南京中医药大学



**本科生毕业论文**

信息技术学院计算机科学与技术专业15年级

**学 号** 084015125

**学生姓名** 赵 壮

**论文题目**中医藏象辨证量化诊断

系统的设计与实现

**实习单位** 南京中医药大学

**指导教师** 杨 涛

**起止时间** 2019-2-20～2019-6-10

**2019年 5 月 7 日**

摘 要

目的：本设计旨在深入研究深度学习、集成学习等机器学习理论，并应用相关算法开展中医藏象辨证量化诊断的创新研究和应用，在设计和实现AdaBoost、随机森林、卷积神经网络和谱聚类等应用广泛、表现优良的机器学习算法的基础上，搭建以算法为核心、集成数据采集、数据清洗、量化诊断、算法评估等功能模块的中医藏象辨证量化诊断的一体化平台。

方法：首先利用学校的中医院校资源优势，收集到来自中医院的7964条原始医案数据；其次编写脚本对医案进行基本清洗，再组织中医专业学生对症状、舌象、脉象和证型等数据项进行规范和标准化，再利用整理出来的数据项字典进行批量替换得到最终标准化医案样本；针对AdaBoost、随机森林、卷积神经网络和谱聚类等算法建立相应的模型，将样本处理成规范的特征向量输入到模型参与计算，调整模型参数，应用表现最优的参数在测试样本上评估；利用Flask开发框架、MySQL数据库和Echarts.js等组件，集成最优算法模型，基于MVC的开发模式以及面向对象的开发方法，实现中医藏象辨证量化诊断系统。

结果：基本清洗得到7518条有效医案，已规范700条医案，提取得到1871个症状特征、32个舌象特征、16个脉象特征以及50个证型标签；基于以上清洗结果，AdaBoost、随机森林、卷积神经网络和谱聚类等四种模型的以准确率为指标的最优表现分别为44.62%、47.59%、52.47%和39.28%。

结论：在医案数据未得到完全标准化、样本分布不均匀、模型大小受硬件条件约束的情况下，卷积神经网络在测试样本集上表现最佳，并且随着网络模型复杂程度提高而有明显提高，集成学习算法随着参数调整，表现在40%～50%间波动，谱聚类在无监督学习上的分类表现一般。在样本质量提高的基础上，各模型表现有望得到明显提高。

关键词：量化诊断；AdaBoost；随机森林；卷积神经网络；谱聚类；一体化平台

**Abstract**

Purpose: This design aims to deeply study machine learning theories such as deep learning and integrated learning, and apply relevant algorithms to carry out innovative research and application of TCM syndrome differentiation and diagnosis, design and implement AdaBoost, random forest, convolutional neural network and spectrum. Based on clustering and other well-performing machine learning algorithms, an integrated platform for syndrome differentiation and diagnosis of TCM Tibetan elephants with algorithmic core, integrated data acquisition, data cleaning, quantitative diagnosis, algorithm evaluation and other functional modules is built.

Methods: Firstly, 7946 original medical records from Chinese medicine hospitals were collected by using the resources of the Chinese medicine hospitals in the school. Secondly, scripts were used to basically clean the medical cases, and then the students of traditional Chinese medicine were organized to symptom, tongue, pulse and syndrome. The data items are standardized and standardized, and then the sorted data item dictionary is used for batch replacement to obtain the final standardized medical sample; the corresponding model is established for AdaBoost, random forest, convolutional neural network and spectral clustering algorithms, and the sample is processed. The normalized eigenvectors are input into the model to participate in the calculation, the model parameters are adjusted, and the parameters with the best performance are evaluated on the test samples. The components of the Flask development framework, MySQL database and Echarts.js are integrated to integrate the optimal algorithm model based on MVC. Development model and object-oriented development method to realize TCM syndrome differentiation and diagnosis system.

Results: Basic cleansing resulted in 7518 effective medical records, 700 medical records have been standardized, and 1871 symptom features, 32 tongue features, 16 pulse features and 50 syndrome tags were extracted. Based on the above cleaning results, AdaBoost, random The optimal performance of the four models of forest, convolutional neural network and spectral clustering were 44.62%, 47.59%, 52.47% and 39.28%, respectively.

Conclusion: The convolutional neural network performs best on the test sample set when the medical record data is not fully standardized, the sample distribution is not uniform, and the model size is constrained by hardware conditions, and it is obvious as the complexity of the network model increases. Improve, the integrated learning algorithm fluctuates between 40% and 50% with parameter adjustment, and the spectral clustering performance in unsupervised learning is general. On the basis of the improvement of sample quality, the performance of each model is expected to be significantly improved.

**Key words:** Quantitative diagnosis; AdaBoost; random forest; convolutional neural network; spectral clustering; integrated platform

目 录

[1.绪论 1](#_Toc9064190)

[1.1项目背景 1](#_Toc9064191)

[1.2目的与意义 1](#_Toc9064192)

[1.3任务概述 2](#_Toc9064193)

[1.3.1设计目标 2](#_Toc9064194)

[1.3.2算法要求 2](#_Toc9064195)

[1.3.3平台特点 2](#_Toc9064196)

[2.相关技术简介 4](#_Toc9064197)

[2.1相关算法 4](#_Toc9064198)

[2.1.1 AdaBoost 4](#_Toc9064199)

[2.1.2随机森林 4](#_Toc9064200)

[2.1.3卷积神经网络 5](#_Toc9064201)

[2.1.4谱聚类 5](#_Toc9064202)

[2.2开发技术 5](#_Toc9064203)

[2.2.1前端技术——HTML+CSS+JavaScript 5](#_Toc9064204)

[2.2.2后端技术——MySQL+Flask 6](#_Toc9064205)

[2.2.3算法框架——TensorFlow+Scikit-learn 6](#_Toc9064206)

[3.中医藏象辨证量化诊断方法研究 8](#_Toc9064207)

[3.1数据准备 8](#_Toc9064208)

[3.1.1医案清洗 8](#_Toc9064209)

[3.1.2医案标准化 9](#_Toc9064210)

[3.2 AdaBoost算法建模 10](#_Toc9064211)

[3.2.1构造特征向量 10](#_Toc9064212)

[3.2.2构建决策树 11](#_Toc9064213)

[3.2.3 Boosting集成学习 12](#_Toc9064214)

[3.2.4算法评估 13](#_Toc9064215)

[3.3随机森林算法建模 15](#_Toc9064216)

[3.3.1构造特征向量 15](#_Toc9064217)

[3.3.2构建决策树 15](#_Toc9064218)

[3.3.3 Bagging集成学习 15](#_Toc9064219)

[3.3.4算法评估 16](#_Toc9064220)

[3.4卷积神经网络建模 17](#_Toc9064221)

[3.4.1构造特征向量 19](#_Toc9064222)

[3.4.2设计网络模型 19](#_Toc9064223)

[3.4.3 训练网络 21](#_Toc9064224)

[3.4.4模型评估与调参 22](#_Toc9064225)

[3.5谱聚类算法建模 23](#_Toc9064226)

[3.5.1构造特征向量 23](#_Toc9064227)

[3.5.2构建拉普拉斯矩阵 23](#_Toc9064228)

[3.5.3图切分与聚类 23](#_Toc9064229)

[3.5.4算法评估 24](#_Toc9064230)

[3.6算法总结与比较 25](#_Toc9064231)

[4.中医藏象智能诊断平台的设计与实现 26](#_Toc9064232)

[4.1系统总体架构 26](#_Toc9064233)

[4.2系统功能设计 27](#_Toc9064234)

[4.2.1医案录入 27](#_Toc9064235)

[4.2.2医案清洗 30](#_Toc9064236)

[4.2.3字典维护 33](#_Toc9064237)

[4.2.4量化诊断 34](#_Toc9064238)

[4.2.5结果展示 35](#_Toc9064239)

[4.2.6算法训练 40](#_Toc9064240)

[4.2.7算法评估 41](#_Toc9064241)

[4.3数据库设计 41](#_Toc9064242)

[4.3.1字典表 42](#_Toc9064243)

[4.3.2医案表 43](#_Toc9064244)

[4.3.3算法表 43](#_Toc9064245)

[4.3.4诊断表 44](#_Toc9064246)

[4.4系统功能展示 44](#_Toc9064247)

[4.4.1医案录入 45](#_Toc9064248)

[4.4.2医案清洗 46](#_Toc9064249)

[4.4.3字典维护 47](#_Toc9064250)

[4.4.4量化诊断 48](#_Toc9064251)

[4.4.5结果展示 49](#_Toc9064252)

[4.4.6算法训练 50](#_Toc9064253)

[4.4.7算法评估 50](#_Toc9064254)

[5.总结与展望 52](#_Toc9064255)

[5.1系统优点与特色 52](#_Toc9064256)

[5.2系统缺点与不足 52](#_Toc9064257)

[5.3系统完善思路与展望 53](#_Toc9064258)

[结束语 54](#_Toc9064259)

[致谢 55](#_Toc9064260)

[参考文献 56](#_Toc9064261)

# 1.绪论

## 1.1项目背景

中医诊断的核心内容是辨证，其指导着中医临床的治疗方法、处方用药。在我国长达数千年的中医实践中，历代医家提出了三焦辨证、六经辨证、八纲辨证、脏腑辨证和卫气营血辨证等多种辨证方法，这些方法彼此重叠、相互补充、共同发展并指导着中医临床实践。但是，临床上病情千变万化，证型异同难辨，辨证方法越来越多，辨证思维却越发混乱，这给临床辨证带来极大困惑[1]。怎样从海量的临床实践中找到辨证的规律和原理，从而为病症证的规范化提供依据和参考，这已成为当下中医辨证的研究重点和发展方向。

随着信息化在中医领域的发展以及电子病历的广泛应用，数据化的病历资料为中医辨证的临床实践经验挖掘提供了数据基础。中国中医科学院的刘保延研究员认为“只要能够将真实世界的临床实践的诊疗信息快捷、准确、全面的数据化，形成前所未有的大数据，才可能使中医辨证论治个体诊疗正常实施中所蕴含的各种创新得以科学地展现[2]”。

本设计正是顺应“以数据为导向”的中医临床科研发展趋势，以中医藏象辨证理论为中医理论指导，以机器学习领域广泛应用、表现优良的深度学习和集成学习理论为方法基础，以面向对象思想、MVC模式为开发规范，设计和实现中医藏象辨证量化诊断系统，为大数据时代中医辨证研究提供方法探索和实践参考。

## 1.2目的与意义

从学科发展的角度讲，本设计尝试利用集成学习、深度学习等机器学习算法从中医临床数据中挖掘常见证型的量化诊断规律，形成“以数据为支撑”、“以算法为驱动”的辨证量化诊断体系，为大数据时代中医辨证研究提供参考。

从临床实践的角度讲，本设计尝试利用Bootstrap.js，JqueryUI.js、Echarts.js等组件开发用户友好的中医藏象辨证量化诊断系统，支持医案的傻瓜式录入、诊断结果的直观式展示以及算法模型一键式评估，能够提高临床实践中医案录入、医案清洗、量化诊断和算法评估等操作的易用性和可用性。

从个人提升的角度讲，通过本设计的探索实践，可以学习到中医藏象辨证理论，理解机器学习算法实现与优化原理，掌握Web开发技术，熟悉各种开源组件，提高独立开发能力和整体专业水平，为进一步的科研或工作打下理论与实践基础。

## 1.3任务概述

### 1.3.1设计目标

对深度学习、集成学习等算法进行深入研究，探索利用相关算法开展中医辨证的创新研究与应用，利用相关算法实现临床数据采集、量化诊断和可视化结果展示。具体功能如下：

a) 在藏象辨证体系指导下，利用算法实现中医智能诊断，即从临床信息到中医证型的计算；

b) 利用算法对中医医案进行分析，从准确率、召回率、F1值等比较模型优劣；

c) 设计智能辨证程序，将算法集成到程序中，实现输入“症状信息”得“辨证结果”；

d) 设计两种用户交互模式，包括单个病人智能辨证和批量数据导入后的智能辨证；数据导入支持采用文本、Excel、表单和数据库等常用格式；相关结果存储到数据库中，用于历史记录的查询；

e) 历史辨证数据的可视化结果展示，包括症状体征的分布、证型的分布等，采用图表展示。

### 1.3.2算法要求

为实现中医藏象辨证量化诊断系统对算法的封装集成，考虑用户交互体验以及灵活的算法选择和扩展，算法应达到以下要求：

a) 算法性能表现良好，具有合理的时间复杂度和空间复杂度，在8G内存的个人服务器上能够在合理的响应时间内返回算法结果，避免用户过长时间等待，降低用户体验；

b) 算法准确率应具有一定的参考价值，至少应保证是随机猜测的平均准确率10倍以上，即假设有n种分类结果，在大量随机猜测条件下期望准确率应为，则算法的准确率至少应保证在以上，说明算法具有一定参考价值，值得进一步研究和提升；

c) 算法应该灵活、可扩展，即仅限定必要的模型参数和默认参数，向系统用户开放一定的调参功能，让用户选择最优的模型参数以及评估指标。

### 1.3.3平台特点

a) 从UI设计角度来说，界面风格简洁，布局美观。采用上导航栏、左菜单栏、右下功能区的区块布局，块内采用单一色调，配以提示性Logo，风格统一，简洁美观，使用户操作一目了然，轻松上手，不存在任何使用障碍；

b) 从功能使用角度来说，支持文本、表格、表单、数据库等多种方式录入医案，兼容性强；能够实现实时智能输入提示，减少用户操作，提高录入效率；支持维护用户字典，定义常用输入选项，可扩展性强；

c) 从用户体验角度来说，诊断及算法评估功能采用柱状图、折线图、饼状图、雷达图、关系网络图等图表形式展示，数据结果直观化，降低用户理解负担和专业水平要求，即便没有专业统计分析知识的一般用户也能理解本系统分析结果。

# 2.相关技术简介

## 2.1相关算法

### 2.1.1 AdaBoost

AdaBoost算法是集成学习技术中Boosting元算法框架的最成功的代表，被称为数据挖掘十大算法之一[3]。它能够将准确度仅比随机猜测准确度高一点的弱学习器增强为准确度高很多的强学习器，其几乎可以使用在目前所有流行的机器学习算法上来进一步加强原算法的准确度。

该算法的核心思想是针对同一批训练样本，使用多个相同或不同的弱分类器进行学习与评估，不同的是其采用不同的线性组合顺序，前一个分类器评估得到错分的样本会被加大权值，使得下一个分类器聚焦于之前被分错的样本，使其能够有针对性地增强对具有特定特征的样本的分类能力，进而使弱分类器全体组合而成的强学习器对全体样本表现出更可靠的分类能力。

### 2.1.2随机森林

随机森林算法是集成学习技术中Bagging元算法框架的最成功代表，其弱分类器采用同为数据挖掘十大算法的决策树，其能够明显提高作为弱学习器的决策树的分类能力。

决策树的思想本质实际上就是寻找最有效的划分方法将目标变量足够地分开，使得相同类别的样本聚集，不同类别的样本分开，不同的决策树区别在于划分依据，ID3使用信息增益，C4.5使用信息增益率，CART使用基尼系数[4]。其大体的建树思路如下图1所示：

