华南农业大学

课程设计

基于元素含量的玻璃种类识别

何　唯

201430120308

学　院　名　称　数学与信息学院

专　业　班　级　14 级信息与计算科学

提　交　日　期　2016 年 06 月

评阅分数 _____

评阅人 _____

基于元素含量的玻璃种类识别

摘　要

本课程设计基于 Kaggle 提供的玻璃类别数据集 glass.csv(6 种玻璃的 214 个样本，其中每个样本具有 9 个特征)，建立 4 个数学模型分别与其他常用的机器学习分类器在本问题上的识别率对比，提出在多分类以及二分类的问题定义背景下对应的最优玻璃类别分类器。

建立模型之前，首先对数据进行预处理，将输入向量按照特征分别进行归一化，剔除异常的数据样本以及带有缺失值的数据样本。然后对样本数据的分布进行分析，判断是否适用或者需要利用主成分分析等算法对数据进行降维。最后，利用交叉验证法将样本集进行随机划分。

建立模型一多分类的 BP 神经网络，首先，对数据集按照 Matlab 模式识别工具箱的输入进行规范处理；然后，根据经验公式以及多次训练，确定模型的参数以及获得收敛得到最优解对应的权值矩阵以及阈值数据；接着，利用训练好的模型，分析以及检验识别的准确率以及有效性；最后，得到识别率为 **75.7%**模型用于之后的仿真以及测试。

建立模型二基于 Bagging 集成学习的多分类随机森林模型，根据划分的数据集利用自助采样法(Bootstrap sampling)，得到多个样本集，从而训练出若干个基学习器；这里的基学习器设置为决策树，对每个个体学习器都利用随机森林对每个结点的属性集合中随机选取子集，再从此子集中按规定选择最优属性进行划分；接着，利用投票法集成多个个体学习器，用于实现玻璃类别的多分类；最后利用训练好的 Bagged Trees 模型，对应识别正确率为 **79.9%**，用于对其他玻璃数据进行仿真，分析识别效果。

建立模型三以及模型四，分别与模型一和模型二的训练方法相似，改变的是期望输出的向量改为二维，各分量取值为 $\{0,1\}$ 的向量。分别得到识别正确率为 **95.3%，95.8%**的分类器模型。最后，分别基于多分类以及二分类问题背景，将建立的模型与常用分类器（如支持向量机、KNN 算法、线性判别、Logistics 回归等等）进行识别效果对比，结果表明此两种算法对于本分类数据具有相对准确的表现。

在模型的改进部分，利用主成分分析(PCA)对于同一数据集的输入向量进行降维，得到相应的的解释变量百分比分别为：**45.4%, 18.0%, 12.6%, 9.8%, 6.9%, 4.2%, 2.6%, 0.4%, 0.0%**。将得到降维后的数据重新放入上述四个模型，对应的识别率没有得到明显提升，说明问题不需要用主成分分析提取特征。

关键词：BP 神经网络；决策树；Bagging 集成学习；随机森林；主成分分析

目　　录

一、问题重述.....1
二、问题分析.....1
三、模型假设与约定.....3
四、 符号说明及名词定义.....3
五、问题一各模型的建立.....4
5.1 数据预处理.....4
5.2 模型一 多分类的 BP 神经网络.....4
5.2.1 隐层数的确定.....4
5.2.2 隐层神经元个数的确定.....5
5.2.3 模型一的求解.....6
5.2.4 模型一的检验.....7
5.3 模型二 基于 Bagging 的多分类随机森林模型.....10
5.3.1 Bagging 集成学习方法.....11
5.3.2 随机森林(Random Forest).....11
5.3.3 模型二的参数确定.....12
5.3.4 模型二的检验.....12
5.4 模型三 二分类的 BP 神经网络.....16
5.4.1 模型三的求解.....16
5.4.2 模型三的检验.....16
5.5 模型四 基于 Bagging 的多分类随机森林模型.....18
5.5.1 模型四的建立.....18
5.5.2 模型四的检验.....19
5.6 问题一总结.....22
六、 问题二的分析.....23
七、模型评价.....24
7.1 模型优点.....24
7.2 模型缺点.....24
八、模型推广与改进.....24
8.1 主成分分析.....24
参 考 文 献.....26
附　录.....27

一、问题重述

随着人们生活水平的日益提高，大多数人开始关心日常生活中一些简单物品的化学组成或者成分组成。一些生产商有时候为了降低成本，常常生产一些货不对板的商品给消费者，而不知情的消费者在使用这些不对版的商品的时候，常常不能满足自己的使用需求，使用不当的时候甚至会导致一些意外发生。

对于日常生活中一些常见的玻璃，我们如何根据其检测的组成元素成分识别此玻璃的类型呢？如何可以根据玻璃组成成分对玻璃生产产业以及玻璃制造质量控制提出建议呢？

- 问题一：根据提供的玻璃组成成分数据集，建立相应种类的判别模型，并根据课程设计所建立的模型，对特定数据进行测试并完成玻璃种类的识别；
- 问题二：对玻璃工业生产提供质量控制的意见以及建议。

二、问题分析

本课程设计完成的是基于元素含量的玻璃种类识别，通过 Kaggle 提供的玻璃数据集进行建模。问题给出了 214 组样本，每组样本给出了 9 个玻璃成分元素特征(RI、Na、Mg、Al、Si、K、Ca、Ba、Fe)以及对应的玻璃类型(共 6 种类型)。数据属性解释如下图一所示。因此，问题的解决方案为根据给定的数据建立分类模型，构造出一个由样本中的 9 个特征到 6 种类型的函数映射或者对应法则，如:

$$f:(RI,Na,Mg,Al,Si,K,Ca,Ba,Fe)\rightarrow(Type_1,Type_2,Type_3,Type_5,Type_6,Type_7) \quad (1)$$

对于此类型问题，将 m 维向量映射到对应 n 维向量上，我们可以考虑 BP 神经网络、决策树、支持向量机(SVM)、线性判别等常用分类器来描述此关系。

表一 数据属性字段解释

数据属性	定义
RI	折射率(refractive index)
Na	元素钠(Sodium)
Mg	元素镁(Magnesium)
Al	元素铝(Aluminum)
Si	元素硅(Silicon)
K	元素钾(Potassium)
Ca	元素钙(Calcium)
Ba	元素钡(Barium)
Fe	元素铁(Iron)
Type of glass	类别(class attribute)

说明：单位元素度量标准为其对应氧化物的质量百分比，玻璃类型的定义如下(其中第4种没用在数据库中)：

Type1: Building_windows_float_processed

Type2: Building_windows_non_float_processed

Type3: Vehicle_windows_float_processed

Type4: Vehicle_windows_non_float_processed (none in this database)

Type5: Containers

Type6: Tableware

Type7: Headlamps

三、模型假设与约定

1.假设样本集中的数据准确无误，并能准确反映特定种类玻璃的元素组成；

2.假设讨论的各种玻璃均处于稳态，元素成分不会发生突变；

3.假设样本集中的样本标签均正确；

4.假设样本集中各个类别玻璃分布情况相同。

四、符号说明及名词定义

符号	说明
input	输入向量
output	输出向量
in	数据样本矩阵
glass.csv	玻璃数据集
m	隐层神经元个数
n	输入层神经元个数
l	输出层神经元个数
α	区间$[1,10]$的常数
E	网络误差信号
TP	真正例个数
TN	真反例个数
FN	假反例个数
FP	假正例个数
TPR	真正例比率
FPR	假正例比率

五、问题一各模型的建立

5.1 数据预处理

首先，需要对数据集 glass.csv 中的缺失值以及奇异值进行处理；

然后，对于玻璃数据集中所有输入特征分别进行归一化处理，归一化的原因是：

1. 输入向量中，各个特征有不同的物理意义，归一化进行变换后所有分量都将落入$[0,1]$区间中。
2. 比如后面用到的 BP 神经网络模型中，隐层激活函数采用 Sigmoid 函数，归一化后可防止因为净输入绝对值过大导致的神经元输出饱和(比如，输入过大神经元输出将非常接近 1，输入过小神经元输出将非常接近 0)。

这里归一化采用公式：

$$input_i' = \frac{input_i - Min(input_i)}{Max(input_i) - Min(input_i)} \qquad ①$$

其中$input_i$为第 i 个输入特征，在本课程设计中$i = 1,2,...,9$，$Min(input_i)$表示所有输入样本第 i 个输入特征的最小值，$Max(input_i)$表示所有输入样本第 i 个输入特征的最大值。

数据预处理后的数据集为附件中的 NNInput_data.mat。

5.2 模型一 多分类的 BP 神经网络

BP 神经网络是利用误差反向传播训练算法的前馈型网络，主要用于实现非线性映射，也是目前使用的最广泛的一种神经网络。

5.2.1 隐层数的确定

根据文献[1]，具有单隐层的 BP 神经网络可以映射所有连续函数，两个隐层足以映射所有函数(包含不连续的函数)。在此我们先考虑设计一个隐层，当一个

隐层不能满足我们对于玻璃分类正确率的要求的时候，再设计一个具有两个隐层的 BP 神经网络。

5.2.2 隐层神经元个数的确定

常用的隐含层节点常用方法有试凑法以及经验公式。如：

$$m = \sqrt{n+l} + \alpha \qquad ②$$

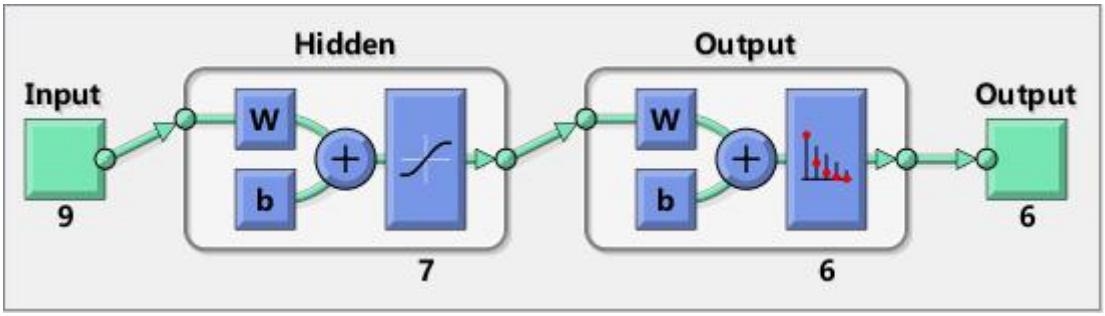
$$m = \sqrt{nl} \qquad ③$$

其中，m 为隐层神经元个数；n 为输入层神经元个数；l 为输出层神经元个数；α 为在区间$[1,10]$的常数。

本模型将采用公式③。由于网络的输入向量是九维向量，由玻璃各成分组成描述，因此输入层设置九个神经元；网络的输出向量是一个六维判别向量，对应六类玻璃的识别结果，因此输出层应设置六个神经元。我们可以确定隐层神经元个数：

$$m=\sqrt{nl} = \sqrt{9\times 6} = 7.348 \approx 7$$

综上，可以设计一个 9-7-6 的 BP 神经网络，如下图一。



图一 9-7-6 的 BP 神经网络结构图

5.2.3 模型一的求解

利用 Matlab 的神经网络模式识别工具箱，我们可以训练出网络的权值矩阵以及获得网络的仿真结果。

模型将采用传统的 BP 算法（Backpropagation），即将数据集作为输入信号正向传播进网络，经过隐层输出层得到网络输出，将网络输出与期望输出比较，产生误差信号;将误差信号按正向传播的反方向传播回网络，调整权值以及阈值，使得网络的误差变小。

具体建模的步骤如下：

首先，根据上述归一化处理后的数据集，我们通过交叉验证法，将数据集划分为 3 个互斥子集，每个子集都保持数据分布的一致性。利用 Matlab，我们将 214 个样本随机分为 150 个样本的训练集(70%)，32 个样本的验证集(15%)以及 32 个样本的测试集(15%)；

然后，初始化各层权值矩阵 w 以及 v，利用共轭梯度法（Scaled Conjugate Gradient）各层反向传播误差信号进行训练，调整权值以及阈值；

Matlab 提供的训练方法有很多种，比如说常用的 Levenberg-Marquardt 算法、BFGS Quasi-Newton 算法。这里采用共轭梯度法（Scaled Conjugate Gradient）的原因在于共轭梯度法（Scaled Conjugate Gradient）需要耗费的空间复杂度相对前两者低，只比简单算法要求多一点存储空间，适合解决小规模的问题且当网络中的激活函数可导即可使用。

最后，计算均方根误差(MSE)作为误差信号，在精度范围内就跳出并输出网络结果，否则，将继续调整权值以及阈值直至满足精度（此处精度选择默认的 0，当 Matlab 计算结果十分接近 0 时也会跳出）。

