**审定成绩：**

|  |  |  |  |
| --- | --- | --- | --- |
| **重庆邮电大学**  **毕业设计（论文）** | | | |
|  | | | |
|  | **中文题目** | **关联分类规则挖掘软件设计与实现** |  |
|  |  |
| **英文题目** | **Design and implementation of association classification rule mining software** |
|  |  |
| **学院名称** | **计算机科学与技术学院/人工智能学院** |
| **学生姓名** | **文锐锴** |
| **专 业** | **智能科学与技术** |
| **班 级** | **04052001** |
| **学 号** | **2020211608** |
| **指导教师** | **周应华 副教授** |
| **答 辩 组**  **负 责 人** | **姓名 职称** |
|  | |

**2024 年 5 月**

**重庆邮电大学教务处制**

**计算机科学与技术/人工智能**学院本科毕业设计(论文)诚信承诺书

本人郑重承诺：

我向学院呈交的论文《 关联分类规则挖掘软件设计与实现 》，是本人在指导教师的指导下，独立进行研究工作所取得的成果。除文中已经注明引用的内容外，本论文不含任何其他个人或集体已经发表或撰写过的作品成果。对本文的研究做出重要贡献的个人和集体，均已在文中以明确方式标明并致谢。本人完全意识到本声明的法律结果由本人承担。

年级 2020

专业 智能科学与技术

班级 04052001

承诺人签名 文锐锴

年 月 日

学位论文版权使用授权书

本人完全了解重庆邮电大学有权保留、使用学位论文纸质版和电子版的规定，即学校有权向国家有关部门或机构送交论文，允许论文被查阅和借阅等。本人授权重庆邮电大学可以公布本学位论文的全部或部分内容，可编入有关数据库或信息系统进行检索、分析或评价，可以采用影印、缩印、扫描或拷贝等复制手段保存、汇编本学位论文。

（注：保密的学位论文在解密后适用本授权书。）

|  |  |
| --- | --- |
| 学生签名： | 指导老师签名： |
| 日期： 年 月 日 | 日期： 年 月 日 |

摘要

传统的关联规则挖掘算法意在从海量数据中挖掘出有意义的规则，以描述数据之间存在的相互联系。而分类问题则是利用分类器，根据样本的属性对样本类别进行预测。对于前者，所能发现的规则是无法事先知晓的；对于后者，目标类别的最终判定结果只能有一个。本文结合这两种算法，设计并实现了一个挖掘关联分类规则的软件。

本文首先选取并介绍了CBA-M1算法、CBA-M2算法、APR算法和CMAR算法的基本原理，然后通过需求分析、确定软件基本功能并对软件进行设计，最后基于Vue3和Flask框架的前后端分离架构进行软件开发，算法主体使用Python语言实现。在系统设计方面，本文采用软件工程原理，构建了一个模块化的系统架构，包括数据预处理、类关联规则提取和分类器构造等关键模块。此外，实现的软件简便易用，能够利用多种算法学习得到分类规则，然后使用这些规则对数据进行分类以及对比分析不同算法的性能。

实验部分，使用UCI公开数据集对软件进行了测试，并对不同算法的挖掘结果和效率进行了对比分析。结果表明，不同算法在准确率、时间开销、规则数目上各有优劣。其中，实验发现APR算法整体性能更好，在测试样本中分类准确率较高的同时，拥有较低的时间开销和较少的类关联规则数目；CMAR算法由于采用了高效的FP-growth算法和CR-tree数据结构，使得其在处理大规模数据集时，仍能表现出低时间开销。

**关键词：**数据挖掘，关联分类，软件实现

**Abstract**

Traditional association rule mining algorithms aim at mining meaningful rules from massive data to describe the interrelation between data. The classification problem is to use the classifier to predict the sample category according to the attribute of the sample. For the former, the rules that can be discovered cannot be known in advance; For the latter, there can be only one final decision result for the target category. Combining these two algorithms, this paper designs and implements a software for mining association classification rules.

This paper first selected and introduced the basic principles of CBA-M1 algorithm, CBA-M2 algorithm, APR algorithm and CMAR algorithm, then determined the basic functions of the software and designed the software through demand analysis, and finally developed the software based on the front-end separation architecture of Vue3 and Flask framework. The main body of the algorithm was implemented in Python language. In the aspect of system design, this paper adopts the principle of software engineering to build a modular system architecture, including key modules such as data preprocessing, class association rule extraction and classifier construction. In addition, the software is simple and easy to use. It can learn classification rules by various algorithms, and then use these rules to classify data and compare and analyze the performance of different algorithms.

In the experimental part, the software is tested using UCI public data set, and the mining results and efficiency of different algorithms are compared and analyzed. The results show that different algorithms have their advantages and disadvantages in accuracy, time cost and number of rules. The experiment shows that APR algorithm has better overall performance, higher classification accuracy in test samples, lower time cost and fewer class association rules. Because CMAR algorithm adopts efficient FP-growth algorithm and CR-tree data structure, it can still show low time cost when processing large-scale data sets.

Key words: data mining, association classification, software implementation

目录

[第1章 引言 1](#_Toc167660044)

[1.1 研究背景和意义 1](#_Toc167660045)

[1.2 国内外研究现状 2](#_Toc167660046)

[1.2.1 国内研究现状 2](#_Toc167660047)

[1.2.2 国外研究现状 3](#_Toc167660048)

[1.3 主要内容和工作安排 4](#_Toc167660049)

[第2章 关联分类规则算法 5](#_Toc167660050)

[2.1 CBA算法 5](#_Toc167660051)

[2.1.1 规则生成 5](#_Toc167660052)

[2.1.2 分类器构建的CBA-M1算法 7](#_Toc167660053)

[2.1.3 分类器构建的CBA-M2算法 7](#_Toc167660054)

[2.2 APR算法 8](#_Toc167660055)

[2.3 CMAR算法 8](#_Toc167660056)

[2.4 本章小结 9](#_Toc167660057)

[第3章 关联规则分类软件设计 10](#_Toc167660058)

[3.1 需求分析 10](#_Toc167660059)

[3.2总体设计 10](#_Toc167660060)

[3.3 详细设计 12](#_Toc167660061)

[3.3.1功能设计 12](#_Toc167660062)

[3.3.2 页面设计 16](#_Toc167660063)

[3.4本章小结 17](#_Toc167660064)

[第4章 软件实现与测试 18](#_Toc167660065)

[4.1 软件实现 18](#_Toc167660066)

[4.2软件测试 22](#_Toc167660067)

[4.3 本章小结 23](#_Toc167660068)

[第5章 算法分析实验 25](#_Toc167660069)

[5.1 关联分类算法的准确率比较 25](#_Toc167660070)

[5.2 关联分类算法的时间开销比较 26](#_Toc167660071)

[5.3 关联分类算法的规则数目比较 27](#_Toc167660072)

[5.4 本章小结 28](#_Toc167660073)

[第6章 总结与展望 29](#_Toc167660074)

[6.1 结论 29](#_Toc167660075)

[6.2 后续研究工作展望 30](#_Toc167660076)

[参考文献 31](#_Toc167660077)

[致谢 33](#_Toc167660078)

[附录A 文献翻译 34](#_Toc167660079)

[一、英文原文 34](#_Toc167660080)