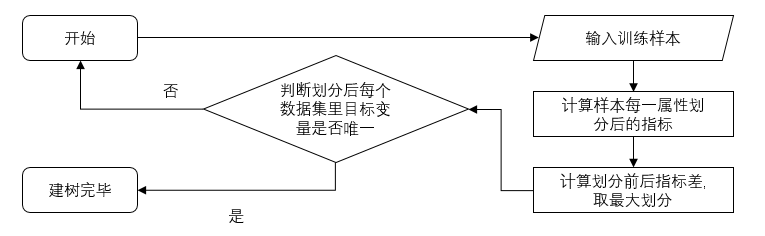


图1 决策树建树流程

而随机森林的思想是使用决策树作为弱学习器，从样本集中多次放回抽样，每次抽样都会输入到对应决策树，这样每棵决策树都能独立地建树剪枝，针对所学习的随机样本子集进行学习分类，最终组合投票得到在样本全集上表现优良的分类结果。

### 2.1.3卷积神经网络

卷积神经网络是深度学习技术中针对图像特征提取和分类的网络模型，其提供了一种端到端的学习方式。型基于传统的神经网络，可以通过梯度下降的方式对模型参数进行训练，不同的是其运用了人体视觉神经的局部性感知原理，上一层的特征可以通过权值共享的卷积核得到下一层特征，这更适合图像特征的学习与表达。[5]

卷积神经网络的基本思想是通过大量的训练样本进行数据拟合，形成多层次、不同规模的特征分布，最终特征组合得到目标类别的特征表达，在此过程中运用共享卷积核组织表达特征的层次性关系，运用池化的方式对目标类别特征进行泛化。

### 2.1.4谱聚类

谱聚类算法是无监督学习中聚类算法的优化和改进。聚类算法是将具有相似特征的样本划分到同一个类或簇当中，使得类内高度相似，类间高度区分。传统的K-means、EM聚类算法都是建立在样本空间呈凸状的条件下的，即问题的全局最优解和局部最优解是同一解，而对于非凸样本空间则无能为力，可能陷入局部最优的错误结果[6]。

而谱聚类则是针对非凸样本空间的改进，其利用谱图理论，将聚类问题转化为图的最优划分问题。其基本思想是用相似矩阵和度矩阵构建拉普拉斯矩阵，根据不同的切图方法对拉普拉斯矩阵使用相应的优化函数，计算优化函数的特征矩阵得到降维后的样本特征表达，再进行传统聚类得到最终聚类结果。

## 2.2开发技术

### 2.2.1前端技术——HTML+CSS+JavaScript

本系统基于B/S架构，以Web网页的形式呈现界面并与用户交互，前端开发技术自然主要集中在网页的编码，其中涉及到网页元素的布局，元素样式的呈现以及元素与用户的动态交互，相应的编码内容分别是HTML、CSS和JavaScript。

HTML，即超文本标记语言，是一种通用标记语言，也是一种标准和一种规范，其本身是文本，通过添加页面元素标签告诉浏览器如何显示超媒体内容，浏览器按照W3C技术标准顺序读取网页文件，根据标签解释并显示标签代表的内容，最终呈现网页的超媒体形式。

CSS，即层叠样式表，是一种用来表现HTML标签显示样式的语言，可以静态地修饰网页，也可以动态地对标签进行像素级别的精准控制，使其根据需要进行格式化。

JavaScript是一种原属于网页的面向对象脚本语言，能够在脱离服务器的条件下动态地相应用户操作或不操作，能够对HTML对象进行标签元素动态地增删，还能够通过设置标签属性或样式属性控制页面样式，如今在众多优秀的开源库和框架的帮助下，JavaScript展现出了丰富的交互功能和强大的表达能力。

在本设计中，Bootstrap、jQuery和Echarts等组件因为出色的功能和频繁的使用不得不提：

a) Bootstrap是一套优秀且完备的前端开发框架，提供了几乎所有会被使用的网页组件，并且定义了美观的组件样式和丰富的交互功能，大大提高了前端开发成本和效率；

b) jQuery是一个快速、优雅的JavaScript框架，正如其设计宗旨“write less， do more”所言，其帮助开发者用更少的代码完成更多的功能，具有清晰的链式语法、简洁的多功能接口和灵活的选择器，其中jQueryUI是基于jQuery的界面插件；

c) Echarts是百度提供的开源可视化库，提供直观、丰富且高度个性化的数据可视化图表组件，只需要简单地配置图表对象即可绘制折线图、柱状图、散点图和K线图等常规图表，在一定数据支持下，甚至可以绘制三维空间图、地图和网络关系图等复杂图表。

### 2.2.2后端技术——MySQL+Flask

MySQL是甲骨文公司的一个关系型数据库管理系统，也是最流行的数据库管理系统之一，使用SQL标准化语言访问数据库，具有体积小、速度快和成本低的优点，能够满足本系统的一切开发需求。

Flask是Python语言编写并应用于Python开发环境的轻量级Web应用框架，其本身旨在核心简单并且易于扩展，因此并不具备Web框架以外的冗余功能，但这些功能都可以有灵活且兼容地扩展到Flask上去。

### 2.2.3算法框架——TensorFlow+Scikit-learn

TensorFlow是采用数据流图形式进行数据计算的机器学习开源库，支持Python语言是被应用于本系统的主要原因。在TensorFlow中，多维数据数组用张量表示，对张量的各种数学操作用节点表示，张量在多种操作间传递构成了节点间的路径，这种形式的算法框架能够方便地控制工作流，查看各节点的输入输出情况，调试异常结果的来源与去向。而且，由于本系统的卷积神经网络模型的负反馈过程中需要进行大量的微分求导运算，模型的训练效率与复杂程度将极大受益于TensorFlow的自动求微分特性。此外TensorFlow能够充分调度多种硬件设备的计算资源，动态调整计算效率，并能够在各种操作系统上移植，为系统的进一步优化完善提供可能。

Scikit-learn是建立在NumPy、SciPy和matplotlib基础上的Python机器学习工具，集成了几乎所有监督学习和无监督学习算法，并且包括了数据挖掘全过程的辅助工具，如特征提取、数据预处理、无监督降维和随机投影等函数，仅仅数行代码即可实现简单高效的数据分析和挖掘。

# 3.中医藏象辨证量化诊断方法研究

## 3.1数据准备

依托学校中医资源，获取到来自中医院的原始医案数据7964条，其形式为：每一行为一条医案数据，每一列为一个数据项，每一数据项又包含若干词条。限于篇幅，前十条医案如下图2所示：

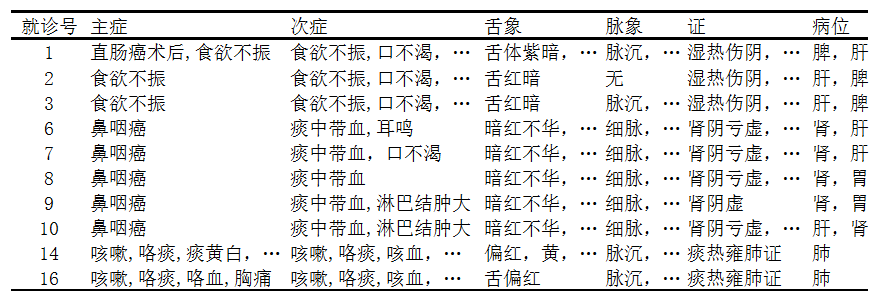


图2 原始医案示例

其中针对中医藏象辨证的研究内容，仅保留相关的主症、次症、舌象、脉象和证型等数据项，其中前四项为诊断依据，最后一项为诊断结果，即需从主症、次症、舌象和脉象中提取特征，从证型中提取标签；同时由于本系统的AdaBoost算法、随机森林算法、卷积神经网络算法为监督学习，而谱聚类也需要根据簇中其他样本的标签计算待分类样本标签，故而证型项无数据的样本即为无效样本，筛去之后共得到7516条有效样本

### 3.1.1医案清洗

上述得到的7516条样本中依旧存在各种类型的脏数据，具体问题和解决方法如下：

a) 数据项里分隔符不统一，有英文逗号、中文逗号、顿号、空格、冒号、分号、“兼”、“和”、“或”或者无分隔符等十种可能，前九种均可作整体替换为英文逗号，最后采用人工清洗的方式，在特征与标签词典收录完备的情况下还可采用字符串匹配提取局部特征与标签；

b) 同一样本的不同数据项包含相同特征，如第一条样本的主症和次症中都包含了“食欲不振”特征，编写脚本过滤掉；

c) 数据项交错重叠，如舌象和脉象数据项颠倒、主症中包含舌象特征等类似错误，人工清洗，在在特征与标签词典收录完备的情况下还可对特征应属数据项作判断归类；

d) 音近词、形近词混用，如“肺壅”和“肺痈”、“瘀青”和“淤青”等类似错误，统一用词，批量替换；

e) 异词同义，如“食欲不振”和“胃口欠佳”、“睡眠不实”和“寐差”等，人工清洗并建立特征同义词典；

f) 无关词句，如“无不适”、“无不良反应”、“化疗8次后”等人工清洗；

g) 程度词造成不必要区分，如“新近感受风寒”和“3天前感受风寒”、“关节炎5年”和“关节炎4年”，人工清洗并过滤相似程度词；

h) 修饰词造成不必要区分，如“夜间大汗”和“中午大汗”、“疼痛尤甚”和“疼痛不止”等，人工清洗并过滤相似程度词；

i) 一词包含多义，如“心气血两虚证”包含“心气虚”和“心血虚”等两种证型，人工清洗并建立词典，编写脚本替换拆分；

j) 前后矛盾，如主症项包含“睡眠不实”，次症项包含“嗜睡”等类似错误，人工删除；

k) 指代不明，如“带下”未指明“带下异常”还是“带下正常”等错误，人工删除。

### 3.1.2医案标准化

如3.1.1中提到的d、e、i等问题有些可以直观辨别出来并纠正，如“高血压”和“高血压病”，而更多的不是这样，如“痰邪阻肺”和“痰湿内停”等错误，必须有专业知识储备才能辨别，因此，本设计在两名中医专业学生的帮助下进行了医案标准化工作，具体任务如下：

a) 基于医案样本，建立同义词典，将意义相近、表述不同的专业术语统一成标准表述；

b) 纠正医案中与术语相关的错字、别字；

c) 将复合证型拆分成单一基础证型。

截至本系统完成，已完成700条样本的标准化工作，而同义词典中尚不包含大量高频、典型特征与标签，仅使用词典中出现的特征与标签将会使样本失真，算法表现大打折扣。因此，考虑到高频词与典型词的参考价值，本系统在统计样本词频与参考基本证型的基础上，结合同义词典，得到症状特征1871条、舌象特征32条、脉象特征16条、证型标签50条，过滤掉频数较低、表述生僻的特征与标签。限于篇幅，部分标签如下表1所示：

表1 部分特征/标签频数统计表

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 症状 | 频数 | 舌象 | 频数 | 脉象 | 频数 | 证型 | 频数 |
| 口干 | 827 | 苔薄 | 4602 | 脉细 | 4012 | 肝经湿热 | 648 |
| 胸闷 | 598 | 舌红 | 4028 | 脉滑 | 3232 | 肝郁脾虚 | 520 |
| 咳嗽 | 551 | 苔黄 | 3860 | 脉弦 | 2770 | 痰瘀 | 392 |
| 寐差 | 377 | 苔腻 | 3693 | 脉沉 | 1064 | 肝肾阴虚 | 382 |
| 疲劳 | 330 | 舌暗 | 3507 | 脉小 | 890 | 肺气阴虚 | 361 |
| 纳差 | 325 | 苔白 | 1973 | 脉数 | 432 | 痰邪 | 329 |
| 腰酸 | 310 | 舌紫 | 1213 | 脉濡 | 217 | 肝胆湿热 | 325 |
| 腹胀 | 309 | 舌淡 | 845 | 脉弱 | 208 | 肾阴虚 | 291 |
| 尿黄 | 305 | 苔淡 | 598 | 脉缓 | 115 | 肝郁气滞 | 268 |
| 心慌 | 271 | 舌裂 | 523 | 脉紧 | 72 | 肾阳虚 | 260 |

## 3.2 AdaBoost算法建模

### 3.2.1构造特征向量

经由上一步数据清洗得到了特征及标签，假设在所有特征（症状特征1871条、舌象特征32条、脉象特征16条）共同构成了以每一特征为维度的1919维特征空间，每个样本映射到特征空间为一个点，若样本中存在该特征，则相应维度的取值为1，否则为0。以四维特征空间{胸闷，舌红，脉滑，肝郁脾虚}为例，假设样本为{胸闷，舌红}，则其特征向量A在特征空间中表示为：

类似地，针对本设计使用的样本数据，构造特征向量逻辑与Python代码如下：

1. #各数据项列号
2. feat\_index\_list = [0, 1, 2, 3]
3. #对于每一样本，操作如下
4. **for** i **in** range(rows - 1):
5. #对于每一数据项，操作如下
6. **for** k **in** range(len(feat\_index\_list)):
7. #使用分隔符切分数据项得到特征列表
8. temp\_list = re.split(',', sheet.row\_values(i + 1)[feat\_index\_list[k]])
9. #对于特征列表每一项，操作如下
10. **for** j **in** temp\_list:
11. #对应特征向量对应特征序号位置置1
12. feature\_matrix[i][featureSet.index(j)] = 1
13. #返回特征向量集合
14. **return** feature\_matrix

由于AdaBoost算法不支持多标签学习，因此针对样本数据构造的标签向量应为单标签值，其逻辑与代码如下：

1. #输入：样本数据路径，标签字典
2. #输出：样本数据对应的标签列表
3. #说明：标签字典形如{'标签名称':序号}->{'心气虚':0}
4. **def** create\_tag\_list(path, tagSet):
5. #打开数据样本文件
6. workbook = xlrd.open\_workbook(path)
7. sheet = workbook.sheet\_by\_index(0)
8. rows = size + 1
9. #初始化标签列表
10. tag\_matrix = [0 **for** i **in** range(rows - 1)]
11. #对于每一样本，操作如下
12. **for** i **in** range(rows - 1):
13. #用分隔符切分证型数据项
14. temp\_lsit = re.split(',', sheet.row\_values(i + 1)[4])
15. #取第一个证型作为标签
16. tag\_matrix[i] = tagSet.index(temp\_lsit[0])
17. #返回标签列表
18. **return** tag\_matrix