误差信号的定义如下：

$$E = \frac{1}{2}(d-o)^2 = \frac{1}{2}\sum_{k=1}^{l}(d_k - o_k)^2 \qquad ④$$

共轭梯度算法基本原理如下：

反向传播误差信号根据误差导数传递误差信号，见公式：

$$X=X+a\times dX$$

其中 X 为权值以及阈值的构成矩阵，dx 为搜素方向，参数 a 是用于让误差随着搜索方向最小化。

第一个搜索方向为误差梯度的负方向，之后每一次迭代的搜索方向根据前一次的搜索方向以及计算新的梯度确定，如下：

$$dX=-gX+dX\_old\times Z$$

其中 gX 为梯度，参数 Z 可根据不同方法计算得出，这里采用 Matlab 计算的默认方法 Fletcher-Reeves variation 共轭梯度法得到：

$$Z = normnew\_sqr / norm\_sqr$$

其中 norm_sqr 表示前一梯度范数平方，normnew_sqr 表示当前梯度的范数平方。

共轭梯度算法在以下几种情况也会跳出训练：

- 达到限制的最大迭代次数（这里取默认的 1000）；
- 达到限制的训练时间（这里取 inf，即不限制）；
- 梯度下降到小于设定的最小梯度（这里取10^{-6}）

具体模型建立和训练以及参数确定代码见附件。

5.2.4 模型一的检验

由于每次 Matlab 在训练神经网络的时候，对于同一组训练样本，初始化权值矩阵和阈值的数值是不一样的，为了得到最优解，这里对于同一组参数训练多次。多次训练后，我们可以得到识别正确率为 **75.7%**的多分类模型用于对玻璃类别的识别。（各层训练后权值矩阵以及阈值见附件中模型一的模型参数，其中隐层权值矩阵为9×7维；输出层权值矩阵为7×6维）

我们可以画出分类结果的混淆矩阵如下图二，根据混淆矩阵，可以初步断定模型的有效性，根据训练集以及测试集的识别率也可以知道模型没有发生过拟合现象。在多分类问题中，75.7%的总识别率是可以接受的。

除此之外，我们也可以画出模型对于分类的 ROC 图。ROC 图也可以反应模型分类的性能度量。根据学习启德预测结果对样本进行排序，按照顺序逐个把样本作为正例进行预测，每次计算出两个重要量的值，分别以它们为横、纵坐标左图，得到 ROC 曲线。其中，纵轴是真正例率，横轴是假正例率，分别可以通过如下公式计算得到：

$$TPR=\frac{TP}{TP+FN}，$$

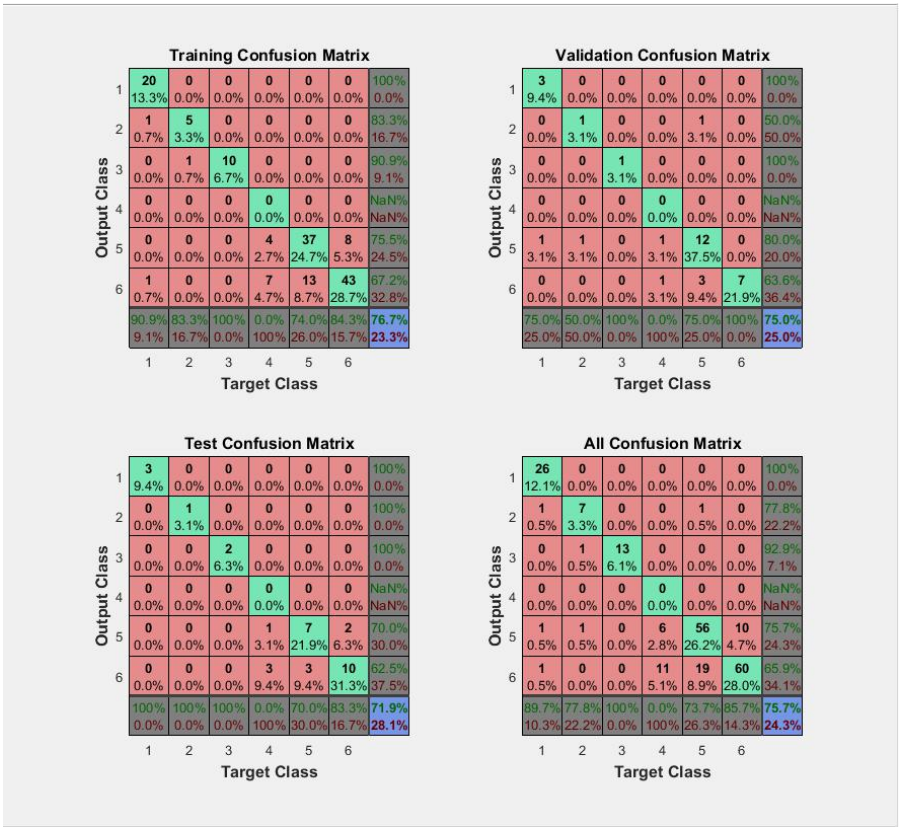
$$FPR=\frac{FP}{TN+FP}$$

其中，TP 为真正例个数，TN 为真反例个数，FN 为假反例个数，FP 为假正例个数，并显然有

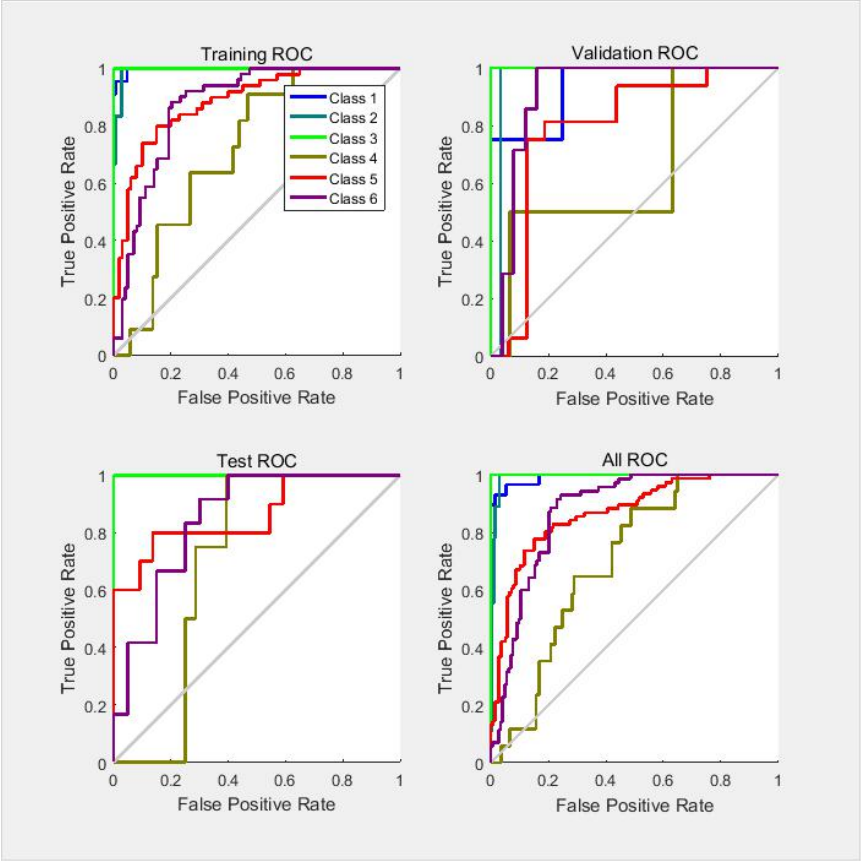
$$TP+FP+TN+FN=样例总数$$

因此，我们可以根据 ROC 曲线下的面积判断识别效果的好坏，即观察 AUC(Area Under ROC Curve)。

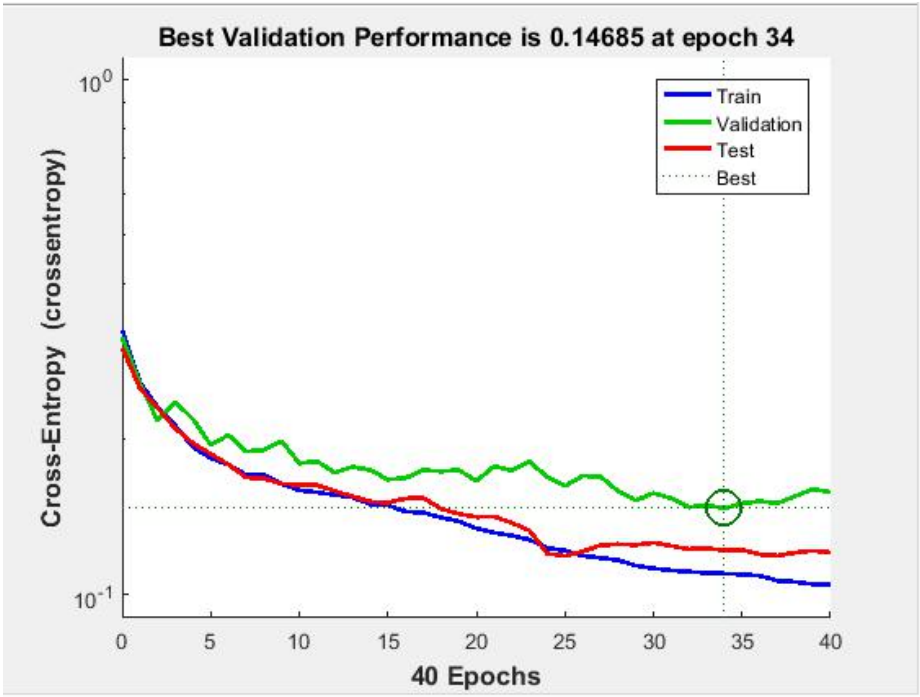
由所画的如下两个图，我们可以发现，模型除了对于第四类玻璃的识别效果相对没那么好之外，其他效果还是可以接受的。后续我们在改进模型的时候可以考虑从第四类样本出发。



图二 模型一的混淆矩阵(Confusion Matrix)



图三 模型一的“受试者工作特征”图(ROC Figure)



图四 模型一的训练集、验证集、测试集每次迭代误差

5.3 模型二 基于 Bagging 的多分类随机森林模型

Bagging 是并行式集成学习方法著名的代表。集成学习是通过结合多个分类学习器来完成学习任务，一般结构为：先生成一组“个体学习器”，再通过某种策略将它们结合起来。个体学习器为我们常见都几种分类器：决策树、BP 神经网络、支持向量机等等。

基学习器表示同质集成中的个体学习器，相应的学习算法成为基学习算法，当然集成也可以包含不同类型的个体学习器，比如同时包括决策树和神经网络，这样的集成是异质的。

常见的集成学习器包括个体学习器存在强依赖关系的、必须串行生成的序列化方法（代表为 Boosting）以及个体学习器中没有强依赖关系、可同时生成的并行化方法（代表是 Bagging 和随机森林）。

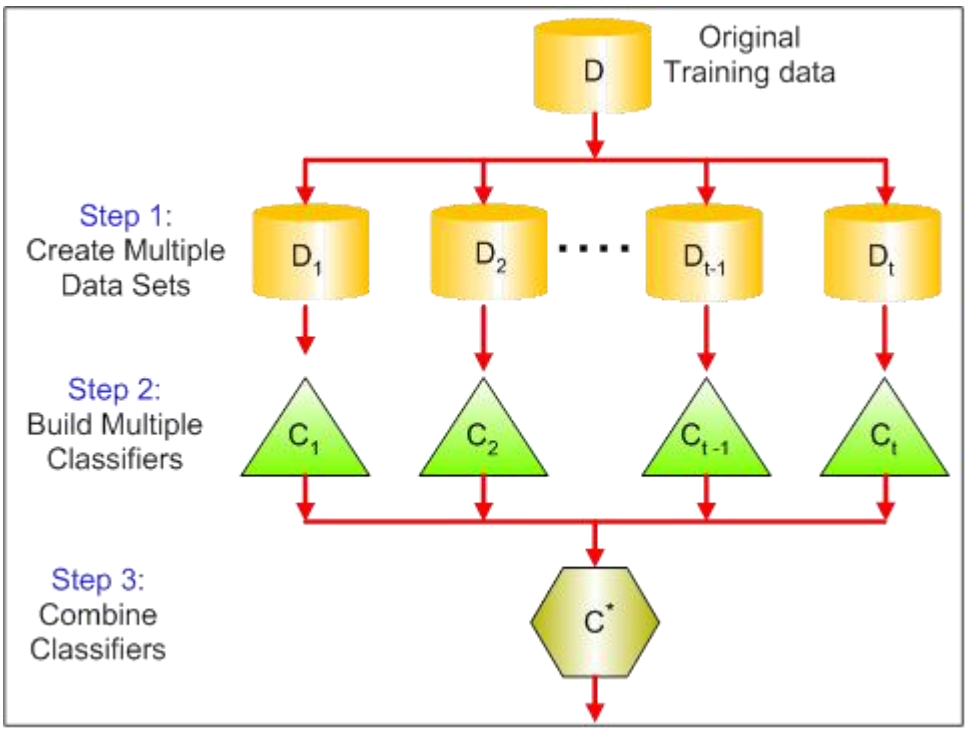
模型二利用 Matlab 提供的集成学习工具箱，建立基于 Bagging 集成方法的 Bagged Trees 模型(前身是 Treebagger 类)。