[二、英文翻译 38](#_Toc167660081)

[附录B 42](#_Toc167660082)

[一、源程序 42](#_Toc167660083)

# 

# 第1章 引言

## 1.1 研究背景和意义

随着全球数据总量的持续攀升，数据分析与挖掘技术在各个领域的应用变得愈发重要。据国家统计局数据显示，截至2023年底，我国数据总量已达到数十ZB级别，并且这一数字预计将在未来几年继续增长。这一数据膨胀趋势使得如何有效地从海量数据中挖掘出有价值的信息，成为了一个亟待解决的问题。利用数据挖掘技术提升决策质量和效率，对于国家经济发展和社会进步具有重要意义。

在具体行业中，零售、医疗、金融等领域对数据挖掘技术的需求尤为强烈。根据前瞻产业研究院发布的《中国大数据行业市场前瞻与投资战略规划分析报告》，未来市场规模将突破七万亿元。各行业纷纷加大对数据挖掘技术的投入，以期通过数据分析提升业务洞察力和竞争优势。特别是在金融行业，通过数据挖掘技术进行风险和欺诈检测，已成为行业的常态化需求。因此，对于挖掘出海量数据中存在的潜在模式以及对数据进行分类有着很大的需求。

关联分类规则挖掘结合了关联规则挖掘和分类两种方法的优点，不仅能够发现隐藏在数据中的关联模式，还能通过这些模式对样本进行准确分类。作为数据挖掘领域的一个重要问题，人们对其进行了大量的研究，并衍生出多种不同的关联分类规则算法，其中较为经典的有CBA算法[1]、CMAR算法[2]。CBA算法包含两个部分，第一部分生成一组类关联规则而非传统的关联规则，这一部分实现的关键在于如何筛选出候选项集，以及如何对类关联规则进行规则剪枝。算法的第二部分就是训练分类器，CBA的分类器构造包含M1和M2两种算法。近几年，研究人员在CBA-M1算法的基础上进行改进，提出了基于主动剪枝规则实现的APR算法[3]。CMAR算法不像CBA算法依赖单一的类关联规则，而是利用多条类关联规则的组合进行分类。本论文选取了CBA-M1算法、CBA-M2算法、APR、CMAR算法进行实现，并整合在一个软件中，可以有效处理各类数据集。通过实验验证多种算法在不同数据集上的性能，发现该软件在不同的数据集中均有良好的分类效果。关联分类规则软件的设计与实现，为行业提供了一个强有力的分析工具，有助于提升相关行业的数据处理能力和决策水平。

## 1.2 国内外研究现状

### 1.2.1 国内研究现状

基于关联规则的分类算法在国内的数据挖掘和机器学习领域也得到了广泛的研究和应用。国内的研究团队在关联分类规则算法的基础上进行了多种创新，特别是针对特定的行业需求和数据特性进行了深入的定制和优化。这些优化措施有效地提高了算法在处理大规模数据集时的性能，同时也增强了算法在复杂数据环境下的稳定性和准确性。

关联分类算法在数据挖掘领域中得到了广泛应用和创新改进。黎文娟等人提出的基于余弦度量的关联分类算法（IACD）通过使用余弦度量来评估项集与类别之间的相关性，并通过新的规则强度对规则进行排序和剪枝，有效减少冗余规则数量，并在多个UCI数据集上显示出较高的分类准确率[4]。秦晨普等人针对传统关联分类算法的不足，如资源消耗大和规则剪枝难，提出了一种基于分类修剪的改进方案ACCP，通过分块挖掘和改进频繁项集及规则修剪过程，显著提高了分类准确率和运行效率[5]。

李家辉等人针对支持度置信度框架难以产生大量高质量规则的问题，开发了改进的关联分类算法IAMC，该算法通过多次学习提取更多高质量规则，并引入了综合置信度和补类支持度的新度量标准来提升规则质量[6]。对于未被覆盖的实例则采用决策树方法再次提取规则，以此提高了在多个UCI数据集上的分类准确率。

郭明环等人通过关联挖掘算法对图书馆读者的行为特征进行了分析，有效地划分了读者类型，该方法不仅提高了分析效率和直观性，还减少了计算成本[7]。翟悦等人提出了一种带约束的改进关联分类规则挖掘方法，通过扩展概念格结构和引入差集概念，加速了支持度和置信度的计算，并在时间和空间性能上都表现出色，满足了重用性要求[8]。

这些研究表明，关联分类算法的持续创新不仅解决了现有技术的局限性，而且在多个应用场景中展示了其强大的性能和实用性。

### 1.2.2 国外研究现状

基于关联规则的分类算法在数据挖掘和机器学习领域中有着广泛的应用，该算法可以有效地提升分类任务的准确性、效率和可靠性。目前国外研究团队在关联分类规则算法的基础上有着多种创新，针对不同领域的具体应用对算法进行了优化。L. Atzori

H. Hasanpour等人针对由于挖掘的规则集过于庞大而导致的高处理开销问题，提出了一种将Apriori算法、Harmony Search与基于分类的关联规则（CBA）算法相结合的新方法，构建了一个高效的基于规则的分类器[9]。具体而言，Harmony Search算法被用于优化规则搜索过程，使得在较大的规则集中能够快速找到最佳分类规则，从而显著提高分类器的效率和准确性。研究结果表明，这种集成方法在处理大型数据集时，不仅能显著减少计算资源的消耗，还能提升分类性能。

C. Thanajiranthorn等人摒弃传统的剪枝过程，提出了一种旨在提高分类精度的新型关联分类（AC）算法，可以直接发现数量紧凑且高效的分类规则[10]。该方法通过严格筛选规则，确保每条规则都具有较高的分类效用，从而减少规则集的冗余，提高分类器的精度和效率。实验结果显示，这种直接发现高效规则的方法不仅能够保证分类准确率，还能显著减少规则数量，使得分类模型更加简洁高效。

R. Dhanalakshmi等人提出了一种基于信息质量值的模糊影响规则的关联规则生成和分类算法，旨在提高分类效率并降低错误率[11]。该算法通过引入信息质量值评估规则的有效性，并结合模糊逻辑处理数据中的不确定性，特别适用于处理敏感数据。实验结果表明，这种方法在多个实际数据集上的分类效果优于传统方法，尤其是在处理带有噪声和不确定性的数据时，能够显著降低错误率，提高分类效率。

L. Cagliero等人通过从过去的学习者相关数据中训练分类模型来预测学生的学习成绩，提出了一种Explainable Learning Analytics解决方案[12]，该方法旨在在课程结束前预测考试成功率，并为教育工作者提供可解释的预测结果，以便进行针对性的教学干预。研究显示，这种方法不仅提高了学生成绩预测的准确性，还增强了模型的可解释性和实用性，为教育领域提供了一种新的分析工具。

