函数的值在迭代初期迅速下降，但是当至 60 次左右后，已十分不明显。而每次迭代中，对最后一组数据（第 49502 组）数据进行分类的损失函数则在第 8 次左右降至最低，之后略有提高，且最小值在 0.6，之后逐步提升至 0.65 左右。可以看出，迭代次数并非越多越好。

3.1.1 softmax 对结果的影响

BP 实际的输出中，虽然各个输出神经元的输出为（0,1）之间，但是所有输出的和并不能保证为 1。softmax 用于多分类过程中，它将多个神经元的输出，映射到（0,1）区间内，使得其输出可以看成概率来理解，从而来进行多分类，使用 softmax 前后对输出进行拟合的结果如表 3-1 所示。

表 3-1 softmax 对结果的影响

输出	损失
不进行输出 softmax	0.67134
进行输出 softmax	1.63300

可以看出没有 softmax 的错误输出反而结果好，原因是 multi-class logarithmic loss 只考虑分类正确的那个概率，尽管 softmax 函数最后的输出概率总和为 1，而不加 softmax 的输出总和不为 1，不加 softmax 的为错误答案，但 softmax 会将正确分类的概率值降低，从而使得分类和为 1。这导致了 LOSS 函数值的升高。这说明，除了 multi-class logarithmic loss，应该考虑使用一个使用所有输出结果的评判函数判断分类器的好坏。

3.1.2 训练数据次数（迭代次数）影响

取隐含层神经元个数为 20 不变，改变迭代次数 j，每次实验所用的实验平台保持一致。结果如表 3-2 所示。

表 3-2 训练数据次数对结果影响

	time	loss
j=10	20s	0.67197
j=100	3m27s	0.67134
j=1000	34m09s	0.78593

可以看出，在保证一定训练数据的前提下，迭代次数对精度提升不明显，迭代次数太多，反而会使精度下降，且需要的时间会大大增加。

3.1.3 隐含层神经元个数影响

取迭代次数 10 次不变，每次实验改变隐含层神经元个数 d1，每次实验所用的实验平台保持一致。结果如表 3-2 所示。

表 3-3 训练数据次数对结果影响

	time	loss
d1=10	19s	0.69264
d1=30	23s	0.62129
d1=100	1m	0.61737

可以看出，在保证一定训练数据的前提下，隐含层神经元个数对分类结果又较大提升，且该个数不需要考虑是否要多于输入层神经元个数。训练时间与神经元个数之间的关系大致成正比，与迭代次数增加相比，该方法可以在不太增加训练时间成本的情况下有效提高分类正确率。

3.2 SVM实验

3.2.1 one vs rest

在理论部分已经提到，这种算法会出现重叠现象以及不可分类现象。实验中对两种情况进行统计(总共 12376 个测试数据)，有 2909 不可分类数据，但并没有重叠现象。同时由于这种方法输出的是类别编码，每个数据正确类别项都会产生 loss 值，不断累加。综合以上两种情况结果可能会产生比较大的误差。结果如表 3-3 所示。

表 3-4 one vs rest SVM 结果

训练时间(s)	测试时间(s)	结果
6853.2	4.19	8.1490

3.2.2 one vs one

1. Fitcecoc

Matlab 没有提供调参的接口，直接使用 fitcecoc 训练模型，predict 函数测试得到结果并提交。

2. Libsvm

由于没有在 MATLAB 自带的 fitecoc 中找到调参的接口，使用 libsvm 进行参数调节的研究，本次采用 50%数据量用于参数调节。首次参数寻优、最后一次参数寻优效果图及各参数图如图 3-3、图 3-4、图 3-5、图 3-6 所示。