至此得到了1919维的二值特征向量集合和单标签序号列表。

### 3.2.2构建决策树

分类决策树的构建分为建树和剪枝两个过程**。**

首先，建树具体包含两个问题：

a) 如何确定特征顺序，即根节点是哪个变量，第二层是哪些变量等等，这些变量是怎么排序的？

决策树的方法是对所有尚未作为节点的特征都尝试一遍选取能够使划分最好的特征作为当前节点使用的特征。如何评价划分好坏？决策树有信息增益、信息增益率和基尼系数三种指标用于评价，分别对应ICD3决策树，C4.5决策树和CART决策树。三种指标计算得到的划分前后差值称为纯度增益，若特征a的纯度增益大于特征b的纯度增益，则当前节点优先选用特征a作为划分依据。

b) 如何选择特征的最佳划分点呢，即对于特征a有多个取值，以哪个值作为划分样本的阈值呢？

决策树的方法依旧是穷举所有离散取值，以划分前后指标作为依据，指标变化越大，纯度增益越大，该值作为划分点越可靠。以基尼系数为例，其计算方式如下：

其中为类别*j*在样本*T*中出现的频率，为类别*j*在样本*T*中出现的个数，为样本个数。信息增益与信息增益率两个指标类似，详情可查参考文献[7]。

其次，剪枝过程，如果任由上述方法生成树，则树的深度将会导致过拟合问题，为避免该问题，可通过下面六个条件进行预剪枝：节点所有样本属同一标签、节点所有特征取值相同、树的深度达到阈值、当前节点所含样本小于父节点应含样本数阈值、子节点样本数小于当前节点应含样本数阈值或者没有特征满足分裂准则的阈值；也可以通过增加惩罚项、试探性剪枝验证误差小于阈值等方式进行后剪枝。

### 3.2.3 Boosting集成学习

Boosting元算法框架是每次使用全部样本对多个弱分类器进行线性组装，各分类器根据错误率调整的权重投票强分类器。具体算法流程如下图3所示：

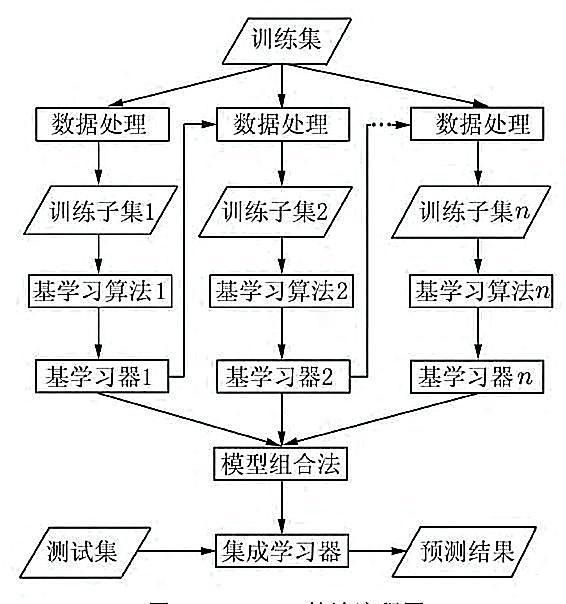


图3 Boosting算法流程

具体实现如下**：**

1. #输入：特征向量集合，标签列表，决策树最大深度，最小分割样本数量，最小样本数量，弱学习器数量，收敛系数
2. #输出：AdaBoostClassifier平均准确率
3. **def** ABC(X, y, max\_depth, min\_samples\_split, min\_samples\_split， n\_estimators，learning\_rate):
4. #构建决策树弱分类器
5. decisionTree = DecisionTreeClassifier(max\_depth=max\_depth,
6. min\_samples\_split=min\_samples\_split,
7. min\_samples\_leaf=min\_samples\_split)
8. #用弱学习器迭代得到AdaBoost分类器
9. abc = AdaBoostClassifier(decisionTree,
10. algorithm="SAMME",
11. n\_estimators=n\_estimators,
12. learning\_rate=learning\_rate)
13. #AdaBoost拟合
14. abc.fit(X, y)
15. #返回平均准确率
16. **return** abc.score()

### 3.2.4算法评估

单标签分类算法评估指标主要有准确率、精确率、召回率和F1值，各指标定义与计算方法如下：

a) 准确率Accuracy是指所有分类结果中被正确分类的百分比，计算公式为：

其中，*TP*为正确分到正类的样本数量，*TF*正确分到负类的样本数量，*FP*为错误分到正类的样本数量，*FN*为错误分到负类的样本数量，这是针对二分类和多分类都适用的指标；

b) 精确率Precision是指正确分到正类占所有分到正类的百分比，计算公式为：

这是针对二分类的指标，对于多分类中的每一类，当前类视作正类，其余类视作负类，则对每个类都有精确率，所有类的算术平均为宏精确率Macro-Precision，计算公式为：

还有微精确率Micro-Precison，其计算公式为：

c) 召回率Recall是指正类中被正确分类到正类的百分比，计算公式为：

这是针对二分类的指标，对于多分类中的每一类，当前类视作正类，其余类视作负类，则对每个类都有精确率，所有类的算术平均为宏召回率Macro-Recall，计算公式为：

还有微召回率Micro-Recall，其计算公式为：

d) 由于精确率和召回率分别侧重分得对和分得全，只考虑其中一方面可能造成严重错误，所以用F1值作为几何平均调和，其计算公式为：

将宏精确率和宏召回率代入得到宏F1，将微精确率和微召回率代入得到微F1。

使用AdaBoost算法对样本数据进行单标签训练分类，在调参过程中，以CART树作为弱学习器、最小分割样本数为5、最大树深度为100的条件下算法得到最优准确率44.62%，此时其他指标如表2所示。

表2 AdaBoost评估指标表

|  |  |  |  |
| --- | --- | --- | --- |
|  | 精确率 | 召回率 | F1值 |
| 微指标 | 37.62% | 47.69% | 42.06% |
| 宏指标 | 39.82% | 46.37% | 42.84% |

该表现极大程度上受到样本单标签化的影响，对于同一样本，由于标签排列顺序不同而被划分到不同类别，给分类器造成混肴的影响，分类结果在多个标签之间震荡，进而导致模型分类不稳定，对多标签样本不敏感，表现较差。

## 3.3随机森林算法建模

### 3.3.1构造特征向量

随机森林的特征向量与AdaBoost算法一样，都是特征对应位置置1的二值向量，可以直接用上述函数构造随机森林的特征向量。

与AdaBoost算法不同的是，随机森林算法支持多标签输出，即同一个医案样本在随机森林算法的分类下可以得到多个类别标签，这就意味着样本标签同样需要建立向量，逻辑与Python代码与特征向量的构建类似，具体如下：

1. #输入：样本数据路径，标签字典
2. #输出：样本数据对应的标签向量集合
3. #说明：标签字典形如{'特征名称':序号}->{'心气虚':0}
4. **def** create\_tag\_matrix(path, tagSet):
5. #打开样本数据文件
6. workbook = xlrd.open\_workbook(path)
7. sheet = workbook.sheet\_by\_index(0)
8. rows = size + 1
9. #初始化标签向量集合，所有标签置0
10. tag\_matrix = [[0 **for** i **in** range(50)] **for** i **in** range(rows - 1)]
11. #对于每一样本，操作如下
12. **for** i **in** range(rows - 1):
13. #使用分隔符切分数据项得到标签列表
14. temp\_list = re.split(',', sheet.row\_values(i + 1)[4])
15. #对于标签列表每一项，操作如下
16. **for** j **in** temp\_list:
17. #对应标签向量对应特征序号位置置1
18. tag\_matrix[i][tagSet.index(j)] = 1
19. #返回标签向量集合
20. **return** tag\_matrix

### 3.3.2构建决策树

随机森林的决策树构建与AdaBoost算法完全一致，具体可参考3.2.2节内容。

### 3.3.3 Bagging集成学习

Bagging元算法框架是多次放回采样通过学习算法形成多个弱分类器，投票（权重相等）形成强分类器，算法具体流程如下：

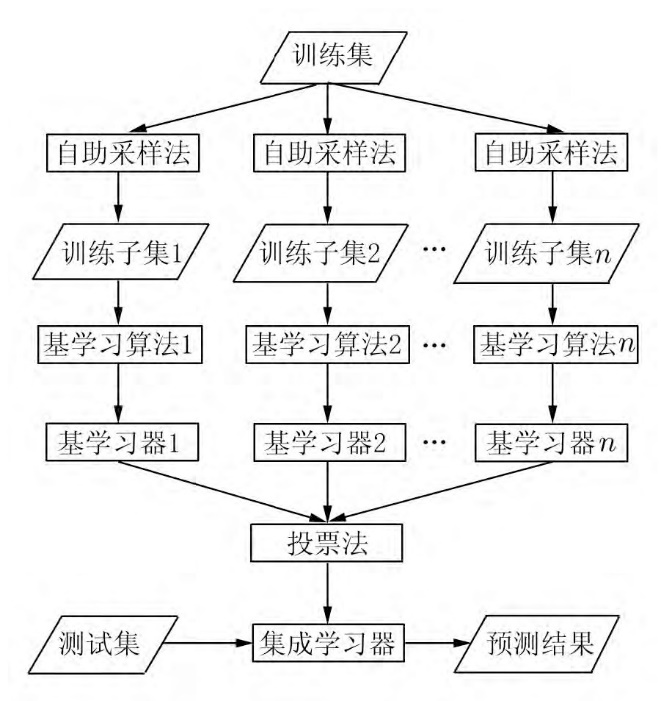


图4 Bagging算法流程

具体实现如下**：**

1. #输入：特征向量集合，标签列表，决策树最大深度，最小分割样本数量，最小样本数量，弱学习器数量，收敛系数
2. #输出：随机森林平均准确率
3. **def** ABC(X, y, max\_depth, min\_samples\_split, min\_samples\_split， n\_estimators，learning\_rate):
4. #构建决策树弱分类器
5. decisionTree = DecisionTreeClassifier(max\_depth=max\_depth,
6. min\_samples\_split=min\_samples\_split,
7. min\_samples\_leaf=min\_samples\_split)
8. #用弱学习器迭代得到随机森林分类器，用一对其余分类器封装使其具有多标签分类能力
9. multiClassifier = OneVsRestClassifier(RandomForestClassifier(decisionTree,
10. algorithm="SAMME",
11. n\_estimators=n\_estimators,
12. learning\_rate=learning\_rate))
13. #多标签随机森林分类器拟合
14. multiClassifier.fit(X, y)
15. #返回平均准确率
16. **return** multiClassifier.score()

### 3.3.4算法评估

多标签分类的评估指标主要有汉明损失、错误率和覆盖率，各指标定义和计算公式如下：

a) 汉明损失Haming Loss是平均差异比率，其计算公式如下：

其中，*N*是样本数量，*L*是标签数量，是对算法输出的标签向量和实际标签向量的每一位作异或运算得到的累计值；

b) 错误率one-error（OE）是分类结果的标签向量中排第一的标签不属于实际标签向量的概率：

其中指的是取中排名第一的标签。当问题为单标签时，1-OE恰为准确率，故以下的所有多标签准确率计算为1-OE；

c) 覆盖率Coverage是在分类结果的标签向量中排名比实际标签向量中最后的标签靠前的平均标签数量，其计算公式为：

其中，指的是取中排名最后的标签，指的是y中排名最靠后的标签在中的排名。

使用随机森林算法对样本数据进行多标签训练分类，在调参过程中，以CART树作为弱学习器、最小分割样本数为5、最大树深度为100的条件下算法得到最优准确率47.59%，此时其他指标如表3所示：

表3 随机森林评估指标表

|  |  |  |
| --- | --- | --- |
| 汉明损失 | 错误率 | 覆盖率 |
| 3.77% | 52.41% | 7.92 |

从表中可知，随机森林算法得到的标签向量平均有7个标签与实际标签不同，并且平均有8个标签排在最后的实际标签之前，错误率也相当高。

## 3.4卷积神经网络建模

卷积神经网络本是用于图像特征提取和分类的深度学习网络模型，其基本原理如图5所示：

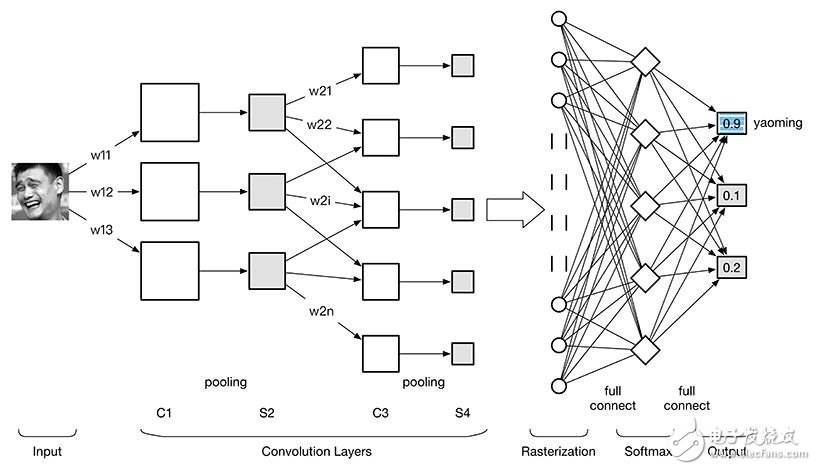


图5 卷积神经网络原理图

图像被视作多通道的矩阵，像素值的分布作为输入层的特征，输入给卷积核，每个卷积核提取局部特征，形成各卷积特征分布图，比如有个提取直线特征的卷积核，它能够提取到图像上的直线分布；再经过池化层池化，将特征泛化，这样就能把近似直线的特征也看作直线；如此经过多层的卷积和池化操作，模型已经提取到相当高维度特征了，比如直线特征组合能够形成四边形特征、圆形特征等，圆形特征和四边形特征组合形成面部特征，最终得到眼睛、鼻子和耳朵等面部器官在图像上的分布；这些特征输入到全连接层，进行logit分类，得到模型输出，即图像分类。

卷积神经网络的一般工作流程如图6所示：

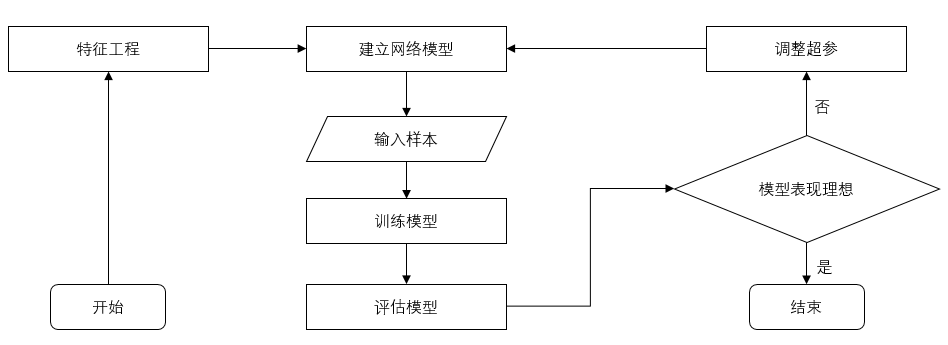


图6 卷积神经网络工作流程

Wang Z等人在研究中“将每位患者的记录视为一维图像，并将症状视为像素，其中缺失值和负值由零像素表示。目的是首先找到相关症状，然后将它们映射到适当的综合症，这类似于计算机视觉中的物体检测问题……采用多实例多任务学习结合卷积神经网络（MIMT-CNN）进行辨证，将区域建议直接作为输入和输出图像标签”[8]。

本设计再此基础上进一步提出自己关于局部性原理的尝试，在图像的像素组织上将相关性特征放到临近位置，以期隐含层能够形成更多规律性的特征分布，用更少的样本学到更多的规律。打个比方，如果人的五官分布打乱，眉毛和眼睛分布较远、鼻子和下巴分布接近，这将使得人对于面部特征的认知受到障碍，需要花费更多认知经验去掌握面部特征规律。