这些研究不仅说明关联分类规则技术在不同领域有着广泛的适用性，同时也表明了关联分类规则算法的有效性深受研究人员的欢迎。

## 1.3 主要内容和工作安排

全文共分为6章，内容结构安排如下：

第1章是引言，引入课题的研究背景及意义。

第2章是关联规则分类算法，介绍CBA、APR和CMAR算法的实现原理。

第3章是关联规则分类软件设计，介绍软件的设计与功能实现。

第4章是关联规则分类软件实现与测试，介绍软件的实现与测试过程。

第5章是算法分析实验，借助开发的软件，使用多种数据集对算法性能分析。

第6章是总结与展望，对本文及实验内容进行总结，并提出未来的发展方向。

# 第2章 关联分类规则算法

关联分类规则算法是一种将关联规则挖掘与分类任务相结合的技术。通过在数据集中挖掘频繁项集，得到一种将属性值与特定类标签相关联的特殊规则，称为类关联规则。在分类任务中，这些类关联规则用于预测未知实例的类别，该算法的核心理念就是通过从训练集中找到强相关的规则、构建性能良好的分类器来提高分类的准确性。与其他传统的分类方法相比，关联分类规则算法具有较强的可解释性，能够帮助揭示数据中的潜在模式。

## 2.1 CBA算法

CBA算法[1]，即基于关联规则的分类算法（Classification Based on Association），它结合了关联规则挖掘和分类器构建的思想，能够从数据中发现类关联规则并进行分类预测。该算法包含两个部分，第一部分是规则生成，这一步是在Apriori算法的基础上进行修改实现的，其核心思想与Apriori算法相似，不过生成的是针对特定类的关联规则而非传统的关联规则。在这个过程中，首先识别出频繁项集，然后对这些频繁项集进行剪枝，以去除那些不满足最小支持度和最小置信度的规则，从而减少规则的数量并提高规则的质量。第二部分是训练分类器，分类器构造中包含M1和M2两种算法。其中M1算法利用贪心策略，按序逐步抽取规则，与数据集中的剩余样本一一匹配，最后构建一个有序规则集合作为最终的分类器。虽然M1算法形式相对简单，易于编程实现，但是当面对大型数据样本时，其效率并不高，因此又提出了M2算法。M2算法通过更复杂的规则选择和管理机制来优化分类过程，特别适合处理大规模和复杂的数据集。

### 2.1.1 规则生成

CBA算法的规则生成部分就是去寻找所有满足最小支持度和最小置信度约束的规则项集，一个规则项集就是一个由条件集（x）及类标签（y）形成的组合（x->y）。条件集是一组特定的属性，而类标签则是可能的输出分类之一。

类关联规则的支持度，反映了同时满足条件集和特定类标签的数据实例在整个数据集中的占比。而置信度，度量了在满足条件集的数据实例中，同时也符合规则定义的类标签的占比。对于一个类关联规则XY，支持度计算公式为：

 (2.1)

式中，count(X∪Y)——数据集中同时包含X和Y的数据项数目

N­­ ——数据项总数目

这条类关联规则的置信度计算公式为：

 (2.2)

式中，count(X∪Y)——数据集中同时包含X和Y的数据项数目

count(X)­­ ——数据集中包含X的数据项数目

如果一个规则项集满足最小支持度约束minSup，则其是一个频繁规则项集，与之相对应的则是非频繁的规则项集。如果一个规则的置信度满足最小置信度约束，那么它就是一个准确的规则。类关联规则应当都是频繁且准确的规则。

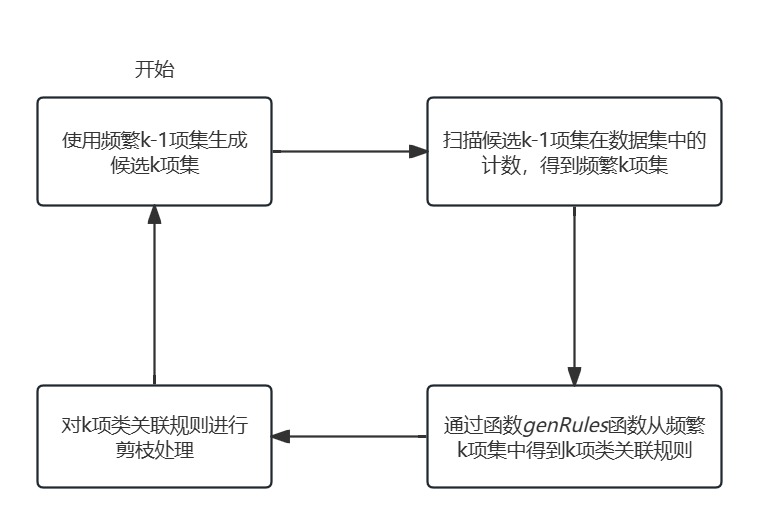


图2.1 生成类关联的关键步骤

### 2.1.2 分类器构建的CBA-M1算法

CBA-M1算法的主要核心是基于贪心策略，尽可能早的选出高优先级的规则。当存在多种规则时，需要确定某个样本优先使用其中的哪一个规则,故需要对生成的类关联规则进行优先级排序，排序规则如下：

给定两个规则ri和rj，规则ri优先级高于规则rj的条件是：

（1）ri的置信度大于rj的置信度。

（2）ri和rj置信度相同，但是ri的支持度大于rj的支持度。

（3）ri和rj的支持度和置信度均相同，但是ri比rj产生的时间要早。

CBA-M1算法的基本思想就是根据生成的类关联规则，构建出一组高优先级有序规则序列< r1, r2, ··· , rn, default\_class >作为最后的分类器。序列中的规则按照优先级降序排列，其中default\_class是默认类，当前面的规则都不能覆盖这条记录时，则将其划分入默认类中。

M1算法首先根据规则序列依次选择规则，并逐一与样本进行匹配。如果样本条件满足当前规则的条件，则说明匹配成功，此时将移除所有符合该规则条件的样本。然后，使用下一条规则继续匹配剩余的数据集。每轮中未匹配成功的样本将被分类为默认类，即剩余数据集中出现次数最多的类标签。在分类器构建过程中，会测试类关联规则集合中的每一条规则，并记录添加每条规则后的累计分类错误数。匹配结束后，将筛选出使得分类错误总数最低的规则，丢弃所有优先级低于此规则的条目，只保留该规则以及优先级高于此规则的规则集合。这些保留的规则构成关联规则分类器，将用于对测试集进行最终的分类。

### 2.1.3 分类器构建的CBA-M2算法

M2算法的中心思想和M1算法基本一致，仍然是先按照上述规则对类关联规则进行优先级排序，找出临界点并丢弃临界点之后的规则，不同的是算法的匹配方法。整个过程包括三个阶段，首先通过缩小候选规则的方法，减少不必要的规则判定，提高训练效率。CBA-M2算法在阶段一中首先对数据集中的样本进行遍历，找出正确分类该样本的最高优先级规则crule，以及错误分类该样本的最高优先级规则wrule。若crule的优先级高于wrule，则该样本被正确分类，并将这条规则添加至指定集合U中。若wrule的优先级高于crule，则需要决定该样本由crule覆盖还是由wrule覆盖，记录后在阶段二处理。阶段二中，遍历集合U，如果某个样本的wrule是其他样本的crule，那么就由wrule覆盖这个样本。否则，从集合U中返回所有错误分类该样本且优先级大于该样本crule的规则。阶段三中，对前两个阶段得到的规则集进行筛选，丢弃那些不会提高分类器准确率的规则。和M1算法类似，以分类错误数最少的规则为临界点，在此规则之后的所有规则都将被丢弃，因为它们只会产生更多的错误。