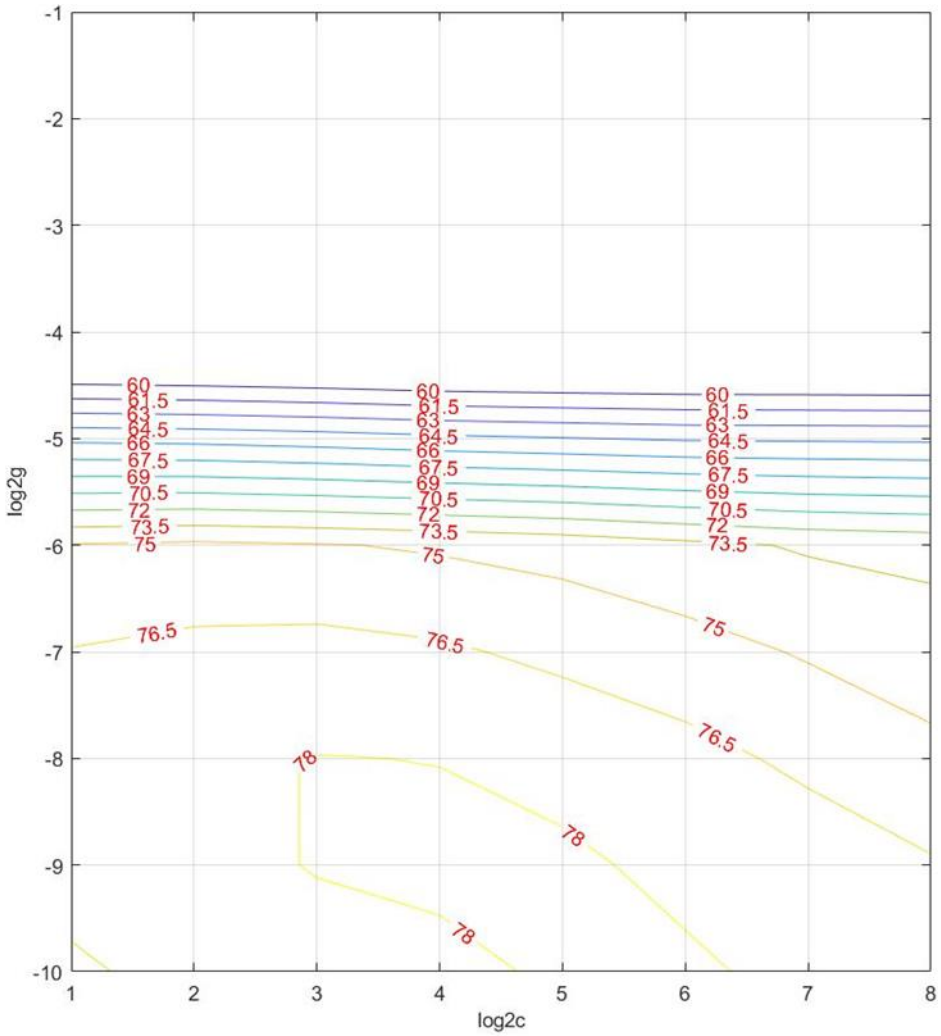

图 3-3 首次参数调节效果图

accstep	1
bestacc	78.2750
bestc	16
bestg	0.0020
cmax	7
cmin	4
cstep	1
gmax	-9
gmin	-14
gstep	1

图 3-4 首次参数调节的参数

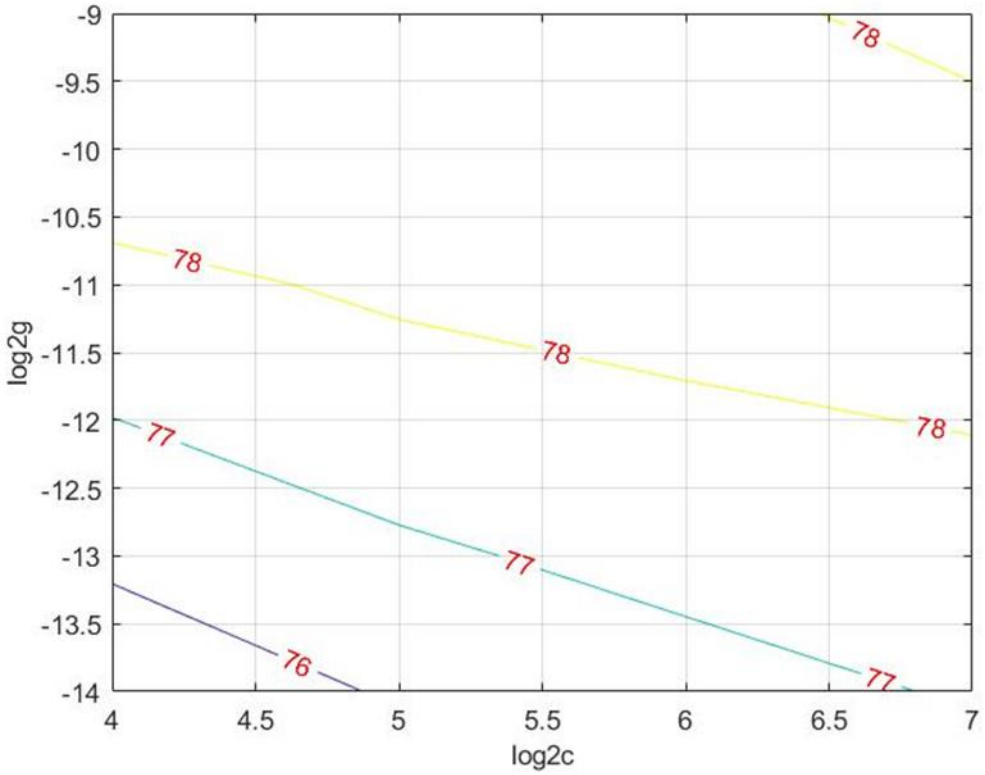

图 3-5 最终参数调节效果图

accstep	1
bestacc	78.9880
bestc	16
bestg	0.0020
cmax	7
cmin	4
cstep	1
gmax	-9
gmin	-14
gstep	1

图 3-6 最终参数调节的参数(选用 bestc，bestg 用于训练与测试)