### 3.4.1构造特征向量

与随机森林算法和AdaBoost算法不同，卷积神经网络的特别之处在于其特征向量的排列需要根据特征之间相关性填充成图像规模**。**

根据样本中提取的1919条特征，可将线性排列的特征向量转换成44\*44的矩阵，考虑到44的因子过少，不利于卷积神经网络的多层次池化处理，将其转换成48\*48的矩阵，能够允许以2、3及其公倍数为步长的池化处理。

在逻辑和Python代码上，卷积神经网络的特征向量区别仅在于定义的特征字典的不同，具体而言字典中key越相关，value值越可能具有n或48+n（n趋近于0）的绝对差值关系。具体而言，“腹部疼痛”和“腰部疼痛”两个症状具有极大相关性，体现在其在脚本统计中的高频伴随出现关系，此时输入层的这两个特征可能指向隐含层的特征，因此在特征像素图上，两个像素点应左右相邻（绝对差值趋近0）或上下相邻（绝对差值趋近于48+0）。

特征向量的其他处理与3.2.1节相同，标签向量与3.3.2节相同。

### 3.4.2设计网络模型

网络模型参考AlexNet网络结构[9],针对上述所得的48\*48个特征图像，设计网络结构分为两个卷积池化层、两个全连接层和一个输出层。第一个卷积层有32个8\*8的卷积核，第二个卷积层有64个32通道的4\*4卷积核，池化采用2\*2的最大池化方式。每一层均有偏置矩阵，卷积层使用relu激活函数，全连接层使用sigmoid激活函数，第一个全连接层前特征矩阵由12\*12的64个矩阵转换为12\*12\*64的一维向量，标签向量每一维为对应标签的概率。

具体如图5所示：

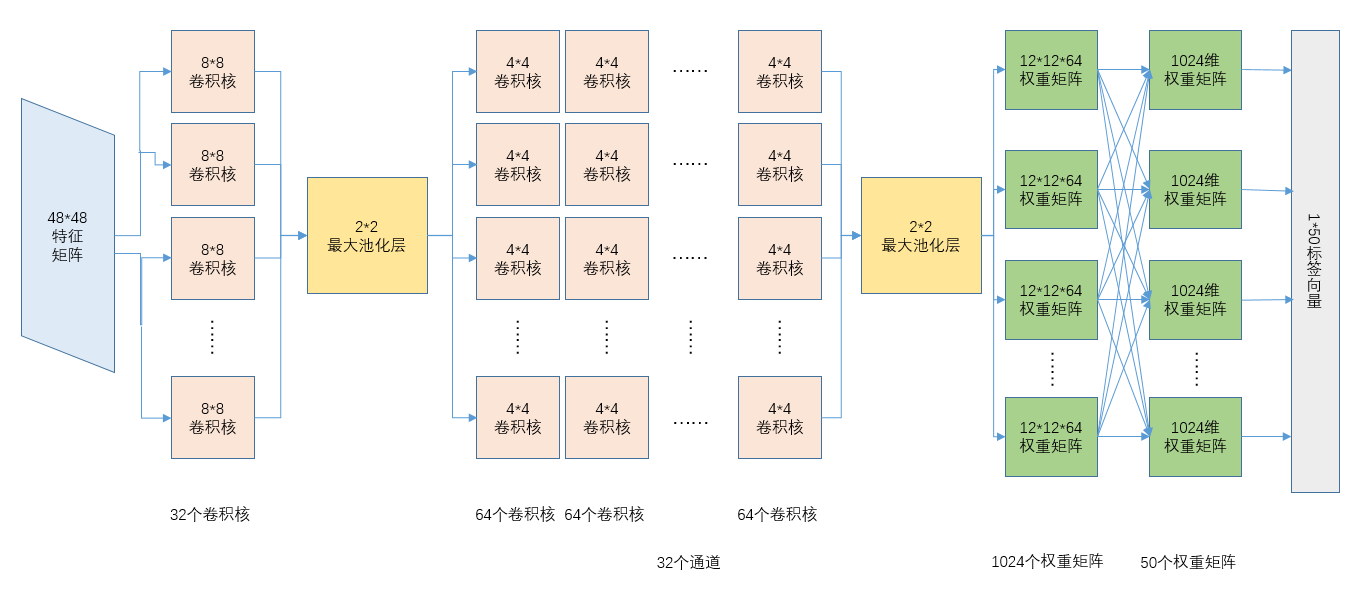


图7 网络模型

根据以上模型，建立网络逻辑与Python代码如下：

1. #输入：权重矩阵维度
2. #输出：权重矩阵
3. **def** weight\_variable(shape):
4. initial = tf.truncated\_normal(shape, stddev=0.1)
5. **return** tf.Variable(initial)
7. #输入：偏置矩阵维度
8. #输出：偏置矩阵
9. **def** bias\_variable(shape):
10. initial = tf.constant(0.1, shape=shape)
11. **return** tf.Variable(initial)
13. #输入：图像和卷积核
14. #输出：卷积特征分布矩阵
15. **def** conv2d(x, W):
16. **return** tf.nn.conv2d(x, W, strides=[1, 1, 1, 1], padding='SAME')
18. #输入：待池化矩阵
19. #输出：2\*2池化处理后矩阵
20. **def** max\_pool\_2x2(x):
21. **return** tf.nn.max\_pool(x, ksize=[1, 2, 2, 1], strides=[1, 2, 2, 1], padding='SAME')
23. #待输入特征向量
24. x = tf.placeholder(tf.float32, [None, 2304])
25. #待输入标签向量
26. y\_ = tf.placeholder(tf.float32, [None, 50])
27. #向量转化为图像
28. x\_image = tf.reshape(x, [-1, 48, 48, 1])
30. #创建32个8\*8卷积核
31. W\_conv1 = weight\_variable([8, 8, 1, 32])
32. b\_conv1 = bias\_variable([32])
33. #使用Relu激活图像卷积后特征图像
34. h\_conv1 = tf.nn.relu(conv2d(x\_image, W\_conv1) + b\_conv1)
35. #2\*2池化处理
36. h\_pool1 = max\_pool\_2x2(h\_conv1)
37. #创建64个32通道的4\*4卷积核
38. W\_conv2 = weight\_variable([4, 4, 32, 64])
39. b\_conv2 = bias\_variable([64])
40. #使用Relu激活图像卷积后特征图像
41. h\_conv2 = tf.nn.relu(conv2d(h\_pool1, W\_conv2) + b\_conv2)
42. h\_pool2 = max\_pool\_2x2(h\_conv2)
44. W\_fc1 = weight\_variable([4 \* 4 \* 64, 1024])
45. b\_fc1 = weight\_variable([1024])
46. #将图像展开成向量
47. h\_pool2\_flat = tf.reshape(h\_pool2, [-1, 4 \* 4 \* 64])
48. #全连接
49. h\_fc1 = tf.nn.relu(tf.matmul(h\_pool2\_flat, W\_fc1) + b\_fc1)
51. keep\_prob = tf.placeholder(tf.float32)
52. #dropout处理
53. h\_fc1\_drop = tf.nn.dropout(h\_fc1, keep\_prob)
55. w\_fc2 = weight\_variable([1024, 36])
56. b\_fc2 = weight\_variable([36])
57. #使用sigmoid激活全连接输出得到各类分到的概率
58. y\_conv = tf.nn.softmax(tf.matmul(h\_fc1\_drop, w\_fc2) + b\_fc2)

### 3.4.3 训练网络

由此得到前馈网络的预测分类结果，接下来负反馈传播预测分类与实际标签的损失，调整各层矩阵权重和偏置参数取值。首先定义损失函数和交叉熵，以降低交叉熵为优化目标，定制优化器，设置学习速率，其影响收敛速度以及是否造成损失震荡。设置dropout比率避免，过度拟合，增强网络泛化能力，输入特征向量与标签向量，启动优化器，开始训练。

其训练具体过程本质上就是通过链式求导微分，将损失梯度下降到各参数，从而调整各层参数使得模型损失尽可能减小，即在训练样本上分类结果尽可能接近实际标签。

具体Python代码如下：

1. #定义交叉熵
2. cross\_entropy = tf.reduce\_mean(-tf.reduce\_sum(y\_ \* tf.log(y\_conv), reduction\_indices=[1]))
3. #设置Adam自动优化器优化目标为使交叉熵结果最小
4. train\_step = tf.train.AdamOptimizer(1e-4).minimize(cross\_entropy)
5. #计算正确分类标签数量
6. correct\_prediction = tf.equal(y\_conv, y\_)
7. #计算准确率
8. accuracy = tf.reduce\_mean(tf.cast(correct\_prediction, tf.float32))
9. #初始化全局变量
10. tf.global\_variables\_initializer().run()
11. batch = clean\_data.next\_batch(50)
12. **for** i **in** range(50):
13. #投入当前批次特征向量与标签向量，进行训练并返回准确率
14. train\_accuracy = accuracy.eval(feed\_dict={x: batch[0], y\_: batch[1], keep\_prob: 1.0})

### 3.4.4模型评估与调参

模型在700条标准化样本上测试准确率为52.47%，其他指标如表4所示：

表4 卷积神经网络评估指标表

|  |  |  |
| --- | --- | --- |
| 汉明损失 | 错误率 | 覆盖率 |
| 3.24% | 47.59% | 5.40 |

从上表知，卷积神经网络算法得到的标签向量平均有6个标签与实际标签不同，并且平均有6个标签排在最后的实际标签之前，错误率也相当高。

算法结果在测试集上呈现出高偏差特点，同时训练集上最高也仅64.33%的准确率，方差不高，其说明样本特征尚未充分学习，同时样本数据本身存在不均衡的问题，大部分样本尚未清洗、标准化。

针对以上问题，调参思路如下：在样本清洗完毕，标准化处理充分的条件下，增加模型复杂程度，增加网络层数、卷积核数量；待到分类准确率呈现低偏差、高方差特点时，增加最大池化层规模，提高dropout比率，增强模型泛化能力，避免过拟合。

## 3.5谱聚类算法建模

### 3.5.1构造特征向量

谱聚类算法的特征向量是在3.2.1节得到的特征向量基础上，进一步将特征向量视作特征空间中的点，计算个样本映射到特征空间中的点与其他样本点的距离，一般是高斯相似度、余弦距离或者欧式距离。新的特征向量或者称作相似矩阵是样本之间关系的表达，从图的角度来看，它描述的是一个关系网络。

其构造逻辑和Python代码如下：

1. #定义交叉熵
2. cross\_entropy = tf.reduce\_mean(-tf.reduce\_sum(y\_ \* tf.log(y\_conv), reduction\_indices=[1]))
3. #设置Adam自动优化器优化目标为使交叉熵结果最小
4. train\_step = tf.train.AdamOptimizer(1e-4).minimize(cross\_entropy)
5. #计算正确分类标签数量
6. correct\_prediction = tf.equal(y\_conv, y\_)
7. #计算准确率
8. accuracy = tf.reduce\_mean(tf.cast(correct\_prediction, tf.float32))
9. #初始化全局变量
10. tf.global\_variables\_initializer().run()
11. batch = clean\_data.next\_batch(50)
12. **for** i **in** range(50):
13. #投入当前批次特征向量与标签向量，进行训练并返回准确率
14. train\_accuracy = accuracy.eval(feed\_dict={x: batch[0], y\_: batch[1], keep\_prob: 1.0})

### 3.5.2构建拉普拉斯矩阵

拉普拉斯矩阵*L*的定义十分简洁**：**

其中*W*即为上节得到的相似度矩阵**，**D为度数矩阵，也是对角矩阵，其定义为：

即，度数矩阵对角线上的每一值为相应的相似度矩阵对应行及对应列相似度之和。[10]

### 3.5.3图切分与聚类

谱聚类的切图实际上就是对相似度矩阵进行降维，得到最能够将样本空间划分成高内聚、低耦合的若干簇的特征。在谱聚类理论发展过程中，共有三种切图方法，切图效果依次增强——MiniCut、RatioCut和NCut，其中MiniCut只是简单地将图中一个顶点与其他顶点分开，而合理的切分应该是每个切分中顶点尽可能多，进而有了MiniCut和NCut。

根据文献[10]的结论，MiniCut和NCut寻找最优的图切分在数学形式上的表达分别为：

因此，只需要求出的最小的前*k*个特征值，然后求出对应的特征向量并标准化，最后得到特征矩阵*F*，进行一次聚类即可。

使用Scikit-learn，仅需要如下代码**：**

1. #在给定范围内寻找最佳簇数
2. **for** index, k **in** enumerate((3, 4, 5, 6)):
3. #输入相似度矩阵
4. X = construct\_W(X)
5. #设置聚类簇数为k，拟合并预测
6. y\_pred = SpectralClustering(n\_clusters=k).fit\_predict(X)

根据聚类结果在类内取样本标签的众数，对于多标签分类则通过设置阈值，当类内样本标签的出现频数高于阈值的，都作为分类结果输出。

### 3.5.4算法评估

当簇数取6的时候，算法取得最高准确率为39.28%，其他指标如表5所示：

表5 谱聚类评估指标表

|  |  |  |
| --- | --- | --- |
| 汉明损失 | 错误率 | 覆盖率 |
| 2.94% | 60.72% | 10.26 |

谱聚类算法得到的标签向量平均有6个标签与实际标签不同，并且平均有11个标签排在最后的实际标签之前，错误率最高。主要原因是，在对标签分类任务中，人为设置阈值太过机械，没有充分的依据，对于样本较少的类别，更倾向于输出单标签结果，反之，样本较多的类别倾向于输出多标签结果。

## 3.6算法总结与比较

在以上四种算法中，卷积神经网络以其局部性特征提取以及随着模型复杂提升的拟合能力在准确率方面表现最佳；随机森林以效果出众的决策树作为弱分类器，通过大量迭代和一对其他分类器的封装，在多标签分类任务中也有着相对不错的准确率；AdaBoost和谱聚类对于多标签分类表现较差，但谱聚类算法以其相对简单的矩阵特征值计算和聚类，在算法效率上表现最佳。