## 2.2 APR算法

APR（Active Pruning Rules）作为2019年提出的算法[3]，在CBA-M1算法的基础上引入了动态剪枝机制，能够在分类器构建过程中实时调整规则的优先级和有效性。CBA-M1算法按照既定的规则序列依次匹配样本，完成匹配后会将所有满足条件的样本移除，然而，丢弃的数据实际上会影响其他优先级较低的规则，从而影响分类器性能。而APR算法则在每次规则匹配后，将主动更新剩余规则在未被匹配数据集中的支持度和置信度，对不满足最小支持度和最小置信度的规则进行剔除，减少冗余，确保仅保留最有效的规则，从而提高了分类器的效率和准确性。此外，APR算法通过减少规则之间的重叠和优化规则集合的大小，显著提高了分类过程的效率和准确度，特别适用于处理大规模和复杂的数据集。

## 2.3 CMAR算法

CMAR（Classification based on Multiple Association Rules，基于多重关联规则的分类）算法[2]在关联分类规则算法的基础上进行改进，旨在克服传统关联规则分类中的过度依赖单个高置信度规则的问题。CMAR算法与传统关联规则分类方法不同，它不依赖单个高置信度的规则，而是从关联规则中挑选出一组高置信度、相关的规则进行分类预测。同时，使用卡方分析方法进行加权，确保选择的多重关联规则集能够为新样本提供更全面的分类决策，减少预测中偏差从而提高分类的准确性。

为了高效地挖掘满足支持度和置信度要求的规则，CMAR使用FP-growth算法的变种结构，通过构建关联FP-tree，能够快速地处理大型数据库中的频繁项集挖掘任务。同时，CMAR采用一种新的数据结构CR-tree用于存储和检索已挖掘的规则。该结构通过前缀树的方式来共享规则中的重复项，大大降低了规则的检索时间。

对新样本进行分类预测时，CMAR首先从CR-tree中检索出与新样本特征相匹配的规则集合，这些规则按照目标类别进行分组，每个分组对应一个可能的分类标签。然后，CMAR根据每个规则的卡方统计值和加权分析方法计算该分组的综合权重，从而评估分组的整体效果。通过对比不同类别分组的组合效果分数，CMAR选择综合效果最强的组标签作为最终预测的分类标签，从而对新样本进行准确的分类。如果匹配的规则不一致或存在冲突，则通过综合所有规则的效果进行最终决策，以确保分类预测的稳定性和准确性。这种基于多重规则的分类策略有效避免了单一规则引起的偏差与过拟合问题，提高了CMAR模型的预测性能。

## 2.4 本章小结

本章主要介绍了四种关联分类规则算法的基本原理，其中CBA、APR算法都是基于单一的类关联规则进行分类预测，而CMAR基于多个类关联规则进行判别。APR算法引入了动态剪枝机制，能够有效减少冗余规则，提升规则的质量。这些关联分类算法各有其特点和应用场景，提供了一种相较于传统分类方法更具解释性的方式。目前已经深刻了解了算法的基本原理，这一步十分关键，对于下一步软件设计具有重要意义。

# 第3章 关联规则分类软件设计

关联分类规则挖掘软件需要实现多种算法对指定数据的关联规则挖掘，软件应具有良好的交互性设计以及数据的可视化展示，帮助用户更好地对比和分析算法结果。

## 3.1 需求分析

关联分类规则挖掘软件，最主要的功能是使用关联分类规则算法构建分类器，并测试不同算法的性能。通过分析，该软件需要具备以下功能：支持用户自定义选择数据集，可以自由设定最小支持度和最小置信度参数；支持用户选择多种算法中的某一个算法进行分析，获取该算法在当前阈值条件下针对当前数据集生成的类关联规则、准确率、运行时间等结果，使用户能够充分了解该算法在当前数据集下的运行效率。

为了让用户能够自定义选择数据集，需要实现本地文件上传功能，让用户能够上传指定格式的数据文件，并且软件需要识别文件中的数据并在展示区域进行展示，方便用户确认上传的数据是否存在错误。

由于用户在使用软件时，需要对不同算法以及自身需求设置不同的阈值，所以软件需要提供参数设置部分，允许用户灵活地调整阈值大小。同时，软件页面中应提供一个算法选择区域，实现对不同算法的调用。为了展示算法的运行结果、比较不同算法的准确率和效率，软件应在结果展示页面中展示分类器中的类关联规则并提供多种算法评价指标，帮助用户更好地了解算法的分类效果和性能。

软件中包含多种算法实现，为了比较不同算法的运行效率和结果，软件中应当对不同算法结果使用柱形图或者图表进行展示，方便用户清晰地看到不同算法的特点及差异。

## 3.2总体设计

为了提高软件的可维护性和扩展性，计划采用前后端分离的软件架构。前端部分负责用户界面展示和交互，后端负责处理数据和算法运算，两者之间通过网络请求实现高效交互，这种模式不仅能提高开发的灵活性，还便于后续的系统维护和升级。

软件的前端部分，主要包含两部分功能，一是操作区，二是展示区。操作区需要实现文件的上传、参数选项设置、不同算法的选择等功能。展示区需要实现文件数据的展示、算法运行结果以及不同算法对比分析结果的展示。操作区与展示区两部分之间需要实现数据通信，使操作结果能够在快速在展示区进行展示。这部分页面设计应当简洁明了，易于操作，提供一个用户友好的操作界面。

软件的后端部分负责数据预处理、不同算法的代码实现以及搭建服务器。数据预处理模块负责接收前端上传的文件数据，并对数据进行预处理操作。算法实现模块是算法的主体部分，负责实现多种算法，返回的结果包含分类器中的类关联规则、分类准确率、算法运行时间等指标。搭建的服务器负责接收前端发出网络请求并返回响应结果，实现两者的交互。服务器部分根据前端发送的不同网络请求，执行相应的操作并返回结果。