5.3.1 Bagging 集成学习方法

Bagging 算法基于自助采样法，给定包含 m 个样本的数据集 D，随机将一个样本放入采样集，再将该样本放回原始数据集，使得下次仍有机会选中此样本，达到采样相互有交叠使得训练出来的模型更有泛化能力，十分适合于多分类问题。

这样我们将 T 个含有 m 个训练样本的样本集{D1,D2,...,DT}执行 T 次，训练出 T 个基学习器{h1(x),h2(x),...,hT(x)}，之后再通过投票法或平均法将这些基学习器组合，对于分类问题 Bagging 常用的方法为简单投票法。

具体 Bagging 集成学习方法的实现流程，如下图五所示。



图五 Bagging 集成算法流程图

5.3.2 随机森林(Random Forest)

随机森林为在以决策树为基学习器构建 Bagging 集成的基础上，进一步在决策树训练过程中加入随机属性选择的算法。相对于传统决策树在划分属性（假定有 d 个属性）时选择当前最优属性（根据信息熵以及信息增益计算），随机森林

对于基决策树的每个结点，先从结点的属性集合中随机选择一个包含 k 个属性的子集，再从此子集中挑选最优属性进行划分。参数 k 严格控制随机性的引入[5]。

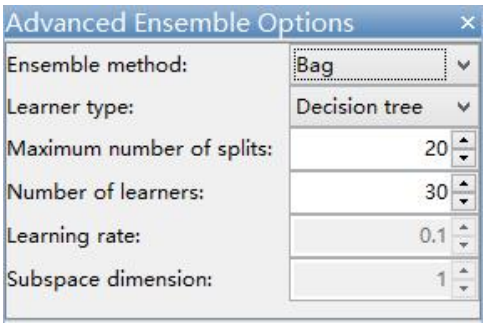
令 k=d，则基决策树构建与传统决策树相同；若令 k=1，则是随机选择一个属性用于划分；本课程设计根据推荐值令 $k=\log_2 d$ 。

5.3.3 模型二的参数确定

数据集的预处理与模型一相似，先进行输入向量的归一化，除此之外输出向量格式为了满足 Matlab 工具箱的要求，将设置为一维的，且数值在集合 $\{1,2,3,5,6,7\}$ 中。

训练集的划分同样采用交叉验证法，这里设置为 5 折交叉验证。

在 Matlab 中设置集成学习方法为 Bag，基学习器类型为决策树，决策树最大分支为 20，学习器的数目为 30，学习率为 0.1。



图六 模型二参数设定

5.3.4 模型二的检验

训练后，我们可以得到识别正确率到 **79.9%**的 Bagged Trees 模型用于对玻璃的分类识别。利用训练后的模型，我们可以调用决策树森林的结构图(如图七)，

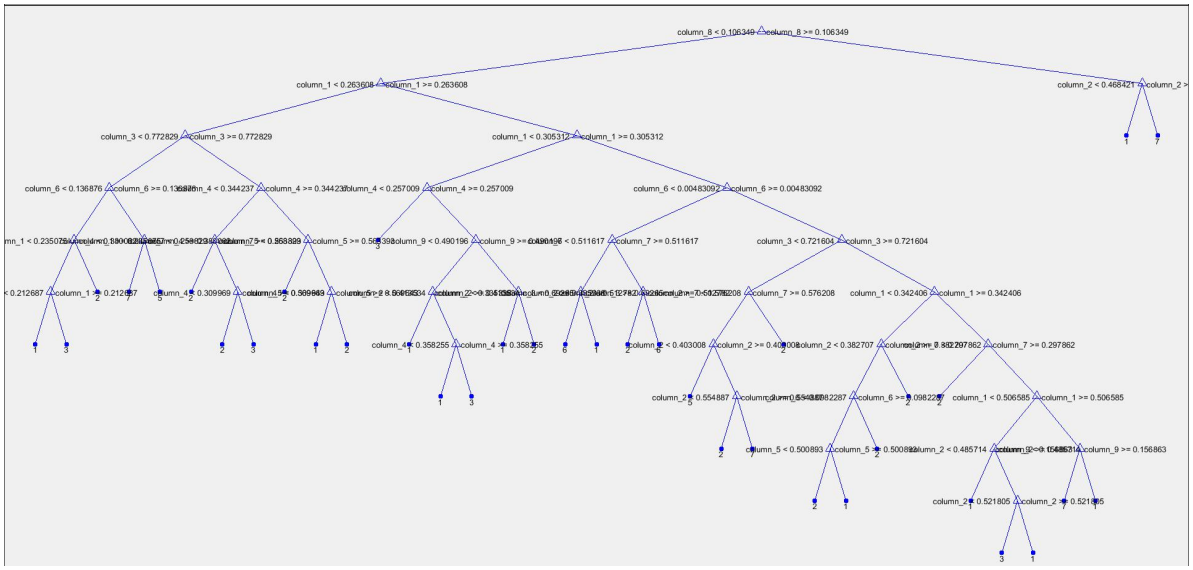
```
view(BaggedTreestrainedClassifier.ClassificationEnsemble.Trained{1})
```

导出的模型参数数据见附件文件 model2date.mat，可用于模型的推广以及额

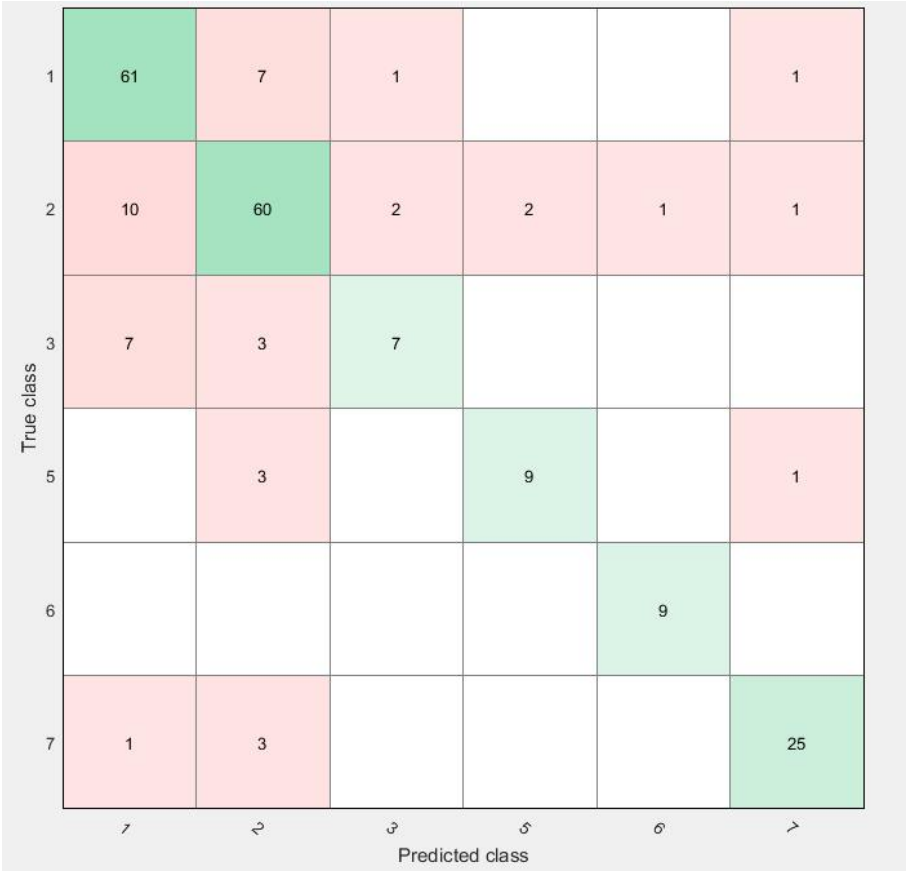
外测试数据的实用，包括分类边界数据，包外估计(out-of-bag estimate)数据等。

可以分别画出基于 Bagged Trees 的玻璃分类混淆矩阵（如下图八九）以及对于六类玻璃分类的 ROC 曲线（见附录附图一到附图六）。可以发现模型二除了对于第三类玻璃的分类效果不理想外(由于第三类玻璃样本比较少)，其他结果均比模型二较好，这体现出了集成学习器对于多分类问题上的优势。对于第三类玻璃的分类效果见图十。