整体而言，四种算法的表现均远未达到实际应用的水平，这在一定程度上由于受到样本数据本身缺陷的影响，在清洗和标准化之后，有望得到进一步提升。

# 4.中医藏象智能诊断平台的设计与实现

## 4.1系统总体架构

本系统总体架构如图8所示：

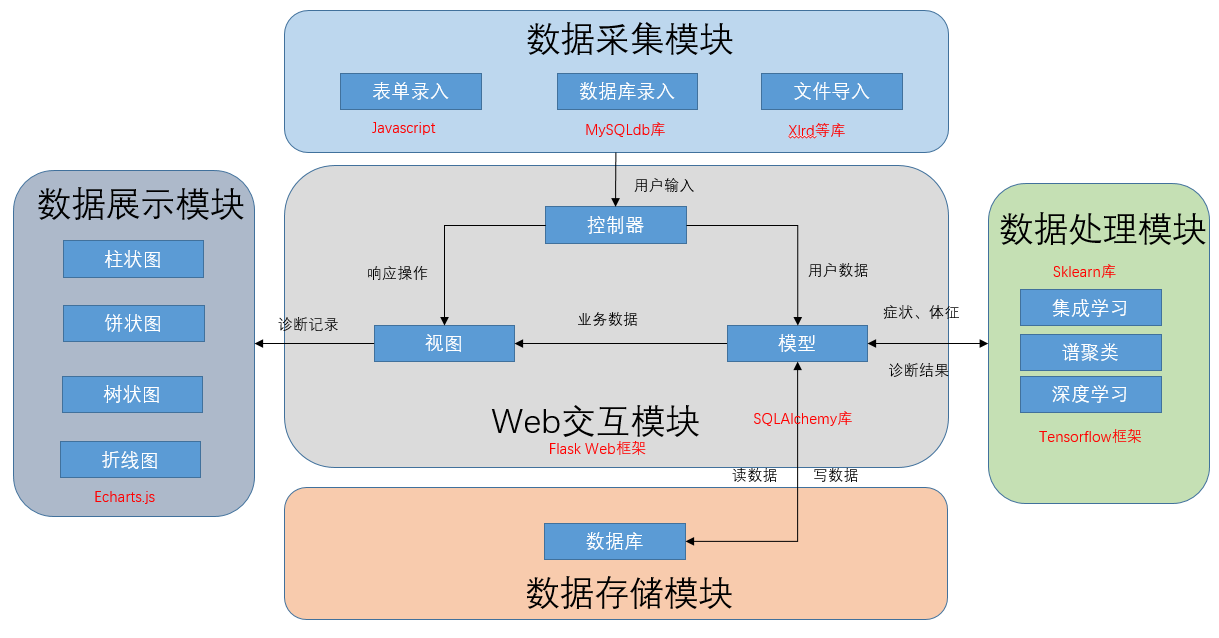


图8 系统总体架构图

本系统采用B/S架构，使用Python语言进行后端开发，Html、Javascript、CSS进行前端开发。系统分为五个功能模块，即数据采集模块、Web交互模块、数据处理模块、数据存储模块以及数据展示模块：

a) 数据采集模块，顾名思义，用于收集用户门诊信息，支持单项表单录入或语音输入，以及批量文件导入。表单录入用Javascript进行预处理之后发送给服务器进行进一步处理；数据库录入只需要用户填写数据库连接以及字段映射关系，服务器使用MySQLdb库实现数据库的复制；文件批量导入功能利用Python强大的工具库对用户本地进行读写操作，提取多条患者信息；

b) Web交互模块，即负责与用户进行交互，同时协调后端组件进行工作，系统基于Python强大的Flask框架进行开发，采用MVC的软件设计模式，控制器负责接受处理用户输入请求，指示响应操作，模型负责与数据库以及数据处理模块的交互，对数据进行读写并发送给视图，视图负责处理要返回给用户的界面；

c) 数据处理模块，即负责算法模型对数据的处理，利用Sklearn包的AdaBoost、随机森林等算法进行集成学习，利用Tensorflow框架进行深度学习，对输入的症状、体征信息进行量化诊断并返回准确率、召回率等指标，以便进行优化调整；

d) 数据存储模块，本系统采用MySQL数据库，仅需要Flask-SQLAlchemy库即可用Python与之交互读写；

e) 数据展示模块，即负责向用户直观地展示历史诊断结果，系统采用Echarts.js组件进行作图。

## 4.2系统功能设计

### 4.2.1医案录入

医案录入的使用流程设计如图9所示：

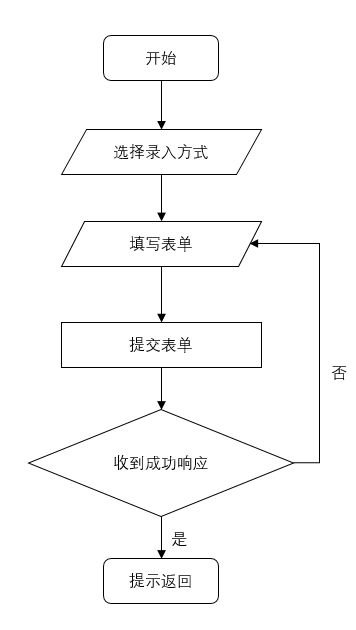


图9 医案录入使用流程

医案录入支持表单、文本、表格以及数据库录入方式。

表单录入即用户直接在系统内填写对应数据项即可，用户在填写数据项时，可以通过输入字典库维护的名称首字母缩写，得到候选下拉框，进而快速选择标准词条，如：在填写主症项时，输入“kg”可以得到“口干”、“口干苦”等若干选项，选择完成录入该词条。

文本录入即用户直接从文本文件中摘取医案段落粘贴到文本编辑区，按照医案模板进行格式化。

表格录入即用户点击文件输入控件打开文件框，选择xls或xlsx格式的Excel文件，并填写行列与数据项的映射关系。

数据库录入即用户填写数据库连接字符串、表、字段映射关系。

用户可根据需要清空或提交已填写内容。

医案录入功能的实现分为前端页面设计、脚本编写，后端路由、视图层、控制层和模型层编写，具体设计细节如下：

a) 前端页面主要使用bootstrap的nav类导航栏、tab类的选项卡和form-control类的表单样式以及自己定义的button样式和文件上传样式；

b) 脚本主要是表单录入需要智能提示，使用的是autocomplete组件，其需要用到后端字典的查询和输入控制，具体代码以症状的自动完成为例：

1. $('.auto\_symptom').autocomplete({
2. source: **function** (request, response) {
3. **var** data = {
4. abbr: extractLast(request.term),
5. }
6. $.ajax({
7. url: 'search\_symptom',
8. type: 'POST',
9. data: JSON.stringify(data),
10. success: **function** (data) {
11. **if** (data['code'] != "-1") {
12. response(data);
13. } **else** {
14. toastr.options.timeOut = "2000";
15. toastr.options.closeButton = **true**;
16. toastr['error']('<small>查询症状失败</small>', '', {positionClass: 'toast-bottom-center'});
17. }
18. }
19. })
20. },
21. search: **function** () {
22. **var** term = extractLast(**this**.value);
23. **if** (term.length < 2) {
24. **return** **false**;
25. }
26. },
27. focus: **function** () {
28. **return** **false**;
29. },
30. select: **function** (event, ui) {
31. **var** terms = split(**this**.value);
32. terms.pop();
33. terms.push(ui.item.value);
34. terms.push("");
35. **this**.value = terms.join(",");
36. **return** **false**;
37. }
38. });

其中，查询结果的的提示框使用的是toastr组件，能够提供美观的悬浮动画样式；

c) 在routes中配置路由，接受GET和POST两种方式，并且需要授权验证，未登录用户不能访问；

d) 模型层主要是对原始医案表进行写操作以及字典查询操作，用Flask—SQLAlchemy的ORM模型对对象操作即可，以症状字典的查询为例，具体代码如下：

1. **def** add\_raw\_rcd(main\_sym, add\_sym, tongue, pulse, syndrome):
2. **try**:
3. rcd = RawRecord()
4. rcd.main\_sym = main\_sym
5. rcd.add\_sym = add\_sym
6. rcd.tongue = tongue
7. rcd.pulse = pulse
8. rcd.syndrome = syndrome
9. db.session.add(rcd)
10. db.session.commit()
11. **return** {'code': 1}
12. **except** Exception as e:
13. **print**(e)
14. **return** {'code': -1, 'message': '异常'}
16. **def** search\_symptom(abbr):
17. **try**:
18. res = db.session.query(Symptom.sym\_name).filter(Symptom.abbr.like('%' + abbr.replace("'", "") + '%'),
19. Symptom.std\_id == 0).all()
20. data = [i.sym\_name **for** i **in** res]
21. **return** data
22. **except** Exception as e:
23. **print**(e)
24. **return** {'code': -1}

e) 视图层主要是对前端ajax封装的request数据进行解析，传递给控制层；而控制层主要是对调用模型层的数据库操作和预处理函数，以及对错误的输入输出返回相应的提示，以数据库方式添加原始医案为例，代码如下：

1. **def** add\_database\_rcd(data):
2. (user, password, host, port, database) = re.findall(r'(.+):(.+)@(.+):(.+)/(.+)', data['conn'])[0]
3. **try**:
4. conn = pymysql.connect(user=user, password=password, port=int(port), host=host, database=database)
5. cur = conn.cursor()
6. sql = 'select %s, %s, %s, %s, %s from %s' % (
7. data['main'], data['add'], data['tongue'], data['pulse'], data['syndrome'], data['table'])
8. cur.execute(sql)
9. res = cur.fetchall()
10. **for** i **in** res:
11. **if** model.add\_raw\_rcd(i[0], i[1], i[2], i[3], i[4])['code'] == -1:
12. **return** {'code': -1, 'message': '插入医案异常'}
13. **return** {'code': 1}
14. **except** Exception as e:
15. **print**(e)
16. **return** {'code': -1, 'message': '数据库连接有误'}

### 4.2.2医案清洗

医案清洗的使用流程设计如图10所示：

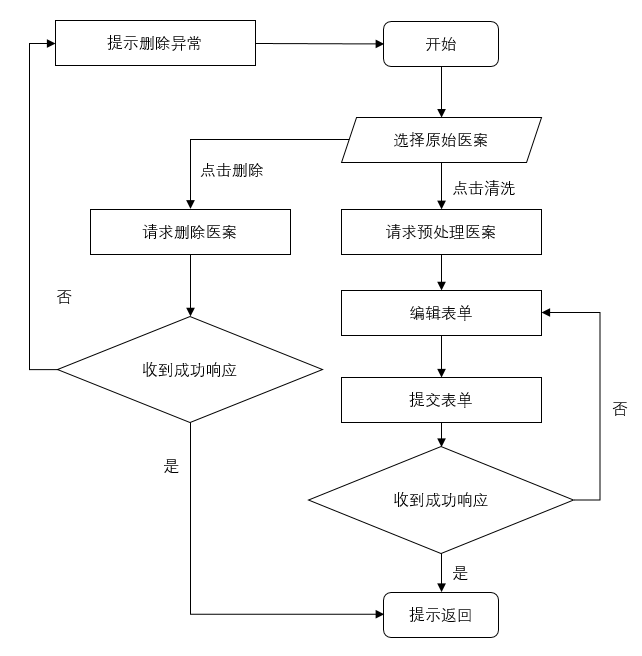


图10 医案清洗使用流程

选择医案清洗菜单后，向用户展示已录入的原始医案，选择单条医案进行清洗或删除。

清洗则弹出模态框编辑对应数据项，通过后台过滤算法对原始医案进行预处理，用字典匹配医案中包含的标准或同义词条，在模态框中显示匹配到的标准词条，用户通过4.2.1节的智能提示功能进行进一步清洗；如果当前医案已有清洗记录，则后台直接返回已清洗结果。

删除则从数据库中删除对应的原始医案，如果该医案已经清洗，则标准化的医案也一起删除。

医案清洗的具体实现细节设计如下：

a) 前端设计和4.4.1节基本一致，新的功能点在于分页组件layui的laypage组件使用，其能够分页请求后端列表，使前端显示内容控制在合理的范围内，同时支持选择页数跳转，其具体代码如下：

1. layui.use(['laypage', 'layer'], **function** () {
2. **var** laypage = layui.laypage
3. , layer = layui.layer;
4. laypage.render({
5. elem: 'cc\_page'
6. , layout: ['count', 'prev', 'page', 'next', 'refresh', 'skip']
7. , count: {{ count }}
8. , curr: {{ index}}
9. , limit: 10
10. , theme: '#1E9FFF'
11. , jump: **function** (obj, first) {
12. **if** (!first) {
13. show\_clean\_table(obj.curr);
14. }
15. }
16. });
17. });

b) 后端新的功能点主要是原始医案标准化过滤函数的编写，其逻辑就是拆分数据项依次匹配字典，匹配不到则舍弃，匹配到则替换成标准表述，具体代码如下**：**

1. **def** filter(main\_sym, add\_sym, tongue, pulse, syndrome):
2. res = [[], [], [], [], []]
3. sym = Symptom.query.all()
4. ton = Tongue.query.all()
5. pul = Pulse.query.all()
6. syn = Syndrome.query.all()
7. syms = dict(zip([i.sym\_name **for** i **in** sym], sym))
8. tons = dict(zip([i.ton\_name **for** i **in** ton], ton))
9. puls = dict(zip([i.pul\_name **for** i **in** pul], pul))
10. syns = dict(zip([i.syn\_name **for** i **in** syn], syn))
12. **for** i **in** main\_sym.split(','):
13. **if** i **in** syms:
14. std = i **if** syms[i].std\_id == 0 **else** Symptom.query.filter(sym\_id=syms[i].std\_id).first().sym\_name
15. **if** std **not** **in** res[0]:
16. res[0].append(std)
17. **for** i **in** add\_sym.split(','):
18. **if** i **in** syms **and** i **not** **in** res[0]:
19. std = i **if** syms[i].std\_id == 0 **else** Symptom.query.filter\_by(sym\_id=syms[i].std\_id).first().sym\_name
20. **if** std **not** **in** res[0]:
21. res[1].append(std)
22. **for** i **in** tongue.split(','):
23. **if** i **in** tons:
24. std = i **if** tons[i].std\_id == 0 **else** Tongue.query.filter\_by(ton\_id=tons[i].std\_id).first().ton\_name
25. **if** std **not** **in** res[0]:
26. res[2].append(std)
27. **for** i **in** pulse.split(','):
28. **if** i **in** puls:
29. std = i **if** puls[i].std\_id == 0 **else** Pulse.query.filter\_by(pul\_id=puls[i].std\_id).first().pul\_name
30. **if** std **not** **in** res[0]:
31. res[3].append(std)
32. **for** i **in** syndrome.split(','):
33. **if** i **in** syns:
34. std = i **if** syns[i].std\_id == 0 **else** Syndrome.query.filter\_by(syn\_id=syns[i].std\_id).first().syn\_name
35. **if** std **not** **in** res[0]:
36. res[4].append(std)
37. return res

### 4.2.3字典维护

该功能的作用主要是用户对症状、舌象、脉象和证型等数据项定义词条，设置该词条对应的缩写与标准词条，标准词条必须已存在且不可重复。

对于已经建立的词条，可以通过选择单个词条进行修改或删除。已定义好的词条能够用于原始医案的自动过滤和智能提示。

字典维护的主要功能有增删改查、分页和自动完成等类似功能的实现与上两节大同小异，不再作具体说明。

症状字典维护、舌象字典维护、脉象字典维护、证型字典维护的结构和功能完全一致，从菜单选择该功能即可显示字典维护项列表，点击新增填写表单即可插入字典项，选中列表中的字典项方可进行修改、删除字典项操作。