软件整体的层次图如图3.1所示：

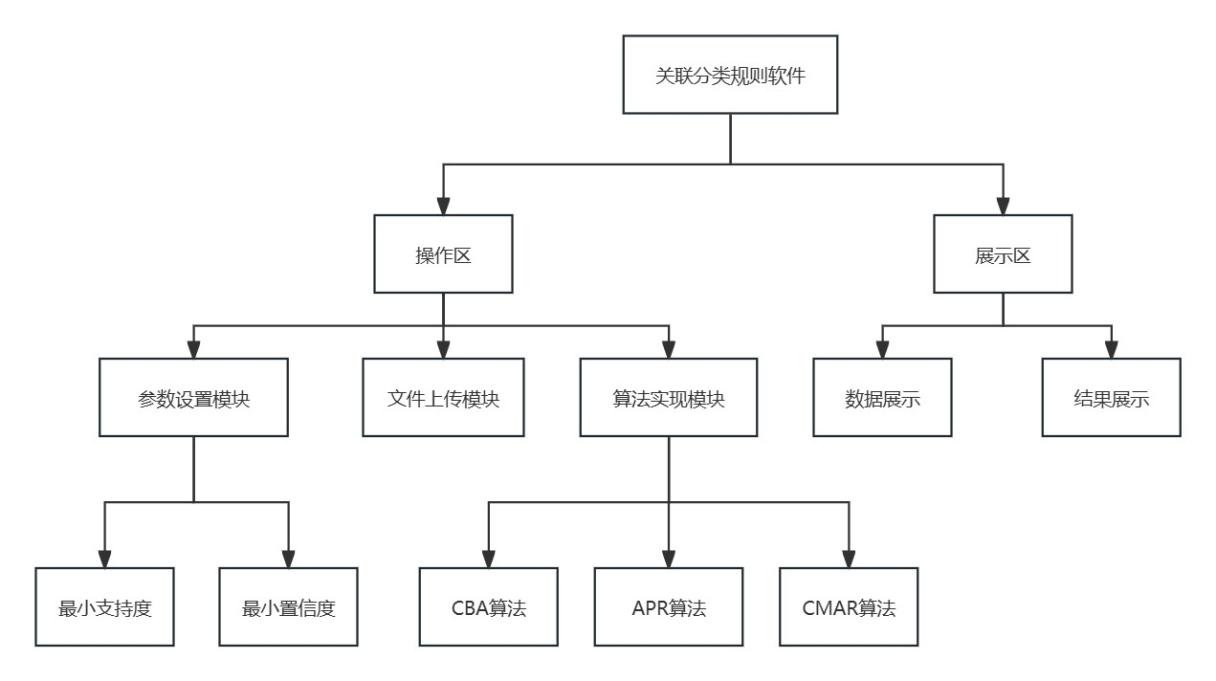


图3.1 关联分类规则软件层次图

## 3.3 详细设计

### 3.3.1功能设计

软件主要包含文件上传与展示、参数设置、算法实现、网络请求四个模块。其中网络请求模块为其他模块提供信息传递服务。

1）文件上传与展示模块

文件上传部分需要满足用户能够自定义上传本地CSV文件，该部分负责接收用户上传的数据，并借助POST网络请求将数据传送给服务端。同时，将用户上传的数据在客户端页面进行展示，让用户可以在上传后立即查看数据的内容，确保上传的数据符合用户的预期。

2）参数设置模块

参数设置部分需要满足用户使用鼠标或键盘自由编辑最小支持度和最小置信度，并且阈值大小应精确到小数点后两位数，以满足用户对于这些参数的精细化调整需求。同时，功能实现中需要提供用户友好的界面，方便用户响应式地输入和调整最小支持度和最小置信度的数值，在用户进行修改后，页面需要即时反馈参数设置的变化。

3）算法实现模块

关联分类规则算法要求数据类型必须为离散型，因此需要对连续型数据进行离散化处理，同时为了减小分类器构建过程中的复杂度，需要对离散化数据值进行编码处理。而从实际中采集到的数据往往存在着格式不统一、属性值缺失、噪声点等一系列问题。尽管测试时所使用的数据是从UCI机器学习资源库中下载的，这些数据已经得到了相关人员的一些预处理，但是仍然存在数据缺失以及需要对连续型数据进行离散化处理两个问题。下面就这两个方面进行介绍。

一般来说，对于缺失值，需要分两种情况区考虑。对于缺失值较多的属性，应当毫无保留的删去，因为这样属性的存在将会对最后分类器的性能带来极大影响。对于缺失值较少的属性，则应当考虑保留。算法的预处理过程中，设定的缺失率阈值为0.5，当缺失率小于阈值时，利用缺失值所属特征的众数进行填充；对于缺失率高于阈值时，则丢弃这一列。

预处理过程中有一个问题是对连续型属性值离散化处理。所谓连续型属性值，就是该属性的取值范围并非有限集，与之相对的则是离散型特征，其取值只可能落在一个有限集合中。连续型属性值非常常见，但是在关联分类规则算法中，只能处理离散型的特征。因此需要对连续型属性进行离散化处理，将指定区间内的数值映射为同一类型。实现过程中，采用了基于信息增益的离散化方法，该算法根据数据的混乱程度找出最佳的离散化临界点。

关联分类算法的实现原理在第二章节已经进行过介绍，其中CBA算法包含CBA-RG（规则生成）和CBA-CB（分类器构建）两部分，为了更加准确地描述算法的设计过程，本处给出相关算法伪代码。CBA-M1算法伪代码如下所示：

|  |
| --- |
| 算法1 CBA-M1算法 |
| 1. **输入**：数据集D，规则集R 2. **输出**：分类器C 3. *R* = sort(*R*) 4. **for** each rule *r* *R* in sequence **do** 5. temp = 6. **for** each case *d* *D* **do** 7. **if** d satisfies the conditions of *r* **then** 8. store *d*.id in *temp* and mark *r* if it correctly classifies *d* 9. **end if** 10. **end** **for** 11. **if** *r* is marked 12. insert *r* a **then** t the end of *C* 13. delete all the cases with the ids the in *temp* from *D* 14. selecting a default class for the current *C* 15. compute the total number of errors of *C* 16. **end** **if** 17. **end** **for** |
| 1. 找出错误总数最少的第一条规则p，并且删除C中p之后的所有规则 2. 将p关联的默认类添加到C的末尾 3. **return** C |

CBA-M2算法首先对数据集样本遍历，找出正确分类该样本的最高优先级规则和错误分类该样本的最高优先级规则，并通过三个阶段对规则进行筛选，构建出最终的分类器。CBA-M2算法的伪代码如下所示：

|  |
| --- |
| 算法2 CBA-M2算法 |
| 1. **输入：**数据集D 2. **输出：**分类器C 3. *Q* = ; *U* = ; *A* =; 4. **for** each case *d* *D* **do** 5. *cRule* = maxCoverRule(*Cc*, *d*) 6. *wRule* = maxCoverRule(*Cw*, *d*) 7. U = U {cRule} 8. cRule.classCasesCovered[*d*.class]++ 9. **if** *cRule* > *wRule* **then** 10. Q = Q {cRule} 11. mark *cRule* 12. **else** *A* = *A* <*d*.id, *d*.class, *cRule*, *wRule*> 13. **end** **if** 14. **end** **for** 15. **for** each entry < *dID*, *y*, *cRule*,*wRule* > **∈** A **do** 16. **if** *wRule* is marked **then** 17. *cRule*.classCasesCovered[*y*]**--** 18. *wRule*.classCasesCovered[*y*]++ 19. **else** 20. *wSet* = allCoverRules(*U*, *dID*.case, *cRule*) 21. **for** each rule *w wSet* **do** 22. *w*.replace = *w*.replace {<*cRule*, *dID*, *y*>} 23. *w*.classCasesCovered[*y*]++ 24. **end** **for** 25. Q = Q wSet 26. **end** **if** 27. **end** **for** 28. *classDistr* = compClassDistri(*D*) 29. ruleErrors = 0 30. *Q* = sort(*Q*) 31. **for** each rule *r* in *Q* in sequence **do** 32. **if** *r*.classCasesCovered[r.class] 0 **then** 33. **for** each entry <*rule*, *dID*, *y*> in r.replace **do** 34. **if** the *dID* case has been covered by a previous *r* **then** 35. *r*.classCasesCoverred[*y*]— 36. **else** 37. *rule*.classCasesCovered[*y*]— 38. **end** **if** 39. **end** **for** 40. ruleErrors = ruleErrors + errorsOfRule(r) 41. classDistr = update(r, classDistr) 42. defaultClass = selectDefault(classDistr) 43. totalErrors = ruleErrors + defaultErrors 44. Insert <*r*, *default-class*,*totalErrors*> at the end of *C* 45. **end** **if** 46. **end** **for** 47. 找出错误总数最少的第一条规则p，并且删除C中p之后的所有规则 48. 将p关联的默认类添加到C的末尾 49. **return** C |