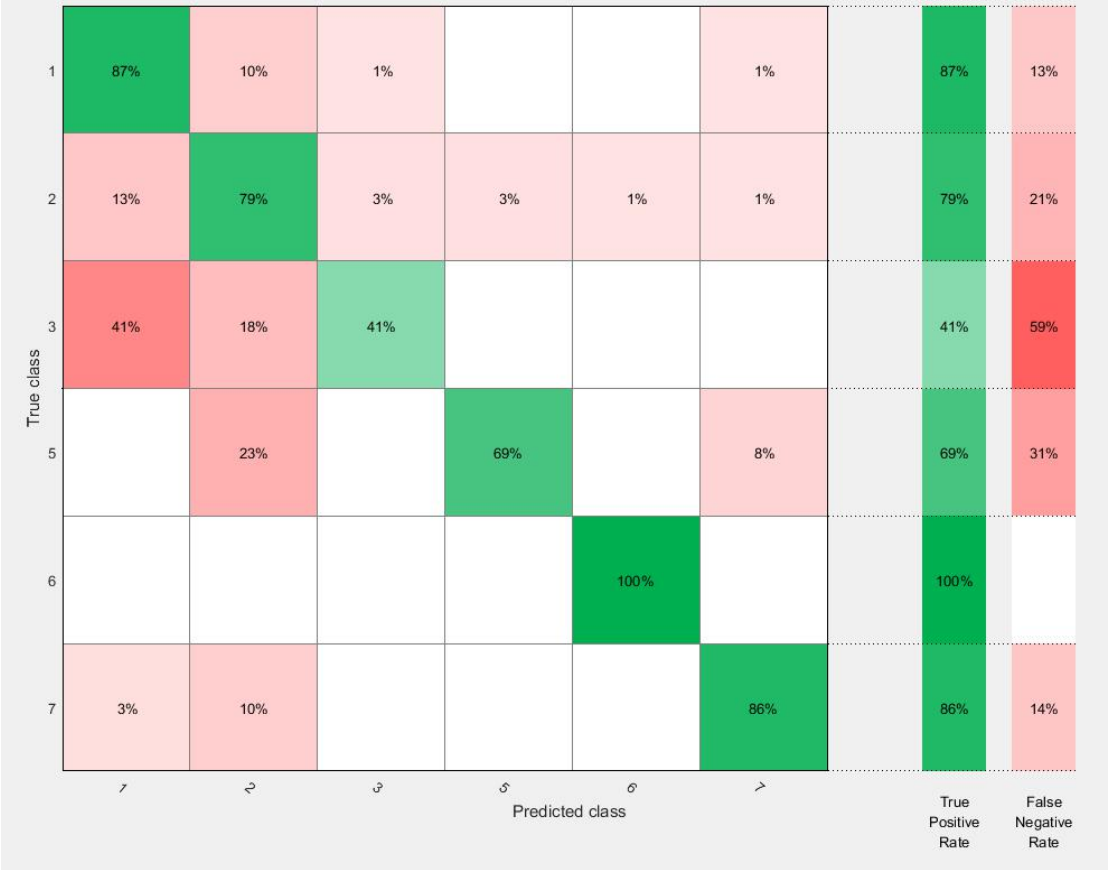
除此之外，由文献[6]，模型二比模型一在多分类问题上具有更好的泛化能力。



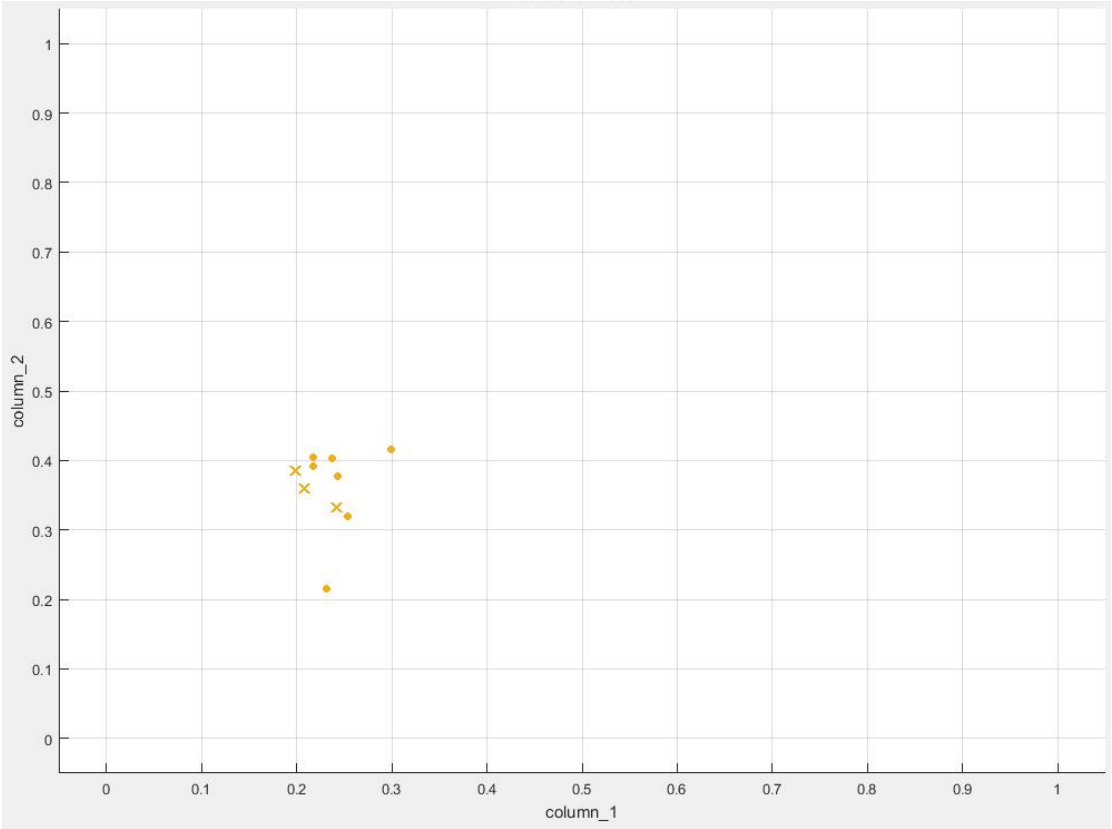
图七 模型二训练后得到的分类树



图八 模型二样本数混淆矩阵（Confusion Matrix）



图九 模型二识别率混淆矩阵（Confusion Matrix）



图十 第三类玻璃分类效果（以前两个特征为例，×代表识别错误）

由上图十可以知道第三类玻璃样本个数本身就比较少，因此识别错误率也相对较高，改进模型可以通过添加足够的样本进行。（可以的话）

基于玻璃多分类除了模型一 BP 神经网络（识别率 **75.7%**）以及模型二基于 Bagging 的多分类随机森林 Bagged Trees（识别率 **79.9%**）以外，基于本课程设计数据集，还分别进行了核函数为高斯函数的支持向量机（Medium Gaussian SVM），加权 K-NN 算法，复杂决策树(Complex Decision Tree)，线性判别分析等分类器模型的训练，识别率如下：

表二 其他分类器对于玻璃多分类的识别效果

分类器名称	多分类识别率
Medium Gaussian SVM	67.3%
Weighted KNN	71.0%
Complex Tree	68.2%
Linear Discriminant	58.4%

对于以上模型的识别率，均没有达到 80%，这里再次观察数据集中的玻璃种类，由表一可得其实可以将样本集分为两类，将多分类问题改为二分类问题处理，即将 Type1-Type3 分为 Windows 玻璃，Type5-Type7 分为 Non-Windows 玻璃。这样就有如下模型三以及模型四。

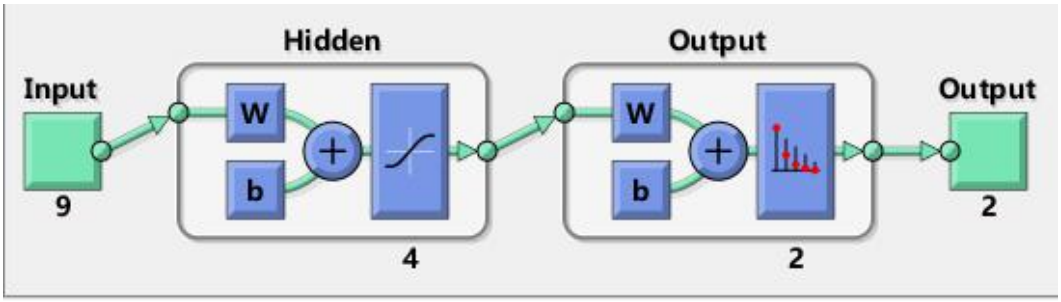
5.4 模型三 二分类的 BP 神经网络

5.4.1 模型三的求解

对于模型三的参数设置以及算法实现方法基本与模型一一致，采用交叉验证法分割样本集，此处将不作进一步介绍，需要改动的是输出向量中的维数改为二维的，即 Type1-Type3 期望输出为$[1,0]$，Type5-Type7 期望输出为$[0,1]$。而输入向量数据不变，隐层设置为一层，计算隐层神经元个数，同样利用经验公式可得，

$$m=\sqrt{nl}=\sqrt{9\times 2}=4.243\approx 4$$

网络拓扑结构如下图十一：



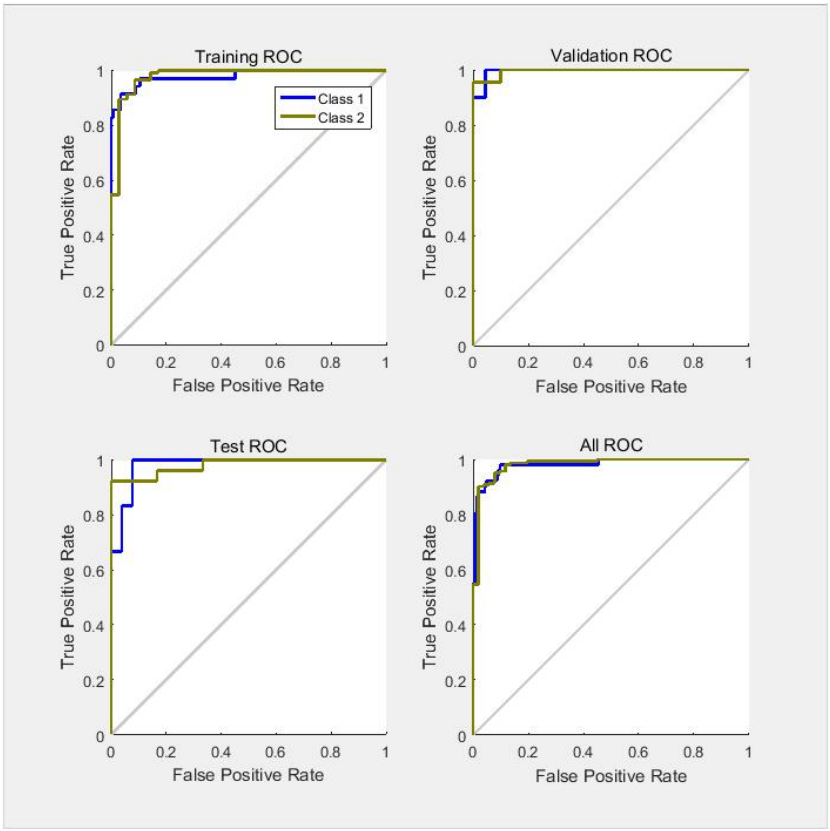
图十一 模型三的拓扑结构

通过 Matlab 模式识别工具箱训练，即可得模型三的权值矩阵以及阈值数据

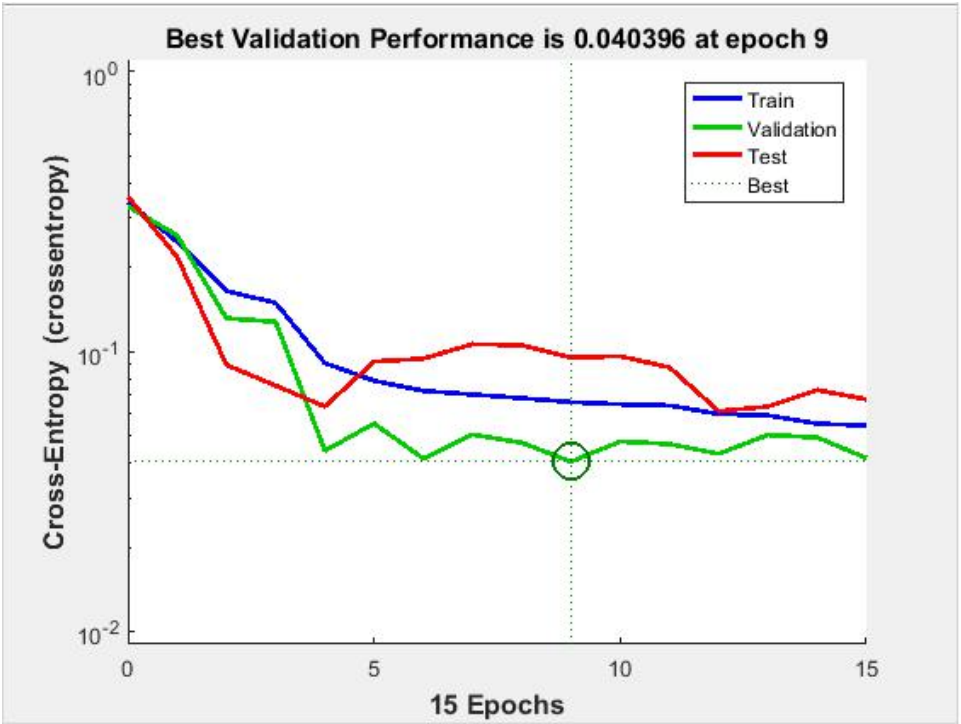
5.4.2 模型三的检验

多次训练后，我们可以得到分类总识别率为 **95.3%**的 BP 神经网络分类器，具体分类器权值矩阵以及阈值数据见附件 **Model3Result.mat**。模型识别混淆矩阵如下图十二：

图十二 模型三的识别结果混淆矩阵（Confusion Matrix）



图十三 模型三的识别结果 ROC 图



图十四 模型三的训练集、验证集、测试集每次迭代误差

由上图十三模型三的识别结果 ROC 图、图十四训练集、验证集、测试集每次迭代误差，我们可以发现模型三相对前两个模型识别率有着显著地提高。一方面是由于，对于多分类问题来说，二分类相对容易识别，以及 BP 神经网络对于神经元总数低的情况下模型收敛速度比较快。

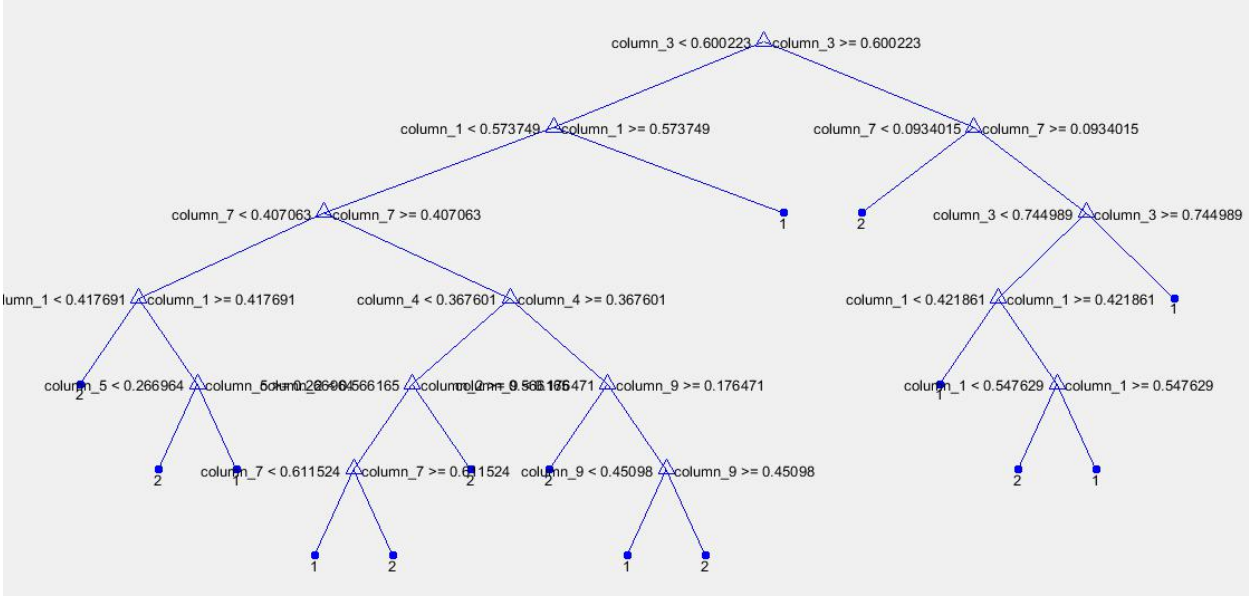
5.5 模型四 基于 Bagging 的多分类随机森林模型

5.5.1 模型四的建立

与模型一模型二的对比相似，我们在此再次利用随机森林构建玻璃分类模型。与模型二不同，模型四输出向量维数修改为一维，取值为{1,2}。

除此之外，模型的学习率、基学习器种类，集成学习方法、最大分支数目均不改变，但改变学习器的数目为 50 个。经过 Matlab 模式识别工具箱训练后，得到识别率达到 **95.8%**的 Bagged Trees 模型。（模型结构如下图十五所示）