字典维护功能的使用流程设计如图11所示：

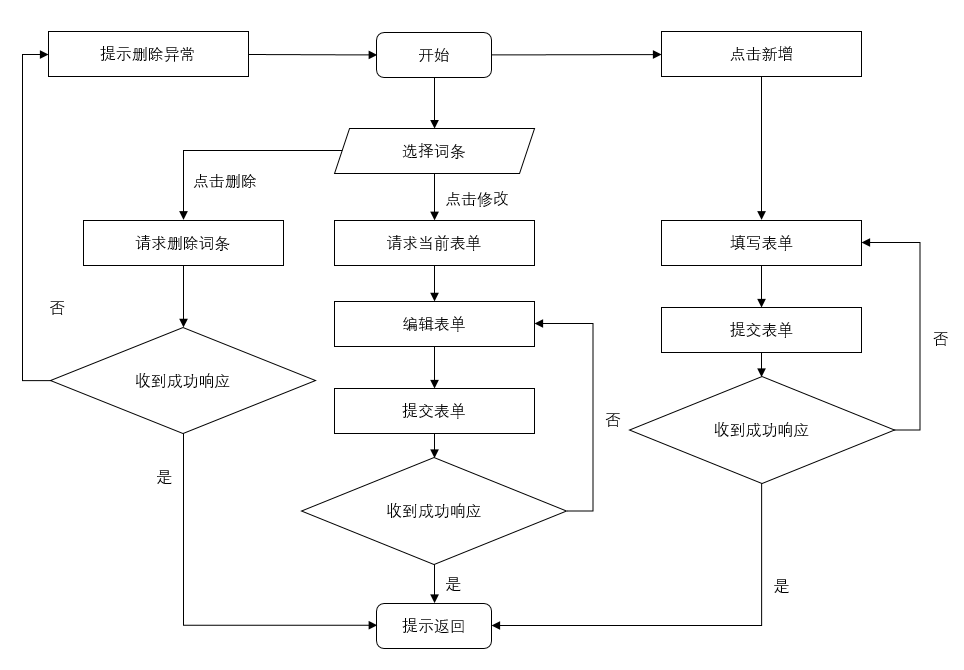


图11 字典维护使用流程

### 4.2.4量化诊断

量化诊断的使用流程设计如下：

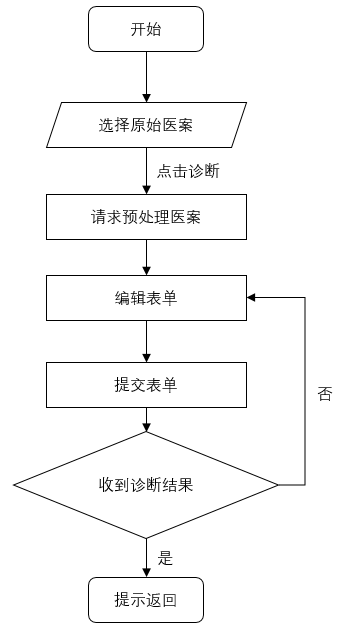


图12 量化诊断使用流程

向用户展示已录入的原始医案，选择单条医案进行量化诊断。

在量化诊断模态框中，用户将看到原始医案已标准化或已过滤的数据项，编辑之后可提交给服务器进行量化诊断，服务器依次使用AdaBoost、随机森林、卷积神经网络和谱聚类对医案进行分类，返回分类结果。

量化诊断的前后端类似功能在上述均已出现，诊断算法的实现参考第3章内容。

### 4.2.5结果展示

结果展示功能在选择该菜单后向用户展示当前已维护字典项在所有医案中的关系分布和统计特征，没有其他操作。

用户通过关系网络的展示可以直观看到所有医案中各词条的相关性与关联程度，关联度越高，在关系网络中距离越近，关联度越低则越远，太低则被后台过滤掉，不会显示。

用户通过柱状图可以直观看到各数据项中的高频词条及其频数。

用户通过饼状图可以直观看到各数据项中高频词条所占比率及其频数。

结果展示功能的具体设计细节如下：

a) 前端难点在于Echarts.js和后端数据的结合展示，实现逻辑是页面加载成功后，使用ajax请求图表数据，在回调函数里配置Echarts组件，数据请求失败则悬框提示，选项卡切换时需要对Echarts组件进行resize刷新，具体代码如下：

1. graph.showLoading();
2. $.ajax({
3. url: 'get\_graph',
4. type: 'GET',
5. success: **function** (data) {
6. **if** (data['code'] != '-1') {
7. graph.hideLoading();
8. xml = echarts.dataTool.gexf.parse(data);
9. **var** categories = [{name: '症状'}, {name: '舌象'}, {name: '脉象'}, {name: '证型'}];
10. xml.nodes.forEach(**function** (node) {
11. node.itemStyle = **null**;
12. node.value = node.symbolSize;
13. node.symbolSize = Math.log2(node.symbolSize);
14. node.label = {
15. normal: {
16. show: node.symbolSize > 20
17. }
18. };
19. node.category = node.attributes.modularity\_class;
20. });
21. **var** graph\_option = {
22. title: {
23. text: "关系网络",
24. top: "top",
25. left: "center"
26. },
27. tooltip: {},
28. animationDuration: 1500,
29. animationEasingUpdate: 'quinticInOut',
30. legend: [{
31. data: categories.map(**function** (a) {
32. **return** a.name;
33. }),
34. bottom: "10%",
35. left: "center"
36. }],
37. series: [
38. {
39. name: "关系网络",
40. type: "graph",
41. layout: "force",
42. force: {
43. repulsion: 1000
44. },
45. draggable: **true**,
46. data: xml.nodes,
47. links: xml.links,
48. categories: categories,
49. label: {
50. normal: {
51. position: 'right',
52. formatter: '{b}'
53. }
54. },
55. lineStyle: {
56. normal: {
57. color: 'source',
58. curveness: 0.1
59. }
60. }
61. }
62. ]
63. };
64. graph.setOption(graph\_option);
65. } **else** {
66. toastr.options.timeOut = "2000";
67. toastr.options.closeButton = **true**;
68. toastr['error']('<small>图表初始化失败</small>', '', {positionClass: 'toast-bottom-center'});
69. }
70. }
71. });

b) 后端实现难点在于Echarts关系网络的描述文件的写入，实现思路是按照官方模板编写脚本从数据库中读取节点的属性，统计边的属性，替换掉模板中的占位符，具体代码如下：

1. **def** create\_gexf():
2. nodes = {}
3. edges = {}
4. all\_nodes = []
5. id\_nodes = 0
6. with open('data/sym.txt') as file:
7. **for** i **in** range(100):
8. line = file.readline()
9. words = line.split('\t')
10. all\_nodes.append(words[0])
11. nodes[words[0]] = {'id':id\_nodes, 'name':words[0], 'freq':int(words[-1]), 'category':'0'}
12. id\_nodes += 1
13. with open('data/ton.txt') as file:
14. **for** i **in** range(10):
15. line = file.readline()
16. words = line.split('\t')
17. all\_nodes.append(words[0])
18. nodes[words[0]] = {'id':id\_nodes, 'name': words[0], 'freq': int(words[-1]), 'category': '1'}
19. id\_nodes += 1
20. with open('data/pul.txt') as file:
21. **for** i **in** range(10):
22. line = file.readline()
23. words = line.split('\t')
24. all\_nodes.append(words[0])
25. nodes[words[0]] = {'id':id\_nodes, 'name': words[0], 'freq': int(words[-1]), 'category': '2'}
26. id\_nodes += 1
27. with open('data/syn.txt') as file:
28. **for** i **in** range(20):
29. line = file.readline()
30. words = line.split('\t')
31. all\_nodes.append(words[0])
32. nodes[words[0]] = {'id':id\_nodes,'name': words[0], 'freq': int(words[-1]), 'category': '3'}
33. id\_nodes += 1
35. id\_nodes = 0
36. workbook = xlrd.open\_workbook('data/65.xlsx')
37. sheet = workbook.sheet\_by\_index(1)
38. rows = sheet.nrows
39. **for** i **in** range(1, rows):
40. line = sheet.row\_values(i)[0] + ',' + sheet.row\_values(i)[1] + ',' + sheet.row\_values(i)[2] + ',' + sheet.row\_values(i)[3] +',' + sheet.row\_values(i)[4]
41. words = line.split(',')
42. std = []
43. **for** j **in** words:
44. **if** j **in** all\_nodes **and** j **not** **in** std:
45. std.append(j)
46. **for** j **in** std:
47. **for** k **in** std:
48. **if** j != k:
49. key = j + '-' + k **if** j > k **else** k + '-' + j
50. **if** key **in** edges:
51. edges[key]['freq'] += 1
52. **else**:
53. edges[key] = {'id':id\_nodes, 'freq': 1}
54. id\_nodes += 1
55. **else**:
56. **continue**
58. header = '''''<?xml version="1.0" encoding="UTF-8"?>
59. <gexf xmlns="http://www.gexf.net/1.2draft" version="1.2" xmlns:viz="http://www.gexf.net/1.2draft/viz" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://www.gexf.net/1.2draft http://www.gexf.net/1.2draft/gexf.xsd">
60. <meta lastmodifieddate="2014-01-30">
61. <creator>Gephi 0.8.1</creator>
62. <description></description>
63. </meta>
64. <graph defaultedgetype="undirected" mode="static">
65. <attributes class="node" mode="static">
66. <attribute id="modularity\_class" title="Modularity Class" type="integer"></attribute>
67. </attributes>
68. <nodes>
69. '''
71. body = ''''' </nodes>
72. <edges>
73. '''
75. end = ''''' </edges>
76. </graph>
77. </gexf>
78. '''
80. node\_model = ''''' <node id="点编号" label="标签">
81. <attvalues>
82. <attvalue for="modularity\_class" value="分类"></attvalue>
83. </attvalues>
84. <viz:size value="大小"></viz:size>
85. </node>
86. '''
88. edge\_model = ''''' <edge id="边编号" source="源点" target="宿点" value="边长">
89. </edge>
90. '''
91. with open("../../templates/graph.gexf", "w", encoding='UTF-8') as file:
92. file.write(header)
93. added = []
94. **for** key, value **in** edges.items():
95. **if** value['freq'] > 300:
96. **if** key.split('-')[0] **not** **in** added:
97. file.write(node\_model.replace('点编号', str(nodes[key.split('-')[0]]['id'])).replace('标签', nodes[key.split('-')[0]]['name']).replace('分类', nodes[key.split('-')[0]]['category']).replace('大小', str(nodes[key.split('-')[0]]['freq'])))
98. added.append(key.split('-')[0])
99. **if** key.split('-')[1] **not** **in** added:
100. file.write(node\_model.replace('点编号', str(nodes[key.split('-')[1]]['id'])).replace('标签', nodes[key.split('-')[1]]['name']).replace('分类', nodes[key.split('-')[1]]['category']).replace('大小', str(nodes[key.split('-')[1]]['freq'])))
101. added.append(key.split('-')[1])
102. file.write(body)
103. **for** key, value **in** edges.items():
104. **if** value['freq'] > 300:
105. file.write(edge\_model.replace('边编号', str(value['id'])).replace('源点', str(nodes[key.split('-')[0]]['id'])).replace('宿点', str(nodes[key.split('-')[1]]['id'])).replace('边长', str(math.ceil(100.0/math.log(value['freq'],2)))))
106. file.write(end)

### 4.2.6算法训练

算法训练的使用流程设计如图13所示：

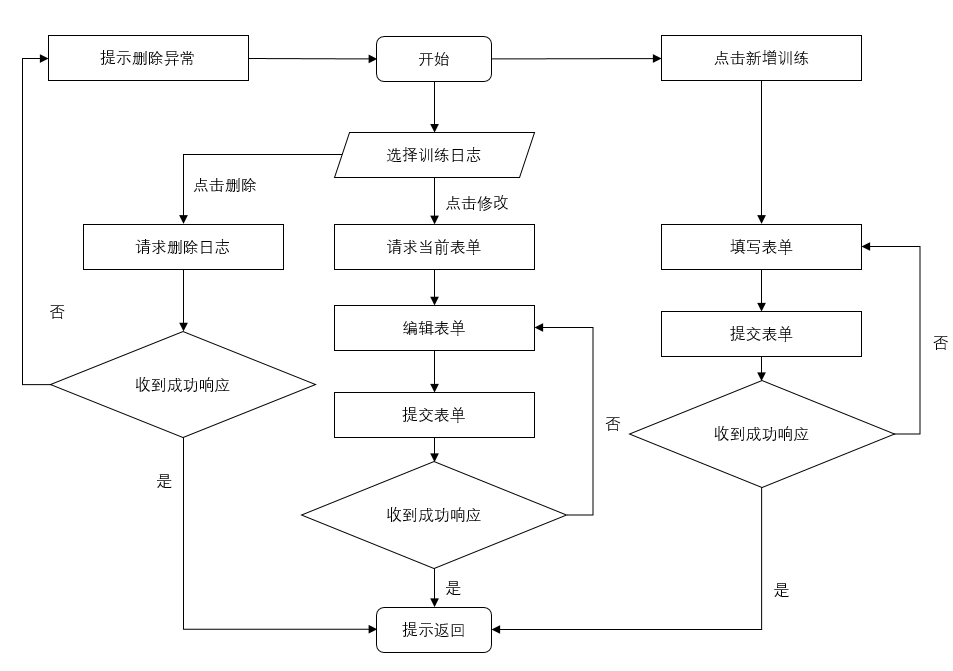


图13 算法训练使用流程

选择该菜单后，用户记录算法和模型的训练过程，包括使用到的参数和评估得到的各种算法指标，包括精确度、召回率和F1值等，以及训练的时间。

该功能打开时首先向用户展示历史训练日志，根据需要可以打开添加训练日志模态框，填写相应数据项，提交给服务器。

也可以选择单条训练日志，进行日志修改和删除。

本功能可进一步扩展为从前端选择算法模型，输入新的格式化测试数据进行在线评估；添加训练记录功能扩展为前端调用算法模型，输入模型参数，选择要评估的算法指标，操控后台自动化训练算法模型，训练完成后自动向数据库添加训练日志，而非分割训练与记录过程，手动输入日志。

算法训练功能目前已实现部分均已在上述出现类似功能，不多赘述**。**

### 4.2.7算法评估

算法评估功能没有其余的操作，选择该菜单则将训练日志可视化，用Echarts的雷达图组件，以算法指标为各维度，指标值为坐标值绘制算法评估数据图，直观展示各算法模型优劣。

算法评估功能主要是Echarts组件的使用，在结果展示部分已详细说明，不再赘述。

## 4.3数据库设计

数据库整体库表结构设计如图14所示：

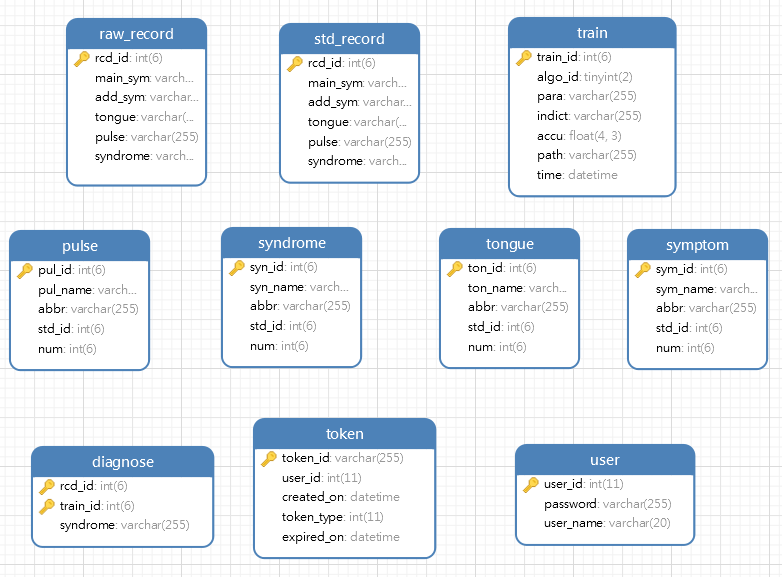


图14 库表结构图

其中raw\_record和std\_record是原始医案和标准医案表;train是算法训练表；pulse、syndrome、tongue和symptom分别是脉象、证型、舌象和症状字典表；diagnose是诊断表；token和user是用户登录验证用的表。