APR算法在CBA-M1算法的基础上，在每轮匹配过程中都对规则集中余下的规则进行剪枝处理，提高了分类效率和模型的精度。APR算法伪代码如下所示：

|  |
| --- |
| 算法3 APR算法 |
| 1. **输入：**数据集D，规则集R 2. **输出：**分类器C 3. **for** each rule R in sorted rules **do** 4. **for** each dataset example e in **D** 5. **if** e is covered by R **then** 6. Add R to classifier C 7. Remove e from D 8. continue 9. **end** **if** 10. **end** **for** 11. **for** each remaining rule R' in sorted rules **do** 12. update support and confidence of R' based on remaining data in D 13. Re-rank rules 14. **end** **for** 15. **end** **for** 16. 找出错误总数最少的第一条规则p，并且删除C中p之后的所有规则 17. 将p关联的默认类添加到C的末尾 18. **return** C |

4）网络请求模块

由于使用的是前后端分离架构，所以彼此之间的网络通信尤为重要。后端部分借助Flask框架搭建服务端，一是算法主体使用Python代码编写较为方便，二是考虑到Flask框架的轻量级和易于扩展的特性，非常适合用于开发数据密集型的Web应用。前端部分使用Axios第三方库封装网络请求，通过发送POST类型的网络请求，将前端本地的参数或者数据发送到后端的指定路由，后端对网络请求响应后，前端接收网络响应返回的结果，并将结果渲染到页面中。

### 3.3.2 页面设计

页面设计的主要目标是提供一个简洁易用的界面，方便用户上传文件并通过不同的算法进行处理。页面设计遵循简洁明了、响应式设计和一致性的原则，确保用户能够轻松理解并使用各项功能。页面布局包括左侧的操作区和右侧的展示区。操作区包括文件上传按钮、多个算法选择按钮、参数设置滑动条以及参数显示区域，而展示区用于显示处理结果或提示信息。通过线框图和原型设计，确定了各个组件的位置和功能，确保用户可以方便地上传文件、选择算法并设置参数。用户界面设计强调易用性和一致性，按钮和滑动条使用统一的颜色和风格，用户可以直观地看到当前设置的“最小支持度”和“最小置信度”。用户体验设计方面，通过实时反馈机制，当用户上传文件或调整参数时，展示区会立即更新，提升用户交互的顺畅度和满意度。

发出网络请求但响应结果未返回时，显示如下加载页面，提醒用户已正确发送数据，后端部分仍在处理中。

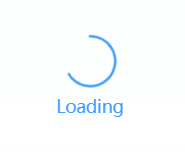


图 3.2 加载页面

当用户未连接网络，或者网络发生故障导致网络请求无法发送时，将出现如下提示页面，出现问题及时为用户做出反馈。



图3.3 提示页面

通过这些简洁易懂的提示语，可以显著提升用户体验，帮助用户更好地理解和操作软件，从而减少困惑和操作失误，使用户与软件之间的交互更加顺畅和高效。

## 3.4本章小结

本章主要描述了关联分类规则软件的设计过程，该软件在设计上强调用户友好性，同时注重交互效果。在界面布局设计上，采用左侧操作栏和右侧展示栏的简约风格，方便用户能够快速理解和操作，提升使用效率。此外，本章还对算法实现模块进行了详细设计，并提供伪代码进行描述。在后续阶段，将根据软件设计进行开发，确保最终产品满足需求。

# 第4章 软件实现与测试

关联分类规则挖掘软件采用前后端分离架构，前端基于Vue3框架开发，负责实现数据的上传与展示，用户操作以及算法结果的可视化功能，后端采用Flask框架搭建，负责实现算法的主体功能和网络通信。

## 4.1 软件实现

软件的页面主要基于Vue 3框架进行开发，分为操作栏和数据展示两大部分，这两部分密切配合以实现软件的所有功能。利用Vue3的响应式编程和组件化的特性，可以实现功能的模块化开发，同时使用Pinia实现状态管理，实现不同组件之间的数据通信。该框架的组合式API使状态管理更加直观和灵活，极大适应了复杂交互页面的需求。前端部分实现了数据上传按钮、参数设置选项、算法选择按钮、多算法对比以及结果展示的在内的多个界面组件。在页面开发中，结合了Element-Plus组件库和ECharts可视化库来实现数据结果的动态可视化展示，确保用户能够轻松上传数据，选择合适的参数和算法，并且能够清晰的看到算法运行结果。

前端页面的操作栏部分主要负责用户操作和参数设置，包含多个交互组件。用户可以通过上传文件按钮选择本地文件进行上传。点击该按钮后，会弹出一个文件选择面板，用户可以在其中选择要上传的文件。确认文件后，文件将上传到服务器。同时，页面右侧的数据展示部分将显示上传文件的预览内容。左侧操作栏包含两个滑动栏，可以使用鼠标最小支持度和最下置信度参数。此外，操作栏还包括选择不同算法的按钮。点击按钮后，将会触发监听事件，向服务器发送POST网络请求以及相关参数，服务器会对已上传的数据进行一系列处理，包括数据清洗、数据离散化和数字编码等一系列数据预处理操作。预处理后的数据将被传入封装好的算法模块，算法处理后的结果将通过网络响应返回给客户端，客户端接收后将结果渲染到数据展示区域，方便用户查看算法运行结果。如同上章介绍的页面设计，当响应结果未返回时，将展示加载页面提醒用户。

操作栏提供的两个滑动栏，用于调节最小支持度和最小置信度。这些滑动栏允许用户根据需要设置算法参数，从而影响算法的执行结果。调整滑动栏时，可以使用鼠标修改，当鼠标无法满足精细化需求时，也可以使用键盘进行微调。

数据展示部分则负责显示数据的详细信息和算法处理结果。初次上传文件后，原始数据将显示在此区域。当用户选择不同的算法并设置相关参数后，算法的执行结果也会在数据展示部分展示。其中右侧的上半部分为算法执行过程中预处理后的数据，下半部分为算法的执行结果，包含分类准确率、算法运行时间、分类器中的默认类、分类器中的类关联规则以及规则数目。

通过操作栏与数据展示部分的结合，用户能够方便地上传文件、调整算法参数以及执行不同的算法。同时软件中添加有“按钮”，可以集中直观地查看到多种算法的处理结果，对比不同算法的性能，如图4.1所示。