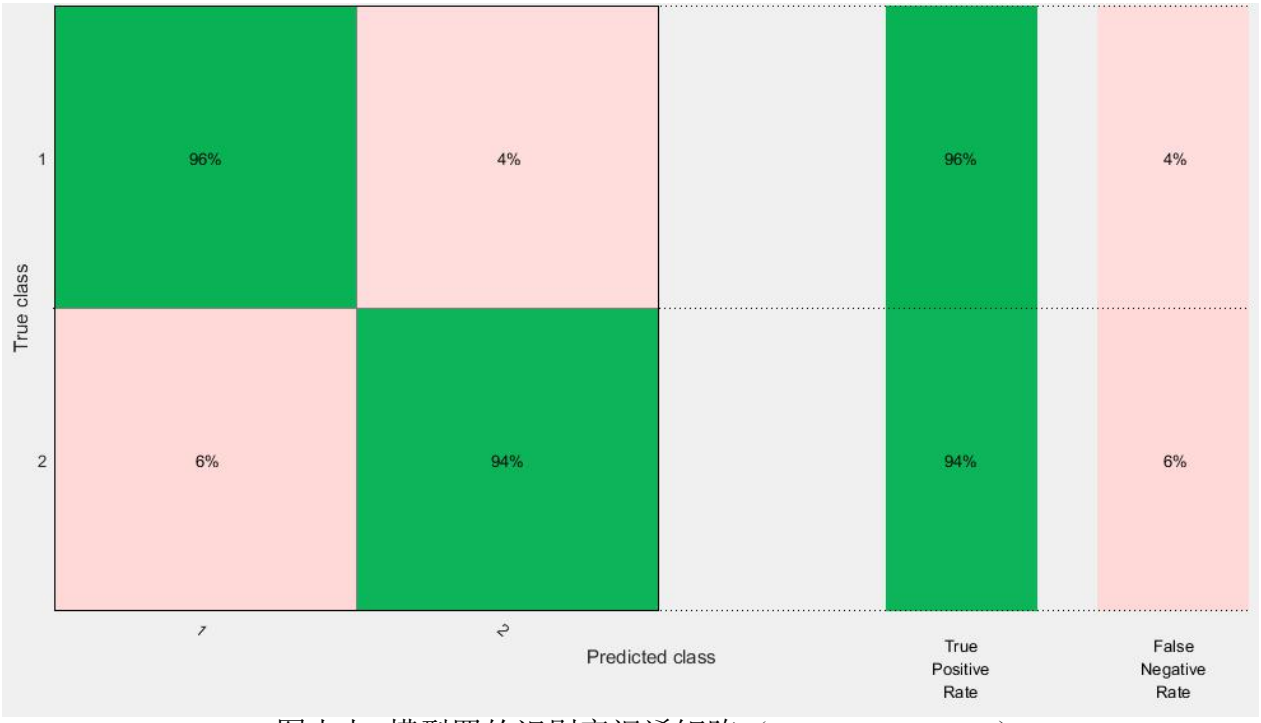
具体模型参数见附件 Model4Result.Mat。



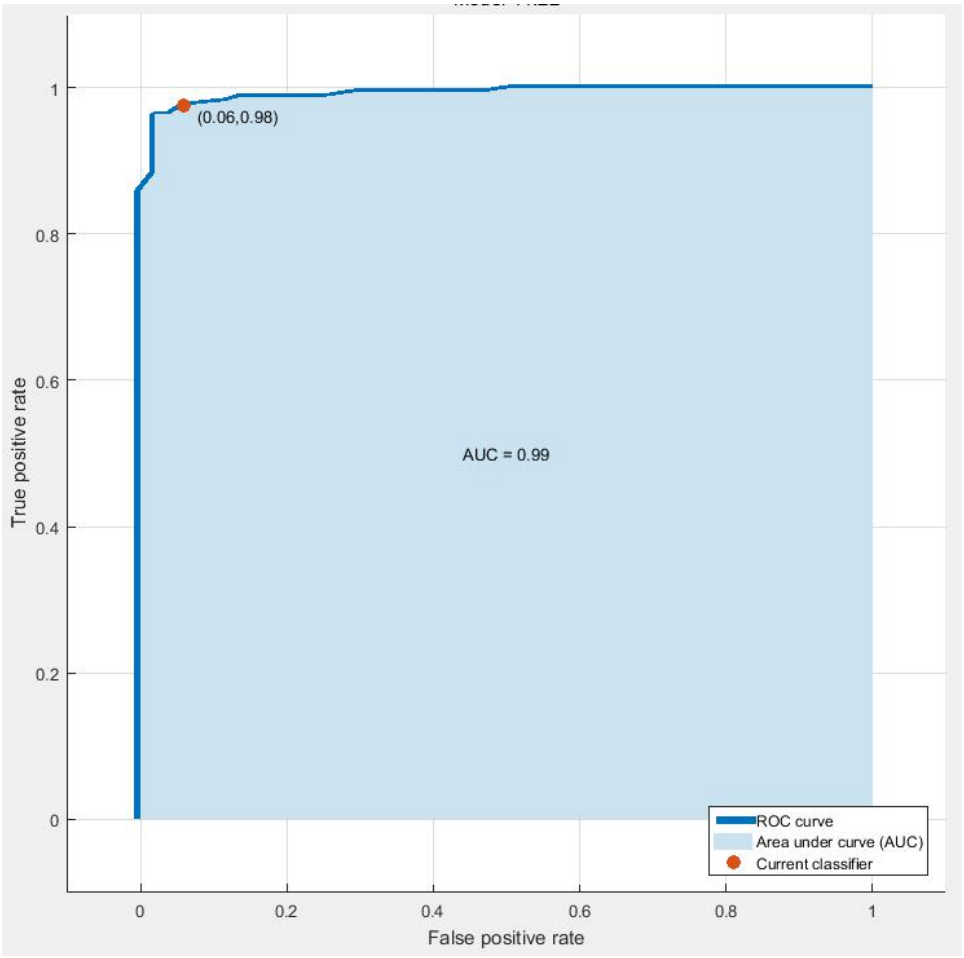
图十五 模型四决策树结构图

5.5.2 模型四的检验

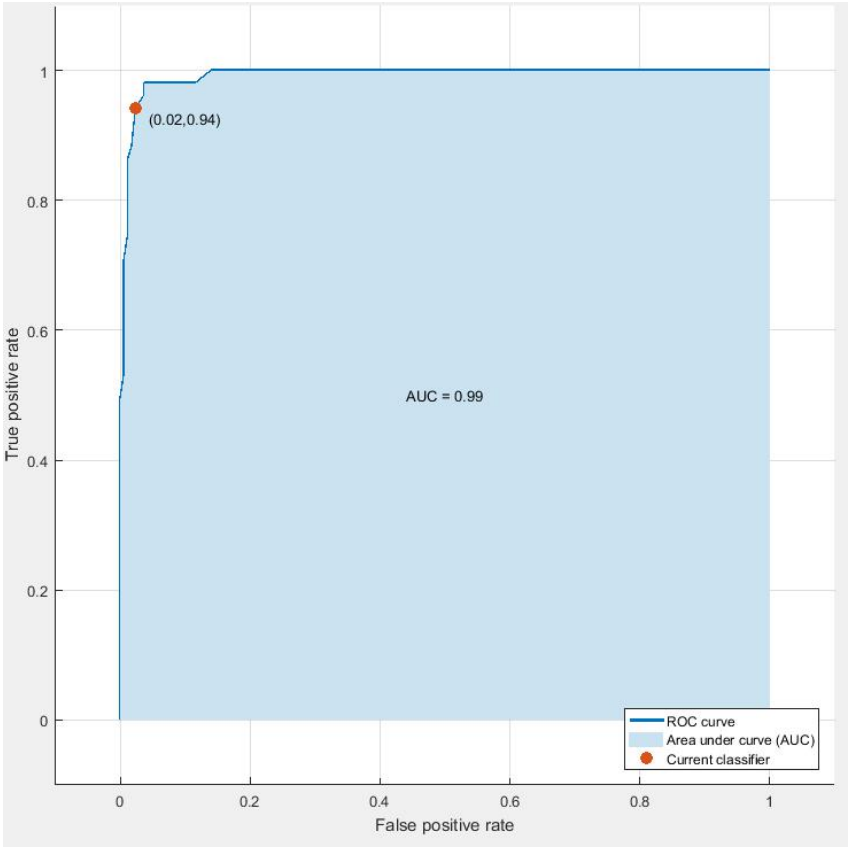
图十六 模型四的识别结果混淆矩阵（Confusion Matrix）



图十七 模型四的识别率混淆矩阵（Confusion Matrix）



图十八 模型四的第一类识别结果 ROC 图



图十九 模型四的第二类识别结果 ROC 图

观察模型四混淆矩阵，我们可以发现 Bagged Trees 模型识别率较好，与模型三 BP 神经网络的分类效果接近，具体原因为二分类问题相对于多分类问题比较好处理以及训练。一些基本模型均能达到较高的识别效果。

除了模型三以及模型四外，我们同样用 Matlab 模式识别工具箱尝试其他常用分类器的识别效果，从而选择最合适的模型作为最后的分类器。几类常用的分类器对于玻璃分类问题的识别效果如下：

表三 其他分类器对于玻璃二分类的分类效果

分类器名称	识别率
Subspace KNN	96.7%
Complex Tree（Decision Tree）	91.1%
Cubic SVM	95.3%
Logistics Regression	92.5%
Fine KNN	93.9%

这里特别提到 Subspace KNN 算法，Subspace KNN 分类器是另外一类由 Matlab 模式识别工具箱提供的集成学习分类器。与模型二 Bagging Trees 相似，Bagging 是随机使用部分训练数据，而 Random subspace method 是随机使用部分特征。它对于本问题玻璃种类识别具有最高的识别率，但是这里将不把它作为最后分类器的原因在于，Subspace KNN 常用于特征数量远远大于样本个数的情况，虽然优点也是占用空间复杂度比较低，但是对于本问题是不具有优势的。我们将采用比较简单的算法而不采用复杂的，优先选择可以解决问题的经典算法。

对于其他分类器，我们观察表三可以知道，对应的识别率均不如模型三以及模型四。因此，对于玻璃种类的二分类问题我们将从模型三 BP 神经网络以及模型四 Bagged Trees 中选择最优者。

5.6 问题一总结

由于玻璃的类别可由人为定义，若我们需要把每一种玻璃均识别出来的话就是一个多分类问题；若我们只对两种玻璃的父类感兴趣，可以把数据集划分为两类，从而将分类问题定义为二分类问题。

首先根据识别率来看，对于多分类问题来说，集成学习器 Bagged Tree 的分类准确率明显优于 BP 神经网络，因此我们可以选择模型二作为问题多分类的分类器；对于二分类问题，虽然 Bagged Trees 识别率依旧高 BP 神经网络些许，但是这里我们选择 BP 神经网络作为玻璃分类问题的分类器。原因在于，BP 神经网络是一个个体学习器，学习成本相对于具有五十个个体学习器的集成学习算法来说低了数十倍。其次，对于二分类问题，模型三和模型四的识别率差别不算特别大，对于同样高质量的分类器，我们当然选择实现条件比较简单的。

综上所述，对于问题一，我们对于不同问题的定义，分别挑选了模型一以及模型四作为我们的分类器。

六、问题二的分析

根据分类的结果，我们可以发现九个特征对于玻璃分类都具有一定的贡献率，要衡量这个贡献度，在本课程设计中，利用 ReliefF algorithm 计算各个特征的重要程度，具体代码如下：