由于数据库结构并不复杂，仅诊断表是标准医案表实体与训练表实体的联系表，本文省略该E-R图。

### 4.3.1字典表

字典表分为症状字典表、舌象字典表、脉象字典表和证型字典表，其结构完全一致，仅记录内容不同。其作用是记录和统计各特征和标签及其标准化表达。

其结构为整数型主键id，字符串型名称，字符串型缩写、整数型标准词条id，以及整数型频数。

SQL语句依次如下：

1. **CREATE** **TABLE** `symptom` (
2. `sym\_id` **int**(6) NOT NULL AUTO\_INCREMENT COMMENT '主症、次症、病史编号',
3. `sym\_name` **varchar**(255) NOT NULL COMMENT '症状名称',
4. `abbr` **varchar**(255) NOT NULL COMMENT '拼音缩写',
5. `std\_id` **int**(6) NOT NULL COMMENT '同义标准症状编号，若是自身则为0',
6. `num` **int**(6) **DEFAULT** NULL COMMENT '出现次数',
7. **PRIMARY** **KEY** (`sym\_id`)
8. ) ENGINE=InnoDB AUTO\_INCREMENT=1973 **DEFAULT** CHARSET=utf8;
9. **CREATE** **TABLE** `tongue` (
10. `ton\_id` **int**(6) NOT NULL AUTO\_INCREMENT COMMENT '舌象编号',
11. `ton\_name` **varchar**(255) NOT NULL COMMENT '舌象名称',
12. `abbr` **varchar**(255) NOT NULL COMMENT '拼音缩写',
13. `std\_id` **int**(6) NOT NULL COMMENT '同义标准舌象编号，若是自身则为0',
14. `num` **int**(6) **DEFAULT** NULL COMMENT '出现次数',
15. **PRIMARY** **KEY** (`ton\_id`)
16. ) ENGINE=InnoDB AUTO\_INCREMENT=66 **DEFAULT** CHARSET=utf8;
17. **CREATE** **TABLE** `pulse` (
18. `pul\_id` **int**(6) NOT NULL AUTO\_INCREMENT COMMENT '脉象编号',
19. `pul\_name` **varchar**(255) NOT NULL COMMENT '脉象名称',
20. `abbr` **varchar**(255) NOT NULL COMMENT '拼音缩写',
21. `std\_id` **int**(6) NOT NULL COMMENT '同义标准脉象编号，若是自身则为0',
22. `num` **int**(6) **DEFAULT** NULL COMMENT '出现次数',
23. **PRIMARY** **KEY** (`pul\_id`)
24. ) ENGINE=InnoDB AUTO\_INCREMENT=68 **DEFAULT** CHARSET=utf8;
25. **CREATE** **TABLE** `syndrome` (
26. `syn\_id` **int**(6) NOT NULL AUTO\_INCREMENT COMMENT '证型编号',
27. `syn\_name` **varchar**(255) NOT NULL COMMENT '证型名称',
28. `abbr` **varchar**(255) NOT NULL COMMENT '拼音缩写',
29. `std\_id` **int**(6) NOT NULL COMMENT '同义标准证型编号，若是自身则为0',
30. `num` **int**(6) **DEFAULT** NULL COMMENT '出现次数',
31. **PRIMARY** **KEY** (`syn\_id`)
32. ) ENGINE=InnoDB AUTO\_INCREMENT=812 **DEFAULT** CHARSET=utf8;

### 4.3.2医案表

医案表分为原始医案表和标准医案表，其结构完全一致，仅记录内容不同。其作用是为算法模型的训练和诊断提供样本。

其结构为整数型主键id，字符串型主症字段，字符串型次症字段，字符串型舌象字段、字符串型脉象字段、以及字符串型证型字段。

SQL语句依次如下：

1. **CREATE** **TABLE** `raw\_record` (
2. `rcd\_id` **int**(6) NOT NULL AUTO\_INCREMENT COMMENT '原始医案编号',
3. `main\_sym` **varchar**(255) **DEFAULT** '' COMMENT '主症',
4. `add\_sym` **varchar**(255) **DEFAULT** '' COMMENT '次症',
5. `tongue` **varchar**(255) **DEFAULT** '' COMMENT '舌象',
6. `pulse` **varchar**(255) **DEFAULT** '' COMMENT '脉象',
7. `syndrome` **varchar**(255) **DEFAULT** '' COMMENT '证型',
8. **PRIMARY** **KEY** (`rcd\_id`)
9. ) ENGINE=InnoDB AUTO\_INCREMENT=1363 **DEFAULT** CHARSET=utf8;
10. **CREATE** **TABLE** `std\_record` (
11. `rcd\_id` **int**(6) NOT NULL AUTO\_INCREMENT COMMENT '标准医案编号',
12. `main\_sym` **varchar**(255) **DEFAULT** '' COMMENT '主症',
13. `add\_sym` **varchar**(255) **DEFAULT** '' COMMENT '次症',
14. `tongue` **varchar**(255) **DEFAULT** '' COMMENT '舌象',
15. `pulse` **varchar**(255) **DEFAULT** '' COMMENT '脉象',
16. `syndrome` **varchar**(255) **DEFAULT** '' COMMENT '证型',
17. **PRIMARY** **KEY** (`rcd\_id`)
18. ) ENGINE=InnoDB AUTO\_INCREMENT=1206 **DEFAULT** CHARSET=utf8;

### 4.3.3算法表

算法表用于记录算法模型训练日志。

其结构为整数型主键id，整数型算法id，字符串型参数、字符串型指标，浮点型准确率和日期时间类型训练时间。

SQL语句如下：

1. **CREATE** **TABLE** `train` (
2. `train\_id` **int**(6) NOT NULL AUTO\_INCREMENT COMMENT '训练编号',
3. `algo\_id` tinyint(2) NOT NULL COMMENT '算法编号：cnn--1，随机森林--2，adaboost--3，谱聚类--4',
4. `para` **varchar**(255) **DEFAULT** NULL COMMENT '算法参数，json格式，如{alpha:0.1,k:50}',
5. `indict` **varchar**(255) **DEFAULT** NULL COMMENT '算法指标，json格式，如{F1:0.1}',
6. `accu` **float**(4,3) NOT NULL COMMENT '准确率',
7. `path` **varchar**(255) **DEFAULT** NULL COMMENT '模型文件路径，没有则为''''',
8. `**time**` datetime NOT NULL COMMENT '训练时间',
9. **PRIMARY** **KEY** (`train\_id`)
10. ) ENGINE=InnoDB AUTO\_INCREMENT=27 **DEFAULT** CHARSET=utf8;

### 4.3.4诊断表

诊断表用于记录算法模型对标准医案进行量化诊断的结果记录。

其结构为整数型标准医案id，整数型训练的算法id和字符串型标准证型。

SQL语句如下：

1. **CREATE** **TABLE** `diagnose` (
2. `rcd\_id` **int**(6) NOT NULL COMMENT '标准医案编号',
3. `train\_id` **int**(6) NOT NULL COMMENT '训练的算法编号',
4. `syndrome` **varchar**(255) NOT NULL COMMENT '标准证型',
5. **PRIMARY** **KEY** (`rcd\_id`,`train\_id`)
6. ) ENGINE=InnoDB **DEFAULT** CHARSET=utf8;

## 4.4系统功能展示

本部分将以系统的使用流程为顺序，展示本系统的功能使用，以文字解释操作，以图片展示相应的操作结果，相同的操作结果图片将会指明具体图片编号以省略，具体操作流程与结果如下：