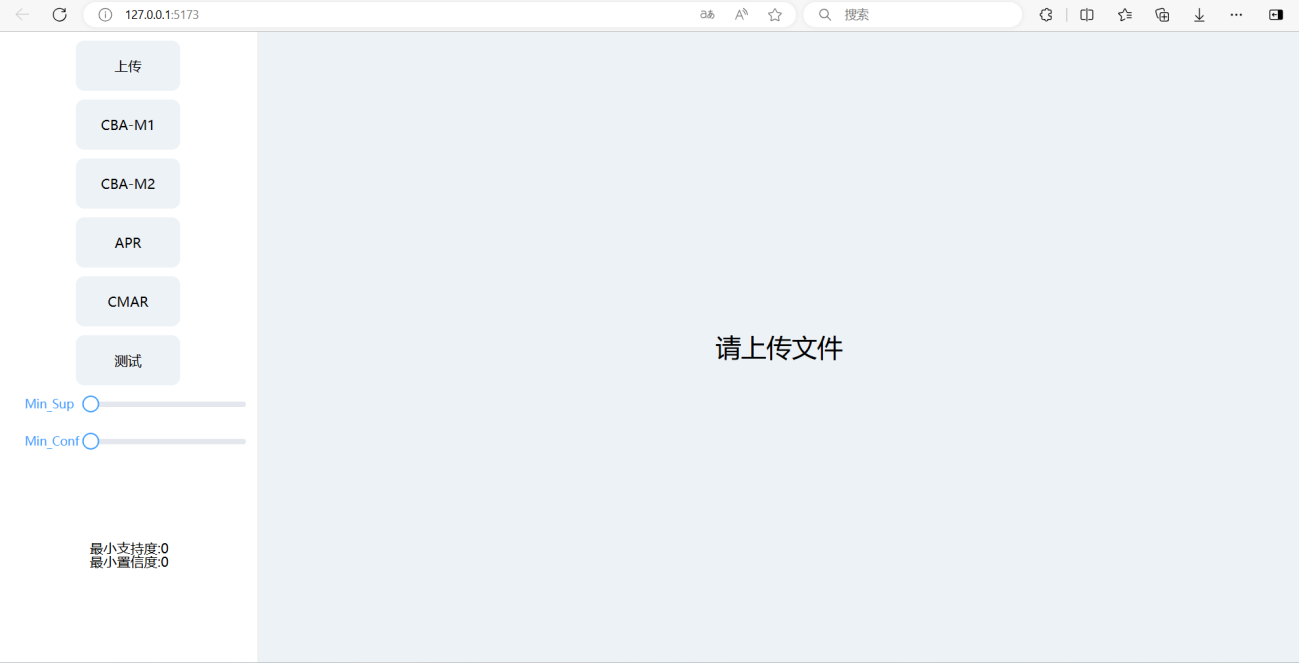


图4.1 软件初始界面

用户点击“上传”按钮后，可以选择本地csv文件进行上传，文件数据将在操作栏右侧展示，此处以UCI机器学习数据库中的鸢尾花（iris）数据集为例，如图4.2所示。



图4.2 数据集展示

使用鼠标操作最小支持度和最小置信度滑动栏，将参数值分别设置为0.01和0.5，点击“CBA-M1”、“CBA-M2”、“APR”、“CMAR”中的一个按钮，右侧将展示对应的算法运行结果，包括分类准确率、算法运行时间、分类器中的默认类和分类器中类关联规则及其数目，如图4.3所示。

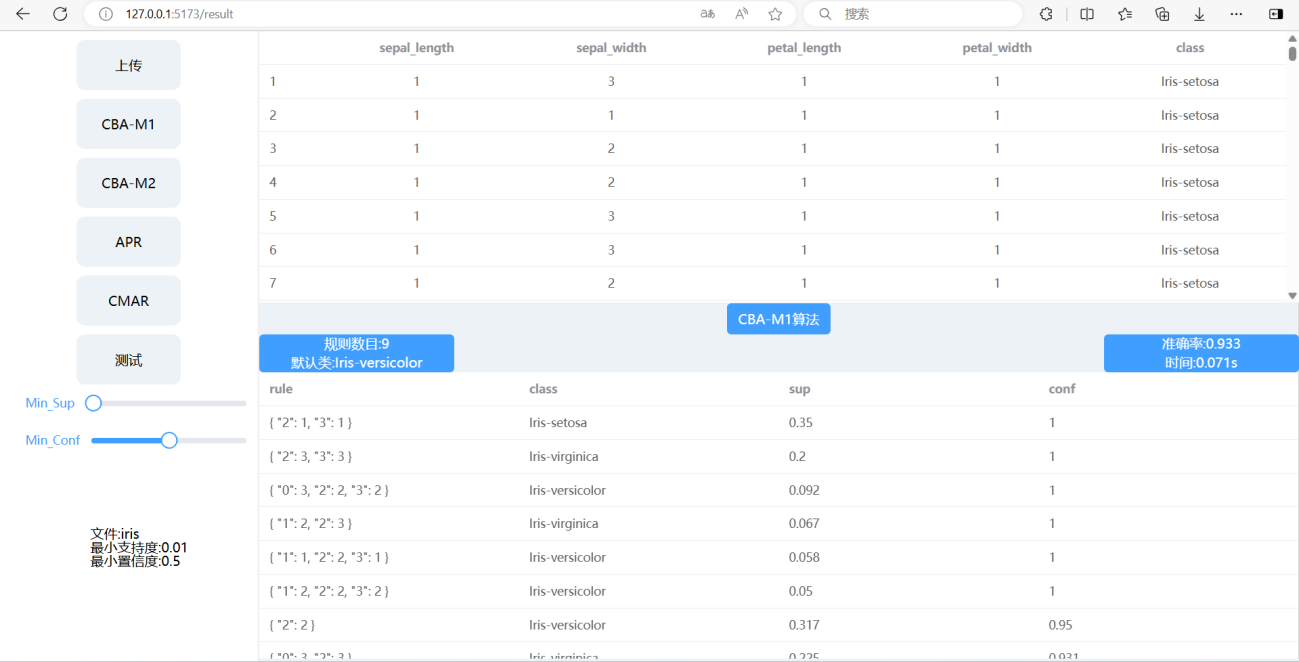


图4.3 CBA-M1算法运行结果

为了直观地比较不同算法，在页面中添加了测试选项，点击“测试”按钮后，后端算法模块将对数据集进行四种算法的十折交叉验证，最后取其平均值作为每种算法的运行结果。前端部分接收到运算结果后，页面中将展示四种算法及其运行结果的图表，包含准确率、规则数目、时间开销三种指标。结果展示包含柱状折线图、结果统计、折线图、柱状图四种方式，如图4.4、图4.5、图4.6、图4.7所示。