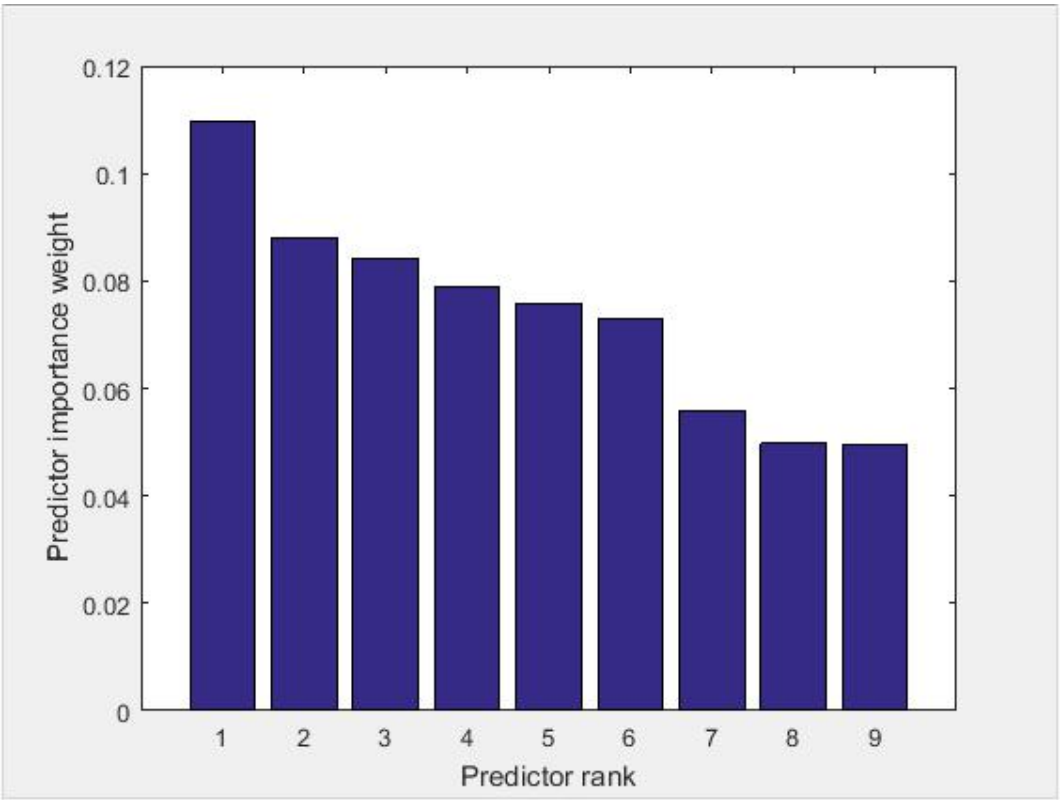
```
[ranked,weights] = relieff(input,output,10);

bar(weights(ranked));

xlabel('Predictor rank');

ylabel('Predictor importance weight');
```

得到结果如下图二十：



图二十 特征分类重要程度排序

由此，我们可见特征一 RI 对于模型分类有着最重要的作用，对应的实际意义是反射率。对于工业制作上，反射率是最能分辨玻璃种类的，因此，工厂在生产玻璃的时候一定要注意对这些关键属性的成分含量调整，使得生产出来的玻璃能满足消费者的需求。

七、模型评价

7.1 模型优点

- 模型具有一定泛化能力，对真实数据的预测分类准确率较高；
- 实现环境为 Matlab 模式识别工具箱，使用十分便利；
- 选择模型有与其他常用分类模型对比识别率，具有一定说服力；
- Bagging Trees 模型结合随机森林算法，将多个随机决策树分类结果结合得到较优分类准确率。

7.2 模型缺点

- 选择集成学习器可能会遇到过拟合的情况，具有一定风险；
- 集成学习器的学习结果并不能很好预测，波动较大，具有一定随机性。

八、模型推广与改进

8.1 主成分分析

对于六维的玻璃输入向量，是否需要用主成分分析，即 PCA（Principal Component Analysis）降维是需要考虑的对模型改进的问题。PCA 是机器学习中常用的降维算法，这里只简要提及其算法原理，不做深入探讨。

输入：样本集$\mathrm{D}=\{x_1,x_2,...,x_m\}$
　　低维空间维数d'
过程：
1：对所有样本进行中心化：$x_i \leftarrow x_i-\frac{1}{m}\sum_{i=1}^{m}x_i$;
2:计算样本的协方差矩阵XX^T;
3:对协方差矩阵XX^T做特征值分解；
4:取最大的d'个特征值所对应的特征向量$w_1,w_2,...,w_{d'}$.
输出：投影矩阵W=（$w_1,w_2,...,w_{d'}$）.

图二十一 PCA 算法

此处利用 Matlab 模式识别工具箱，对模型一到模型四分别使用一次 PCA 后再进行模型训练，最后观察其模型分类的准确率。具体结果如下：

对于 9 个输入特征，利用 PCA 降维后(要求达到 95%的解释变量比例)，得到六个个主成分，相应的的解释变量百分比分别为：45.4%, 18.0%, 12.6%, 9.8%, 6.9%, 4.2%, 2.6%, 0.4%, 0.0%。模型一到模型四利用降维后的主成分进行训练，得到模型最后的识别率并无明显改变，反而下降了些许，分别为 **70.6%**、**69.6%**、**91.4%** 以及 **94.9%**。

因此，主成分分析对于玻璃分类问题的效果的影响不大，分类器没有必要使用 PCA 进行降维。分析其原因，问题中各个特征本身就是相互独立的，并且对于问题来说本身变量个数不多，每个变量对于分类均有一定的贡献度。

8.2 模型的推广

通过以上对问题一的分析以及求解，我们不难发现这是一个简单的分类问题，因此，我们所建立的四个模型均对其他简单分类问题有一定适用性，也具有一定的指导作用。

对于四个模型与其他常用分类模型对于此问题识别率的比较，我们也可以对其他模型有一个大致的比较，分析出本课程设计的模型以及其他模型对于特定分类问题的适用条件，以及如何发挥模型的最大效用。

参 考 文 献

[1]陈雯柏.人工神经网络原理与实践[M]. 西安电子科技大学出版社,2016:56-57.

[2]房少梅.数学建模理论、方法及应用[M]. 科学出版社,2014.

[3]周志华.机器学习[M]. 清华大学出版社,2016.

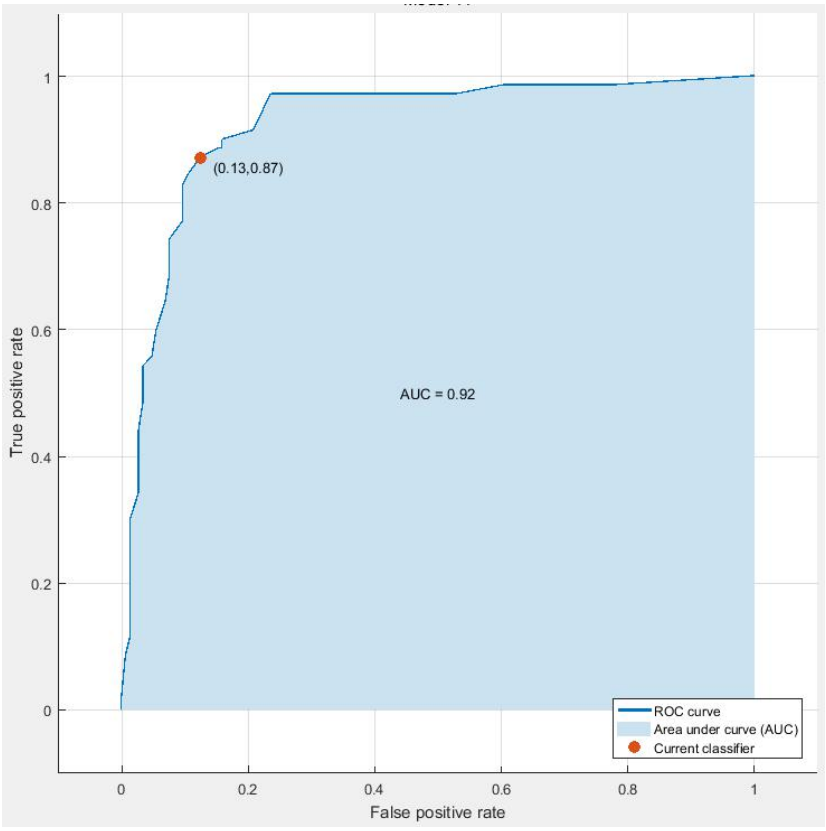
[4]Hagan, M.T., H.B. Demuth, and M.H. Beale, Neural Network Design, Boston, MA: PWS Publishing, 1996.

[5]Breiman, L. Random Forests. Machine Learning 45, pp. 5–32, 2001.

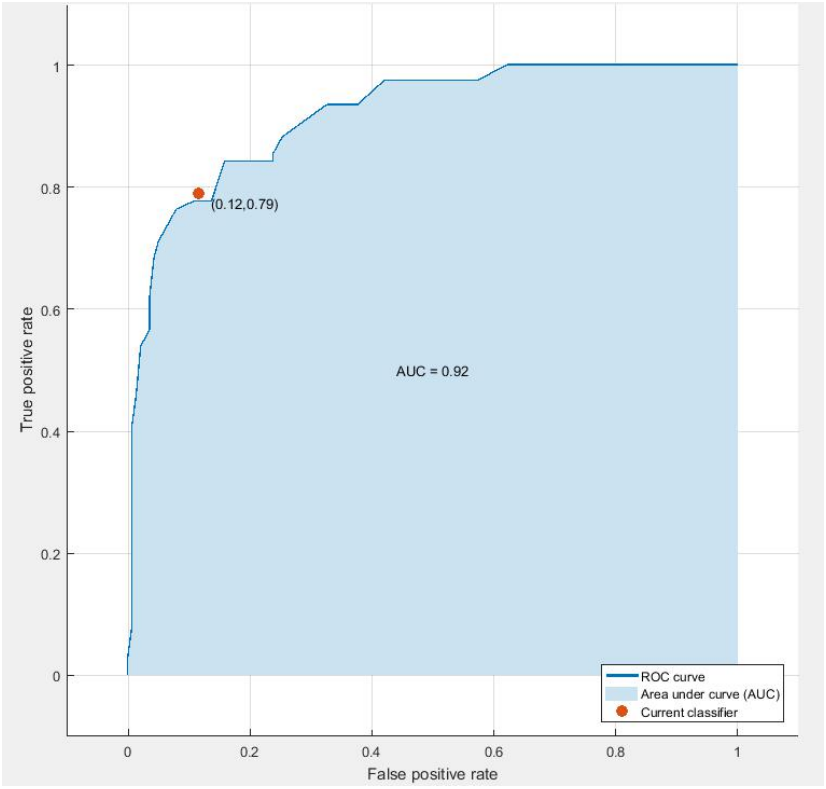
[6]Dietterich, T.G.(2000). ‘‘Ensemble methods in machine learning.’’ *Proceedings of the 1st International Workshop on multiple classifier Systems(MCS)*,1-15,Cagliari,Italy

[7]Ho, T. K. *The random subspace method for constructing decision forests.* IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 20, No. 8, pp. 832–844, 1998.