打开浏览器，输入localhost:5000，进入登录界面，如图15所示：



图15 登录界面

输入帐号、密码，进入主页，如图16所示：



图16 主页

### 4.4.1医案录入

依次点击量化诊断模块和医案录入菜单，进入医案录入界面，输入拼音缩写，选择相应词条，如图17所示：



图17 录入与智能提示

点击提交，提示录入成功，否则提示失败。文本录入、数据库录入、表格录入类似。

### 4.4.2医案清洗

选择样本采集模块的医案清洗菜单，显示原始医案列表，如图18所示：



图18 医案清洗页面

选择列表项，点击清洗得到预处理的标准医案表单，如图19所示：

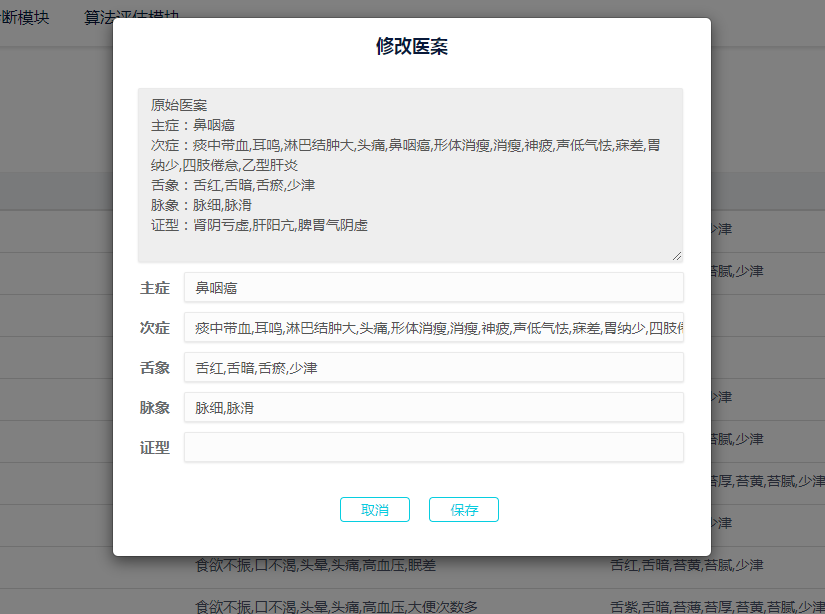


图19 修改医案模态框

利用智能提示，填写表单，点击保存，出现成功操作提示。点击删除按钮，确认删除，得到成功删除提示。

### 4.4.3字典维护

选择字典维护菜单的症状字典维护项，如图20所示：



图20 字典首页

点击新增按钮，弹出模态框，如图21所示：



图21 新增症状

重复插入词条，则得到警告提示，插入正确项，得到成功提示。修改症状、新增症状、删除字典类似。其他字典操作完全一样。

### 4.4.4量化诊断

选择量化诊断模块的量化诊断菜单，显示原始医案列表同图18。选择医案，点击诊断，如图22所示：

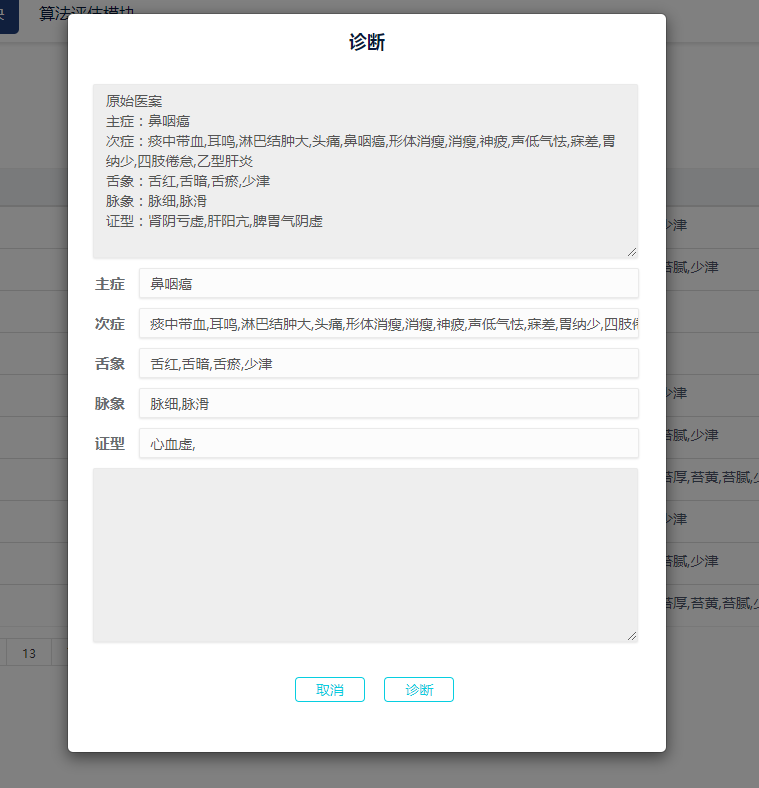


图22 量化诊断模态框

根据需要编辑表单项，点击诊断按钮，得到诊断结果，如图23所示：

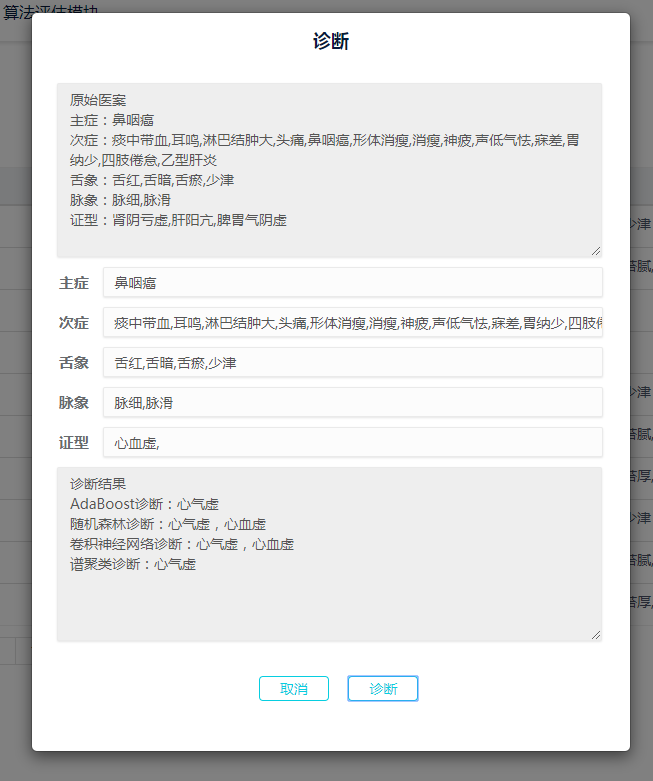


图23 量化诊断结果

### 4.4.5结果展示

选择结果展示菜单，点击关系网络选项卡，如图24所示：

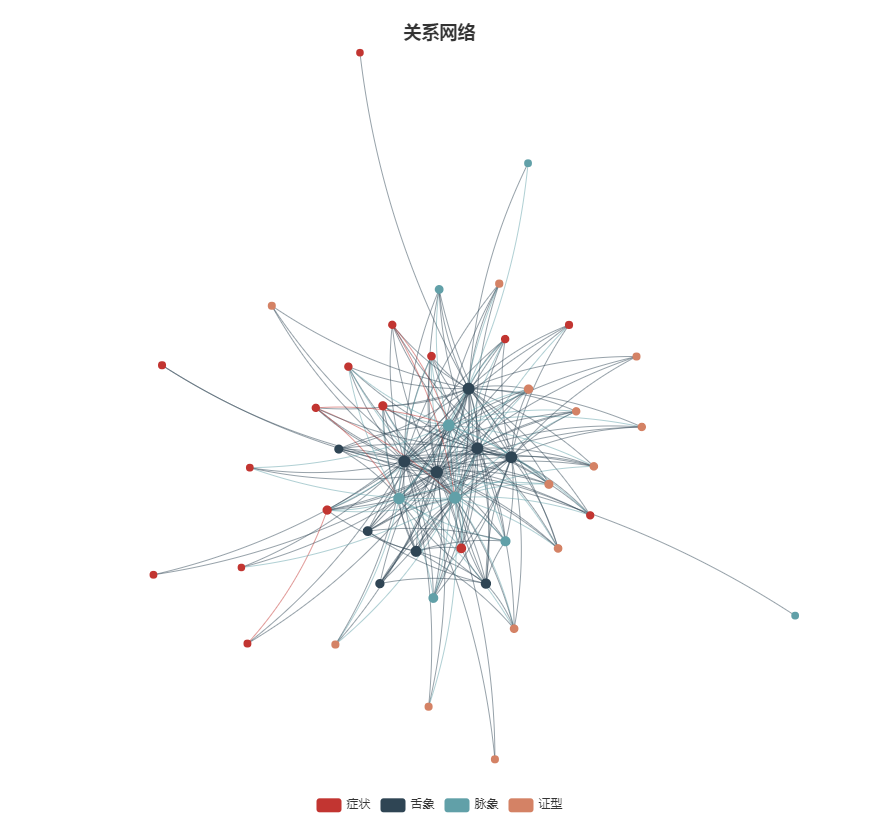


图24 关系网络

切换到柱图分析选项卡，显示高频特征柱状图；切换到饼图选项卡，显示各症状比例。

### 4.4.6算法训练

选择算法评估的训练日志菜单，显示已录入算法训练日志列表，如图25所示：

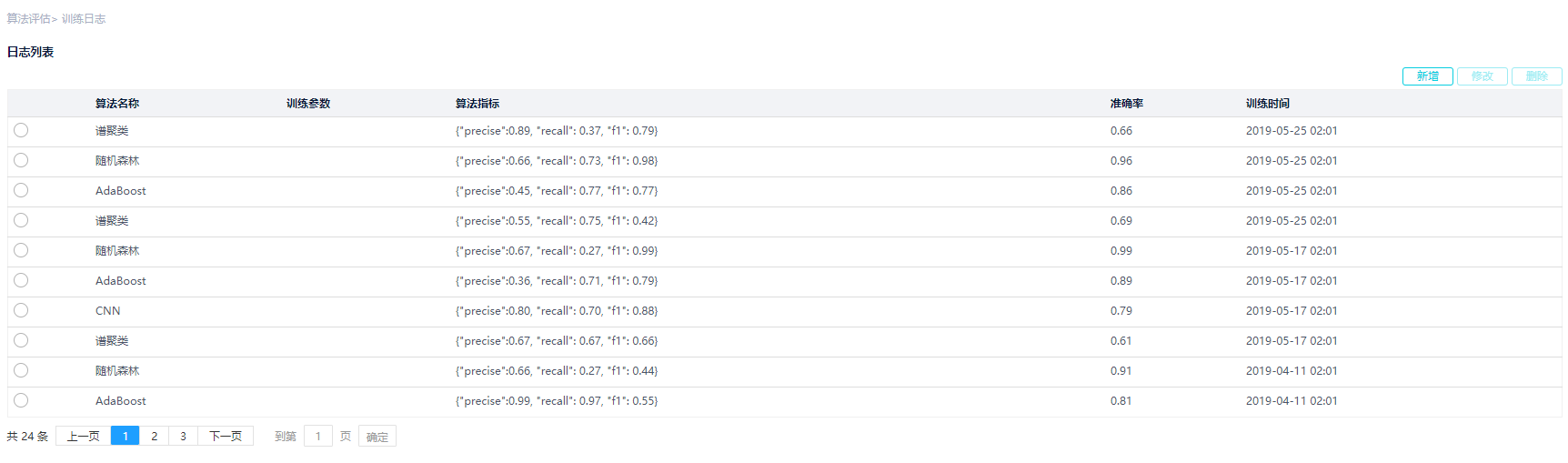


图25 训练日志主页

点击新增日志，弹出模态框，根据提示填写表单。

### 4.4.7算法评估

选择算法评估菜单，切换算法，显示指定算法性能指标，如图26所示：

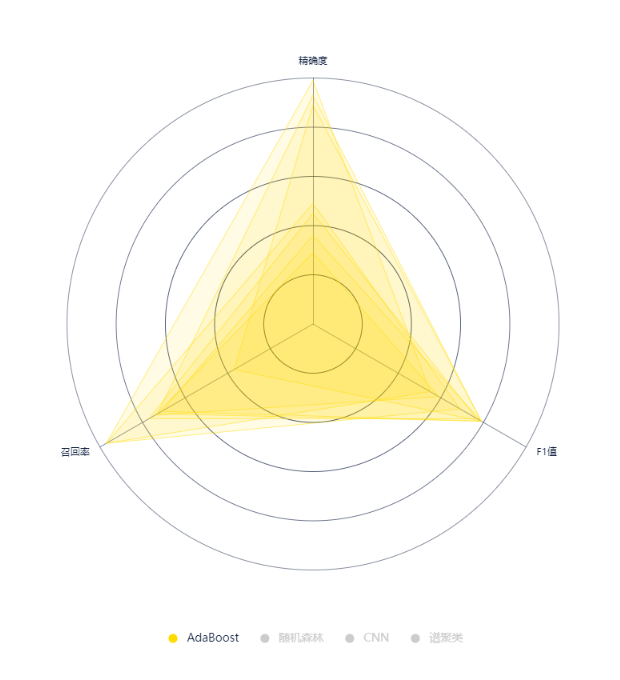


图26 算法评估

选择模型演进菜单，显示准确率变化趋势，如图27所示：

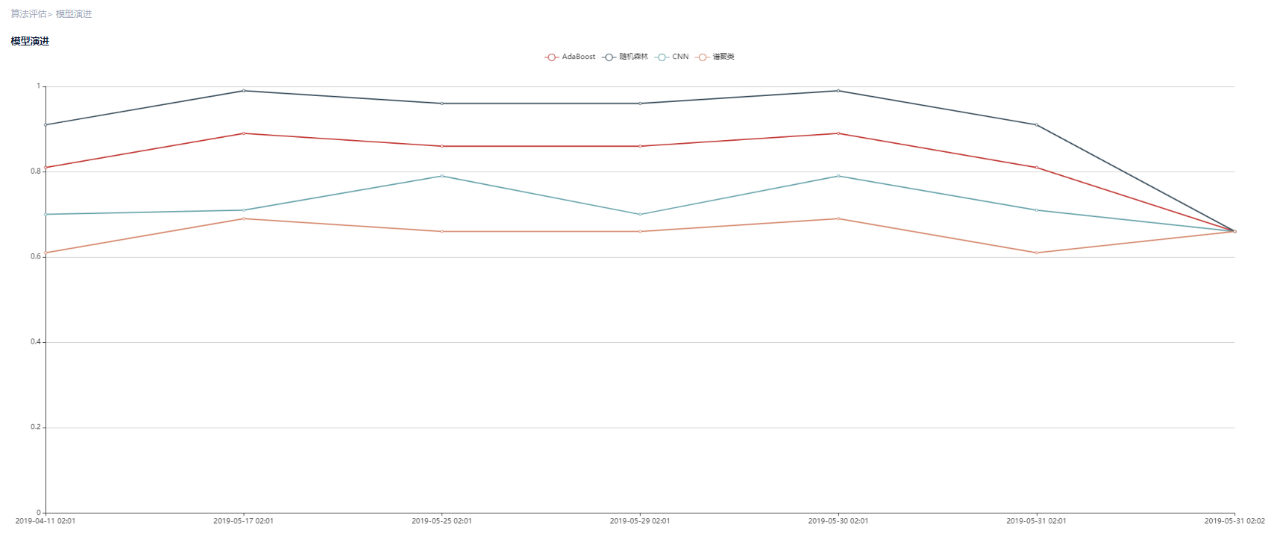


图27 模型演进

# 5.总结与展望

## 5.1系统优点与特色

总结起来，本系统有以下优点：

a) 界面风格简洁，布局美观。采用上导航栏、左菜单栏、右下功能区的区块布局，块内采用单一色调，配以提示性Logo，风格统一，简洁美观，使用户操作一目了然，轻松上手，不存在任何使用障碍；

b) 支持文本、表格、表单、数据库等多种方式录入医案，兼容性强；能够实现实时智能输入提示，减少用户操作，提高录入效率；支持维护用户字典，定义常用输入选项，可扩展性强；

c) 诊断及算法评估功能采用柱状图、折线图、饼状图、雷达图、关系网络图等图表形式展示，数据结果直观化，降低用户理解负担和专业水平要求，即便没有专业统计分析知识的一般用户也能理解本系统分析结果。

有以下特点**：**

a) 创新性，将卷积神经网络应用于量化诊断，并提出相关性特征邻接排列的想法，这是本系统独一无二的创新；

b) 可操作性，本系统在量化诊断方法研究基础上，完成了从数据采集、清洗、算法设计、训练和调参等一系列实践，并将整套流程集成于系统功能；

c) 可扩展性，系统开放了算法评估模块的参数设置和指标设置功能，待系统进一步完善，可进一步实现前端调参评估、后端自动训练的扩展功能。

## 5.2系统缺点与不足

总结起来，系统有以下缺点尚待改进：

a) 不支持批量诊断，量化诊断只能选中单条医案逐个诊断；

b) 不支持基于单次录入的统计分析，结果展示部分的统计图表是基于所有医案的，并不具有当前批次录入的针对性；

c) 算法训练的训练与日志功能是分离的，需要算法训练者手动记录训练日志而不是自动创建训练任务并记录。

d) 各算法的准确率尚待提高和多标签的支持有待完善**。**

## 5.3系统完善思路与展望

针对以上不足，本设计提供以下完善思路：

a) 不支持批量诊断，可以利用诊断表将标准医案中所有未找到诊断记录的医案全部诊断并返回新增诊断表供用户查看；

b) 不支持基于单次录入的统计分析，可以在本次会话中存入一个本次新增医案id列表，所有的统计项基于该列表计算；

c) 算法训练的训练与日志功能是分离的，后端针对算法编写接口即可。

d) 各算法的准确率尚待提高和多标签的支持有待完善，继续清洗原始医案并添加同义词典，提高样本质量，尝试更多支持多分类多标签的算法模型。

在以上功能改善的情况下，本系统有望成为在科研、生产环境中量化诊断方面完善且强力的工具。

#### 结束语

通过本次毕业设计实践，我对机器学习算法有了更深入的学习，认识到算法实现与调参过程中细致入微的问题，以及如何解决这些问题。在此过程中通过阅读技术文档、研究相关论文、在技术社区讨论解决方案甚至推敲开源代码都使我在上下求索中获益匪浅，不仅仅是技术与能力的提升，也是思维习惯和方式的规范。

以前总是以正确实现功能为目标，在追求更美观、更好用的系统过程中，遇到了企业级项目应有的挑战，而解决它们让我掌握了很多改善系统功能和界面的组件，如Bootstrap、Echarts，xlrd和xpinyin等，运用这些工具让我体会到技术的魅力和学习的价值。“可以更好”，将是我的热情所在。

由于医案样本清洗和标准化的原因，算法的研究尚且肤浅而停留于应用层面，更加细致的调参优化过程尚未做到；系统开发也因此进行得仓促，一些表单校验和异常抛出尚未实现，尤其自动训练算法模型功能，是本系统中的核心功能。后面将会在此基础上进行深入研究和完善。

###### 致谢

在毕业设计过程中，我受到过很多人的帮助，在此铭记，以示感谢。

首先感谢我的导师，杨涛老师。不仅是毕业设计，在整个大学学习过程中，杨老师都是良师益友。从大二开始，杨老师一直引导我们去学习专业领域前沿的技术和算法，分享学习的方法和资源，鼓励我们选定研究内容钻研下去，在实践中提升科研能力和技术水平。杨老师屡屡指导我们参加各种竞赛和项目，和我们一起撰写和修改材料，讨论研究思路，学习别人的心得和方法，组织我们分享所学所获。我在毕业设计中所用的研究思维、开发习惯和专业技能皆来源于此。

其次感谢中医专业的两名同学。医案清洗和标准化的工作繁复而需要专业知识，她们在自己学习之余耐心地逐条清洗医案数据，整理同义词典，拆分复合证型，每周提供标准化数据，帮助我的算法研究工作。

另外感谢工作室的四名学弟学妹，虽然在开发过程中没用到他们的工作成果，但在和他们合作过程中学习到了他们对于功能实现的想法，有的比我原有的想法更有效且简单。在组织他们协同工作的过程中，我学会了搭建gitlab平台。

##### 参考文献

[1] 吴承玉, 徐征, 骆文斌, 等. 五脏系统病位特征与基础证的研究[J]. 南京中医药大学学报, 2011(3):201-203.

[2] 刘保延. 真实世界的中医临床科研范式[J]. 中医杂志, 2013(06):451-455.

[3] Zhou Z H, Yang Y, Wu X D, Kumar V.The Top Ten Algo-rithms in Data Mining.New York, USA:CRC Press, 2009, 127149

[4] 栾丽华,吉根林.决策树分类技术研究[J].计算机工程,2004(09):94-96+105.

[5] 李彦冬,郝宗波,雷航.卷积神经网络研究综述[J].计算机应用,2016,36(09):2508-2515+2565.

[6] 蔡晓妍,戴冠中,杨黎斌.谱聚类算法综述[J].计算机科学,2008(07):14-18.

[7]杨学兵,张俊.决策树算法及其核心技术[J].计算机技术与发展,2007(01):43-45.

[8] Wang Z , Poon J , Sun S , et al. CNN based Multi-Instance Multi-Task Learning for Syndrome Differentiation of Diabetic Patients[J]. 2018.

[9] 吕鸿蒙,赵地,迟学斌.基于增强AlexNet的深度学习的阿尔茨海默病的早期诊断[J].计算机科学,2017,44(S1):50-60.

[10] 高琰,谷士文,唐琎,蔡自兴.机器学习中谱聚类方法的研究[J].计算机科学,2007(02):201-203.