注：使用 svmpredict 的概率输出模式(-b 1)时，输出的概率不再按原来的类别顺序排序，需要从训练的 svmmodel(structure 类型)中的 label 矩阵中找到顺序，如图 3-7 所示，重新排序以获得最终的预测结果并提交。

svmmodel.Label

	1	2
1	4	
2	8	
3	6	
4	3	
5	2	
6	1	
7	5	
8	9	
9	7	

图 3-7 libsvm 训练后的类别顺序

两种函数的训练结果(也可以认作是调参前后的结果，因为 fitcecoc 未调参)对比如表 3-3 所示。

表 3-5 采用高斯核函数调参前后结果对比

方法	时间(s)	Loss(s)
Matlab 自带多分类(Fitcecoc)	4068.1	1.12690
Libsvm	3417.5	0.52809

4 总结

4.1 算法部分

BP 算法的核心是数学中的“梯度下降法”，即 BP 网络的误差调整方向总是沿着误差下降最快的方向进行。实质上实现了一个从输入到输出的映射功能，而数学理论已证明它具有实现任何复杂非线性映射的功能。这使得它特别适合于求解内部机制复杂的问题；并能通过学习带正确答案的实例集自动提取“合理的”求解规则，即具有自学习能力。

但是 BP 算法也存在一些缺点，BP 算法为一种局部搜索的优化方法，但它要解决的问题为求解复杂非线性函数的全局极值，因此，算法很有可能陷入局部极值，使训练失败。训练能力差时，预测能力也差，并且一定程度上，随训练能力地提高，预测能力也提高。但这种趋势有一个极限，当达到此极限时，随训练能力的提高，预测能力反而下降，即出现所谓“过拟合”现象。此时，网络学习了过多的样本细节，而不能反映样本内含的规律，本次仿真中可以发现迭代次数超过 7 个训练集的数量，评判指标函数的值反而上升，即预测精度下降。通可以通过增加神经元个数来提高精度，并且不会增加很多的时间，直至逼近某个极限(4.1.3 中的结果)。

SVM 的核心思想就是要找到最佳的支持向量集，对于线性可分的情况，SVM 能够基本能够求解，但对于线性不可分的情况，采用普通的 SVM 求解方法需要进行扩维，但扩维可能会使计算量大大增加，甚至可能会无法求解，因此引入了核函数来求解，核函数在 SVM 中的应用非常广泛，针对不同的问题，可以选择合适的核函数来解决。

不同的核函数对算法的影响很大，本实验采用的线性核函数和高斯核函数对最终的损失函数结果有较大影响。同时，容易发现核函数的参数会对最终的结果产生较大影响。因此，为解决一个具体的问题，参数调节是至关重要的，且为了得到的理想的结果，常常需要不断调节参数，以寻找最优结果，当数据较多时，SVM 耗时较长(分别取十个 c,g 的值时，相当于需要生成 10*10*36 个二元 SVM 分类器)。

BP 算法与 SVM 相比，可以发现采用 BP 算法可以更快地解决本问题，且两者时间差别非常大，当合理选择 BP 算法的神经元个数与迭代次数时，可以大大提高求解精度与求解效率，在效率上 BP 算法要优于 SVM。而从结果上 SVM 要略微好一些，但由于 BP 采用的是三层神经网络，继续增加隐含层数时(深度学习),神经网络的整体性能可能会更佳。

4.2 实验感想

1)本次实验的 BP 算法仅仅研究三层神经网络对于本问题的性能，当神经元个数，数据迭代次数到达一定次数后，预测性能会达到极限，后续需要进行深度学习部分的进一步学习才能继续提高预测的准确率。

2)本次实验 SVM 算法在参数寻优时为了缩小程序运行时间，只采用了 50%训练数据进行训练，并且寻优的指标是预测的正确率而非 lincode 上给出的类别预测概率 multi-class logarithmic loss，此外，数据预处理也采用了最简便的最大最小归一化没有做进一步的研究，后续对这三个方面进行改进，最后结果应该能进一步提高(目前交叉验证正确率为 79)。

3)SVM 算法中通过集成 9 个分类器完成了 one vs rest 的多分类效果但是训练时间居然远远超过了 libsvm 和 fitcecoc 的 one vs one 集成 36 个分类器的方法，说明了算法内部还有很多方面能够优化。此外，由于没有掌握概率输出的方法导致了 one vs rest 效果很差。

4)通过实际的编程仿真，进一步了理解了相关算法的理论以及掌握了部分调优的策略。

附：Github 文件说明

SVM 文件夹：

One-vs-the-rest/changelabel.m、cross.m、ecsvm.m、svmtest.m 一对多 SVM 的相关 Matlab 函数

ONE-VS-ONE/fitcecoc/postsvm.m 是使用 fitcecoc 进行训练的相关 Matab 函数

ONE-VS-ONE/libsvm/ Normalization.m、SVMcg.m、libsvmnorm.m 使用 libsvm 的相关 Matlab 函数，其中 Normalization.m 是数据预处理，SVMcg.m 是参数调节

BP 文件夹：

BP.m、D1F.m、delta.m、loss.m 是 BP 算法的相关 Matlab 函数