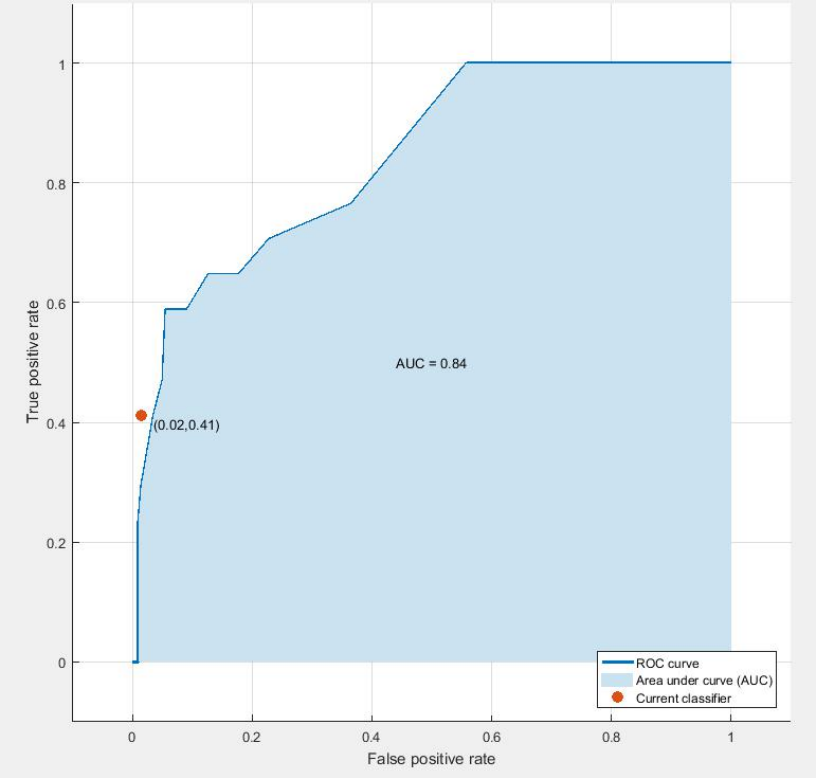
附　录



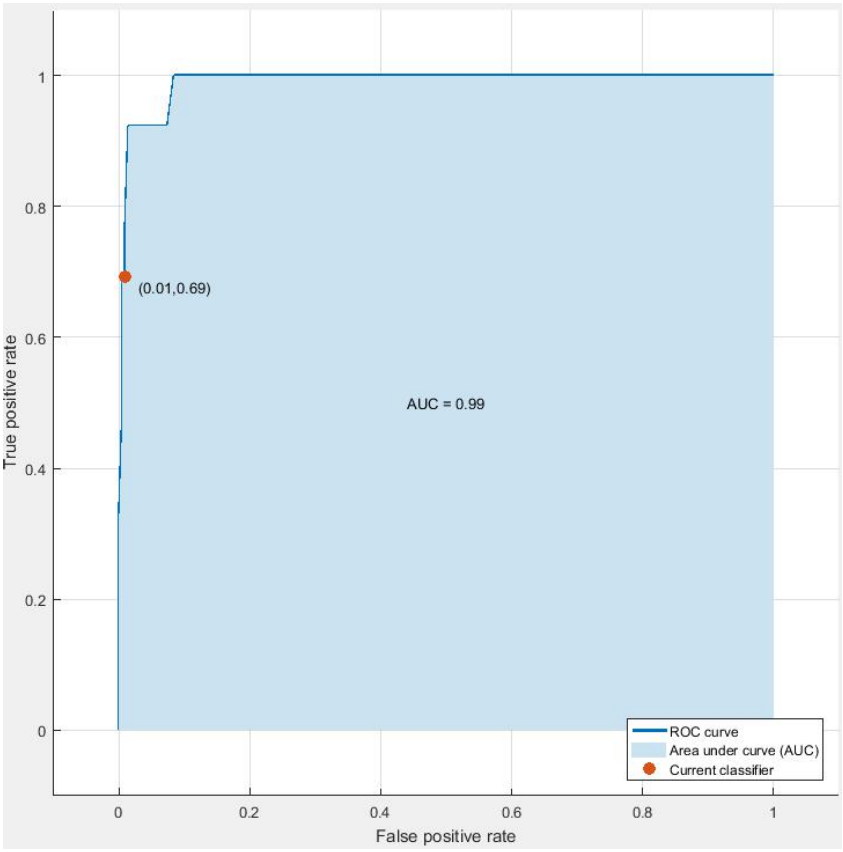
附图一　模型二中第一类玻璃ROC图



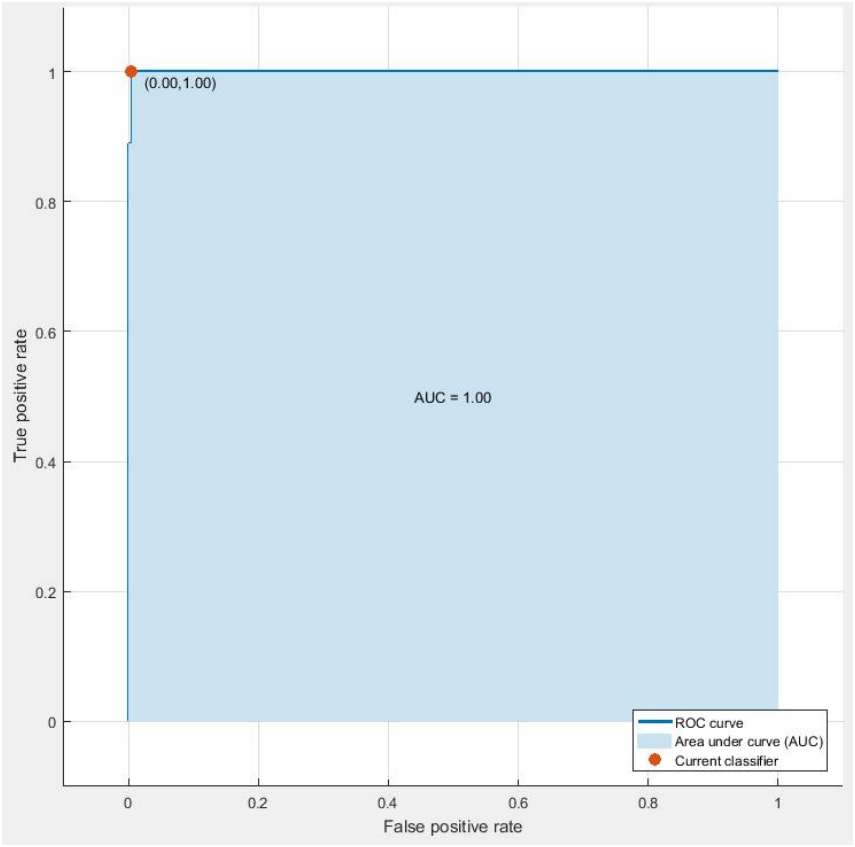
附图二　模型二中第二类玻璃ROC图



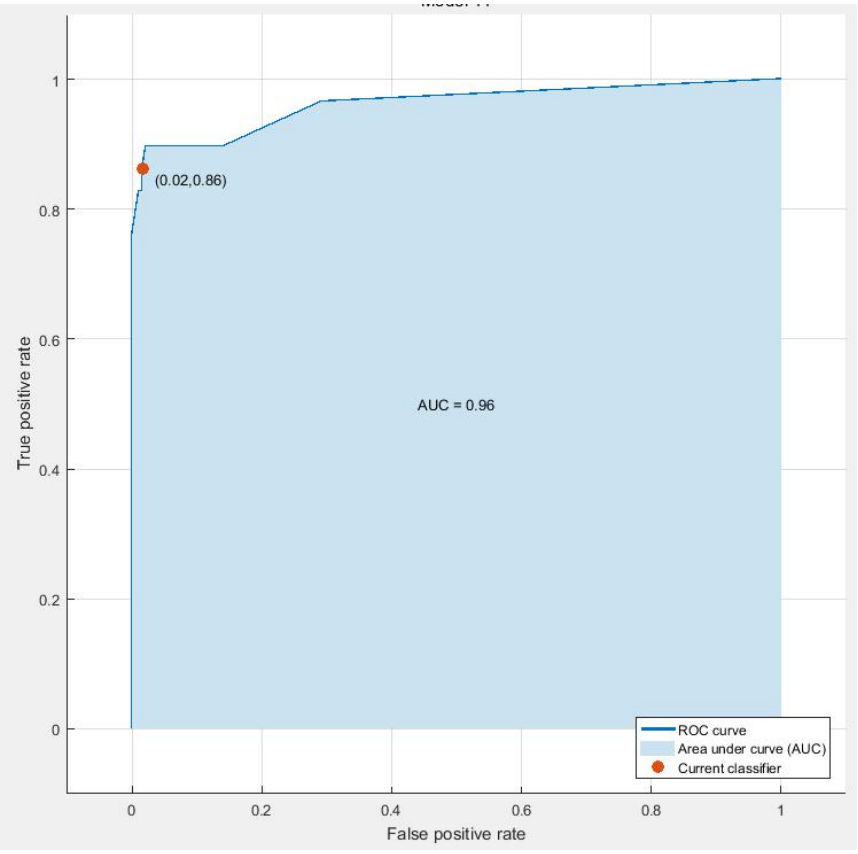
附图三 模型二中第三类玻璃ROC图



附图四 模型二中第五类玻璃ROC图



附图五 模型二中第六类玻璃ROC图



附图六 模型二中第七类玻璃ROC图

模型一训练代码Figure1Train.m

```
% Solve a Pattern Recognition Problem with a Neural Network
% Script generated by Neural Pattern Recognition app
% Created 07-Jun-2017 16:25:40
% This script assumes these variables are defined:
%   input - input data.
%   output - target data.

x = input';
t = output';


% 选择训练算法

% Scaled conjugate gradient backpropagation.
trainFcn = 'trainscg';


%选择训练方法 Scaled conjugate gradient backpropagation.

trainFcn = 'trainscg';

% 创建网络

hiddenLayerSize = 7;
net = patternnet(hiddenLayerSize);

% 划分训练集、验证集、测试集

net.divideParam.trainRatio = 70/100;
net.divideParam.valRatio = 15/100;
net.divideParam.testRatio = 15/100;

% 训练网络

[net,tr] = train(net,x,t);

% 测试网络

y = net(x);
e = gsubtract(t,y);
performance = perform(net,t,y)
tind = vec2ind(t);
yind = vec2ind(y);
percentErrors = sum(tind ~= yind)/numel(tind);

% 输出网络

view(net)
```

模型一的模型参数Figure1data.m

（用于仿真预测）

```
function [y1] = Figure1data(x1)

% Generated by Neural Network Toolbox function genFunction,
07-Jun-2017 16:24:01.
%
% [y1] = myNeuralNetworkFunction(x1) takes these arguments:
%   x = Qx9 matrix, input #1
% and returns:
%   y = Qx6 matrix, output #1
% where Q is the number of samples.

% ===== NEURAL NETWORK CONSTANTS =====

%网络权值矩阵与阈值参数数值

% Input 1
x1_step1.xoffset = [0;0;0;0;0;0;0;0;0];
x1_step1.gain = [2;2;2;2;2;2;2;2;2];
x1_step1.ymin = -1;

% Layer 1
b1 =
[-1.5008964871806751;1.3652886767606376;-0.92790633643051
168;-0.046708610745341494;-0.92572609842045095;0.82274606
261435546;1.6792673806530138];
IW1_1 = [1.3682540847806151 -0.30617548572385306
-1.4406877954299404 -0.40918236764029603
-1.1454153026535805 -0.1301240182494397 1.3215637187115326
-0.089685386431693256
-0.034675006072300368;-0.4470600330672429
1.2278952955951463 1.4680886994853664 0.74024752117264003
-0.1917652926238535 -0.58748708767228586
-0.69836190501376094 2.3474822401691298
-0.26171950385995674;0.8036945768852819 2.3098797519370864
-0.85950048341770435 -1.3755445431374071
1.5996324038506227 -0.60395439405999307
-1.4020358086535059 0.79446629218509446
-1.2490658420143912;-0.60277180003505737
-0.62826133347915725 1.8556995237309883
-3.3278943449072789 0.024287342500169059
1.1337021538405054 0.90907349807913018
```

```
-0.11172861145949825
0.4665539529320733;-1.8295820338831761
-0.27698307950991063 -0.28892175057611907
-1.0667446390094197 0.63362584360842733
0.68131178389780656 -0.22653874729764206
-2.6243929483475648 1.8023933862229293;0.83532615472477589
0.66689772644735368 2.0148566949434739 -2.2363262707454519
-0.39077628920075941 -0.21948857507906772
1.6012040169880339 -0.37283514706088794
0.44503267745777975;0.9046231893252904
-0.28292928149004581 1.4154375138593678
-1.2928048032344461 -1.7385412675046414
-0.31956021732244477 -0.22019953347773957
-0.69795598986505292 0.916544883589183511;

% Layer 2
b2 =
[1.6311705183622134;-0.046109287357436607;-0.042057650575
814225;-0.13256442546560077;1.2978775342150455;0.03612137
8989629783];
LW2_1 = [-0.86145002910570456 0.47372314409636768
2.8485803240879872 0.68735500170279329 -2.716566932469346
-1.20366540681459 -1.3042554604772199;-0.9861837918302333
0.11753420125081063 2.1482834309242054 -1.3814702308632867
0.63341319083783953 -0.21118649752181387
-0.23009271842850068;0.023866084186931376
-2.0889238599584838 -2.3762060861331769
-1.6561261254073159 0.086039345710077972
-2.4674175557734559
0.50110464798066601;0.73906391099048396 1.3399105994014011
-0.5764243968648487 1.72645879782937 -0.22806111407885196
1.1452869994847787 0.48167468398973323;2.4186926909852544
1.5808874018983046 -1.5010670994432476
-0.83028156159846356 1.5672474596172921 1.0501334828610969
2.3413111974696625;-0.051597378760939332
0.48757546967093085 -0.46936483568610904
2.6971108715164256 -0.31254140712808487 1.8268073653965025
0.018850822218335575];

% ===== SIMULATION ========

% Dimensions
Q = size(x1,1); % samples
```

```
% Input 1
x1 = x1';
xp1 = mapminmax_apply(x1,x1_step1);

% Layer 1
a1 = tansig_apply(repmat(b1,1,Q) + IW1_1*xp1);

% Layer 2
a2 = softmax_apply(repmat(b2,1,Q) + LW2_1*a1);

% Output 1
y1 = a2;
y1 = y1';
end

% ===== MODULE FUNCTIONS ========
% Map Minimum and Maximum Input Processing Function
function y = mapminmax_apply(x,settings)
y = bsxfun(@minus,x,settings.xoffset);
y = bsxfun(@times,y,settings.gain);
y = bsxfun(@plus,y,settings.ymin);
end

% Competitive Soft Transfer Function
function a = softmax_apply(n,~)
if isa(n,'gpuArray')
    a = iSoftmaxApplyGPU(n);
else
    a = iSoftmaxApplyCPU(n);
end
end
function a = iSoftmaxApplyCPU(n)
nmax = max(n,[],1);
n = bsxfun(@minus,n,nmax);
numerator = exp(n);
denominator = sum(numerator,1);
denominator(denominator == 0) = 1;
a = bsxfun(@rdivide,numerator,denominator);
end
function a = iSoftmaxApplyGPU(n)
nmax = max(n,[],1);
numerator = arrayfun(@iSoftmaxApplyGPUHelper1,n,nmax);
denominator = sum(numerator,1);
a =
```

```
arrayfun(@iSoftmaxApplyGPUHelper2,numerator,denominator);
end
function numerator = iSoftmaxApplyGPUHelper1(n,nmax)
numerator = exp(n - nmax);
end
function a = iSoftmaxApplyGPUHelper2(numerator,denominator)
if (denominator == 0)
    a = numerator;
else
    a = numerator ./ denominator;
end
end

% Sigmoid Symmetric Transfer Function
function a = tansig_apply(n,~)
a = 2 ./ (1 + exp(-2*n)) - 1;
End
```

模型二训练代码Figure2_TrainNewData.m

```
function [trainedClassifier, validationAccuracy] =
Figure2_TrainNewData(trainingData)

inputTable = array2table(trainingData, 'VariableNames',
{'column_1','column_2','column_3','column_4','column_5',
'column_6', 'column_7', 'column_8', 'column_9',
'column_10'});

predictorNames = {'column_1', 'column_2', 'column_3',
'column_4', 'column_5', 'column_6', 'column_7', 'column_8',
'column_9'};
predictors = inputTable(:, predictorNames);
response = inputTable.column_10;
isCategoricalPredictor = [false, false, false, false, false,
false, false, false, false, false];

% Train a classifier
% This code specifies all the classifier options and trains
the classifier.
classificationEnsemble = fitensemble(...
    predictors, ...
    response, ...
    'Bag', ...
    30, ...
    'Tree', ...
```

```
    'Type', 'Classification', ...
    'ClassNames', [1; 2; 3; 5; 6; 7]);
% Create the result struct with predict function
predictorExtractionFcn = @(x) array2table(x, 'VariableNames',
predictorNames);
ensemblePredictFcn = @(x) predict(classificationEnsemble,
x);
trainedClassifier.predictFcn = @(x)
ensemblePredictFcn(predictorExtractionFcn(x));

% Add additional fields to the result struct
trainedClassifier.ClassificationEnsemble =
classificationEnsemble;
trainedClassifier.About = 'This struct is a trained
classifier exported from Classification Learner R2016a.';
trainedClassifier.HowToPredict = sprintf('To make
predictions on a new predictor column matrix, X, use: \n  yfit
= c.predictFcn(X) \nreplacing ''c'' with the name of the
variable that is this struct, e.g. ''trainedClassifier''. \n
\nX must contain exactly 9 columns because this classifier
was trained using 9 predictors. \nX must contain only
predictor columns in exactly the same order and format as your
training \ndata. Do not include the response column or any
columns you did not import into \nClassification Learner. \n
\nFor more information, see <a
href="matlab:helpview(fullfile(docroot, ''stats'',
''stats.map''),
''appclassification_exportmodeltoworkspace'')">How to
predict using an exported model</a>.');

% Extract predictors and response
% This code processes the data into the right shape for training
the
% classifier.
% Convert input to table
inputTable = array2table(trainingData, 'VariableNames',
{'column_1', 'column_2', 'column_3', 'column_4', 'column_5',
'column_6', 'column_7', 'column_8', 'column_9',
'column_10'});

predictorNames = {'column_1', 'column_2', 'column_3',
'column_4', 'column_5', 'column_6', 'column_7', 'column_8',
'column_9'};
predictors = inputTable(:, predictorNames);
```

```
response = inputTable.column_10;
isCategoricalPredictor = [false, false, false, false, false,
false, false, false, false];

% Perform cross-validation
partitionedModel =
crossval(trainedClassifier.ClassificationEnsemble, 'KFold',
5);

% Compute validation accuracy
validationAccuracy = 1 - kfoldLoss(partitionedModel,
'LossFun', 'ClassifError');

% Compute validation predictions and scores
[validationPredictions, validationScores] =
kfoldPredict(partitionedModel);
```

模型三的训练代码Model3Train.m

```
x = glassInputs;
t = glassTargets;

% Scaled conjugate gradient backpropagation.
trainFcn = 'trainscg';
% Create a Pattern Recognition Network
hiddenLayerSize = 4;
net = patternnet(hiddenLayerSize);

% Setup Division of Data for Training, Validation, Testing
net.divideParam.trainRatio = 70/100;
net.divideParam.valRatio = 15/100;
net.divideParam.testRatio = 15/100;

% Train the Network
[net,tr] = train(net,x,t);
% Test the Network
y = net(x);
e = gsubtract(t,y);
performance = perform(net,t,y)
tind = vec2ind(t);
yind = vec2ind(y);
percentErrors = sum(tind ~= yind)/numel(tind);

% View the Network
view(net)
```

模型三的模型参数Figure3myNeuralNetworkFunction.m

（用于仿真预测）

```
function [Y,Xf,Af] = Figure3myNeuralNetworkFunction(X,~,~)
% Input 1
x1_step1.xoffset =
[1.51115;10.73;0;0.29;69.81;0;5.43;0;0];
x1_step1.gain =
[87.7963125548726;0.300751879699248;0.44543429844098;0.62
3052959501558;0.357142857142857;0.322061191626409;0.18587
3605947955;0.634920634920635;3.92156862745098];
x1_step1.ymin = -1;

% Layer 1
b1 =
[-1.4505196582398243;0.56076261917120307;-0.2624042866016
1036;-1.6290993083426839];
IW1_1 = [0.52417136723188107 0.86684851171060762
0.22280577189131001 0.69357501924207765
0.37489214618898153 -0.90659225091563089
-0.31991292879152483 0.2171164372789616
-0.89762956350184342;-0.20868743233442755
-0.70843633208958379 0.30926993678202347
-0.88278274503087206 -0.59263434046342744
-0.54708723743812449 1.1457006709037629
-0.098048087569543363
1.0143489586855843;1.0830926993167895 -0.88272747755811576
2.9497748061254834 -1.4345799630328264 0.15083135583799587
0.027687956708200748 0.062782351192921221
0.3327571176288277
0.43876635277204684;-0.46670331360121703
0.66673189398641808 0.55896462588936535
0.46201568263753745 0.67815443648943685
0.48694654913643826 0.70528340761833419
-0.11990319113043701 0.51560650306399436];

% Layer 2
b2 = [0.40764427075081999;0.79906790133227712];
LW2_1 = [0.87309249679613177 -1.0752296872322873
-1.4088038648599595
-0.33736800409751772;-0.11779741014421044
2.0408919812667805 2.048895881396898
-0.67504991019083305];
```

```
% ===== SIMULATION ========

% Format Input Arguments
isCellX = iscell(X);
if ~isCellX, X = {X}; end;

% Dimensions
TS = size(X,2); % timesteps
if ~isempty(X)
    Q = size(X{1},2); % samples/series
else
    Q = 0;
end

% Allocate Outputs
Y = cell(1,TS);

% Time loop
for ts=1:TS
    % Input 1
    Xp1 = mapminmax_apply(X{1,ts},x1_step1);

    % Layer 1
    a1 = tansig_apply(repmat(b1,1,Q) + IW1_1*Xp1);

    % Layer 2
    a2 = softmax_apply(repmat(b2,1,Q) + LW2_1*a1);

    % Output 1
    Y{1,ts} = a2;
end

% Final Delay States
Xf = cell(1,0);
Af = cell(2,0);

% Format Output Arguments
if ~isCellX, Y = cell2mat(Y); end
end

% ===== MODULE FUNCTIONS ========

% Map Minimum and Maximum Input Processing Function
```

```
function y = mapminmax_apply(x,settings)
y = bsxfun(@minus,x,settings.xoffset);
y = bsxfun(@times,y,settings.gain);
y = bsxfun(@plus,y,settings.ymin);
end

% Competitive Soft Transfer Function
function a = softmax_apply(n,~)
if isa(n,'gpuArray')
    a = iSoftmaxApplyGPU(n);
else
    a = iSoftmaxApplyCPU(n);
end
end
function a = iSoftmaxApplyCPU(n)
nmax = max(n,[],1);
n = bsxfun(@minus,n,nmax);
numerator = exp(n);
denominator = sum(numerator,1);
denominator(denominator == 0) = 1;
a = bsxfun(@rdivide,numerator,denominator);
end
function a = iSoftmaxApplyGPU(n)
nmax = max(n,[],1);
numerator = arrayfun(@iSoftmaxApplyGPUHelper1,n,nmax);
denominator = sum(numerator,1);
a =
arrayfun(@iSoftmaxApplyGPUHelper2,numerator,denominator);
end
function numerator = iSoftmaxApplyGPUHelper1(n,nmax)
numerator = exp(n - nmax);
end
function a = iSoftmaxApplyGPUHelper2(numerator,denominator)
if (denominator == 0)
    a = numerator;
else
    a = numerator ./ denominator;
end
end

% Sigmoid Symmetric Transfer Function
function a = tansig_apply(n,~)
a = 2 ./ (1 + exp(-2*n)) - 1;
end
```

模型四训练代码（Model4Train.m）

```
function [trainedClassifier, validationAccuracy] =
trainClassifier(trainingData)

inputTable = array2table(trainingData, 'VariableNames',
{'column_1','column_2','column_3','column_4','column_5',
'column_6', 'column_7', 'column_8', 'column_9',
'column_10'});

predictorNames = {'column_1', 'column_2', 'column_3',
'column_4', 'column_5', 'column_6', 'column_7', 'column_8',
'column_9'};
predictors = inputTable(:, predictorNames);
response = inputTable.column_10;
isCategoricalPredictor = [false, false, false, false, false,
false, false, false, false, false];

% Train a classifier
% This code specifies all the classifier options and trains
the classifier.
classificationEnsemble = fitensemble(...
    predictors, ...
    response, ...
    'Bag', ...
    50, ...
    'Tree', ...
    'Type', 'Classification', ...
    'ClassNames', [1; 2]);

% Create the result struct with predict function
predictorExtractionFcn = @(x) array2table(x, 'VariableNames',
predictorNames);
ensemblePredictFcn = @(x) predict(classificationEnsemble,
x);
trainedClassifier.predictFcn = @(x)
ensemblePredictFcn(predictorExtractionFcn(x));

% Add additional fields to the result struct
trainedClassifier.ClassificationEnsemble =
classificationEnsemble;
trainedClassifier.About = 'This struct is a trained
classifier exported from Classification Learner R2016a.';
trainedClassifier.HowToPredict = sprintf('To make
```

```
predictions on a new predictor column matrix, X, use: \n  yfit
= c.predictFcn(X) \nreplacing ''c'' with the name of the
variable that is this struct, e.g. ''trainedClassifier''. \n
\nX must contain exactly 9 columns because this classifier
was trained using 9 predictors. \nX must contain only
predictor columns in exactly the same order and format as your
training \ndata. Do not include the response column or any
columns you did not import into \nClassification Learner. \n
\nFor more information, see <a
href="matlab:helpview(fullfile(docroot, ''stats'',
''stats.map''),
''appclassification_exportmodeltoworkspace'')">How to
predict using an exported model</a>.');

% Extract predictors and response
% This code processes the data into the right shape for training
the
% classifier.
% Convert input to table
inputTable = array2table(trainingData, 'VariableNames',
{'column_1', 'column_2', 'column_3', 'column_4', 'column_5',
'column_6', 'column_7', 'column_8', 'column_9',
'column_10'});

predictorNames = {'column_1', 'column_2', 'column_3',
'column_4', 'column_5', 'column_6', 'column_7', 'column_8',
'column_9'};
predictors = inputTable(:, predictorNames);
response = inputTable.column_10;
isCategoricalPredictor = [false, false, false, false, false,
false, false, false, false];

% Perform cross-validation
partitionedModel =
crossval(trainedClassifier.ClassificationEnsemble, 'KFold',
5);

% Compute validation accuracy
validationAccuracy = 1 - kfoldLoss(partitionedModel,
'LossFun', 'ClassifError');

% Compute validation predictions and scores
[validationPredictions, validationScores] =
kfoldPredict(partitionedModel);
```