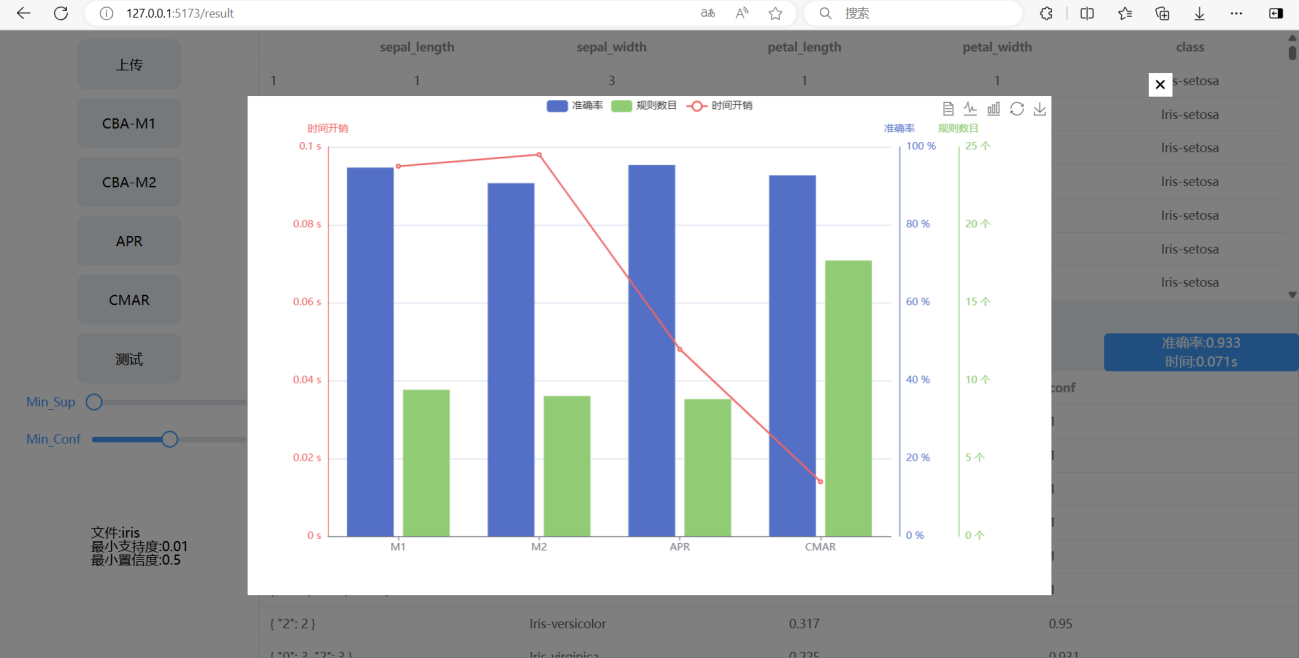


图4.4 四种算法测试结果的柱状折线图

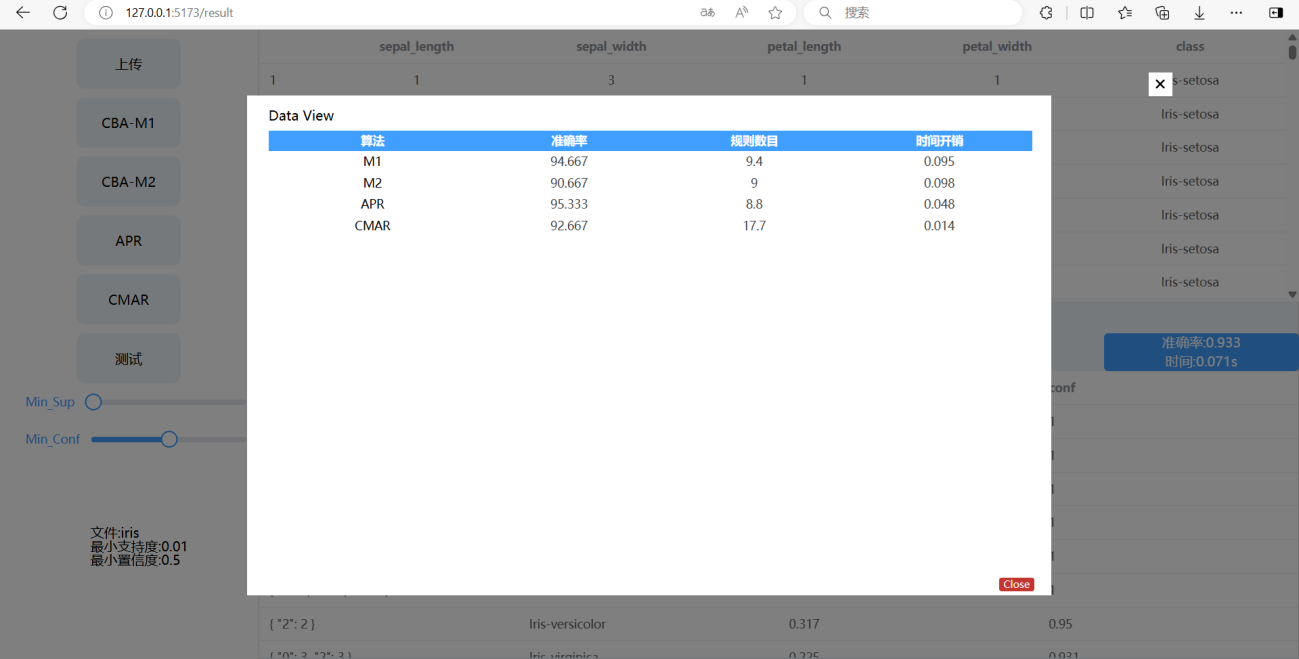


图4.5 四种算法测试结果的统计图

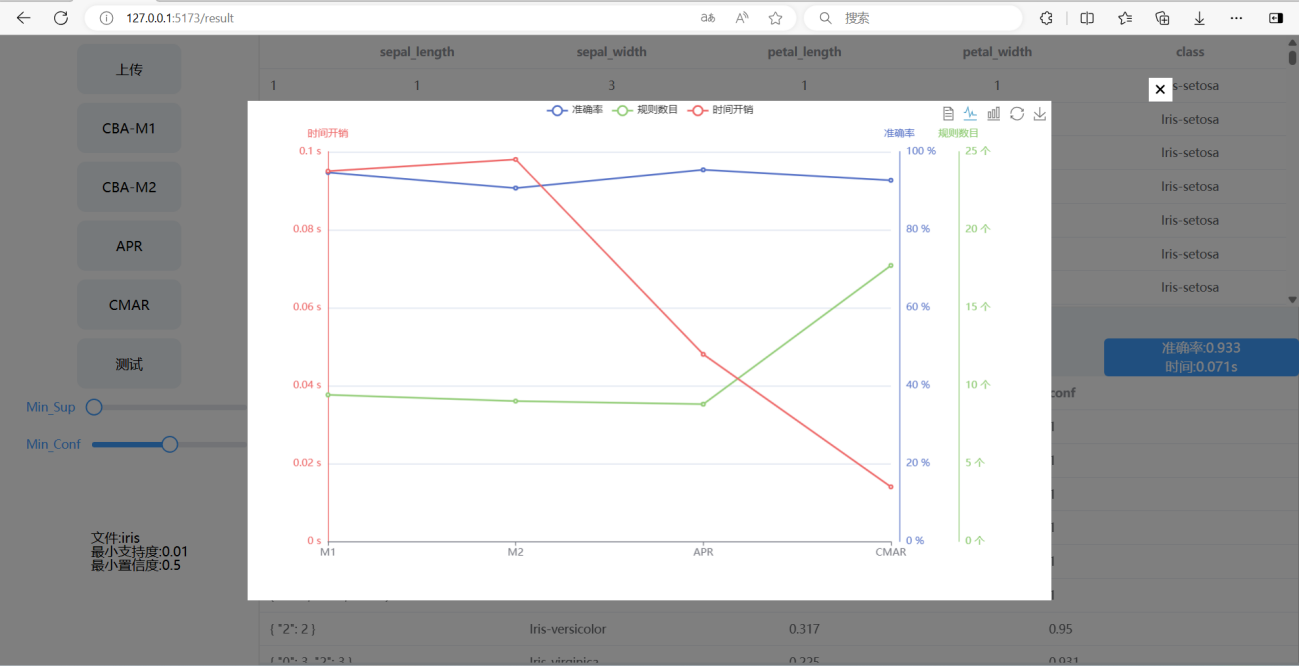


图4.6 四种算法测试结果的折线图

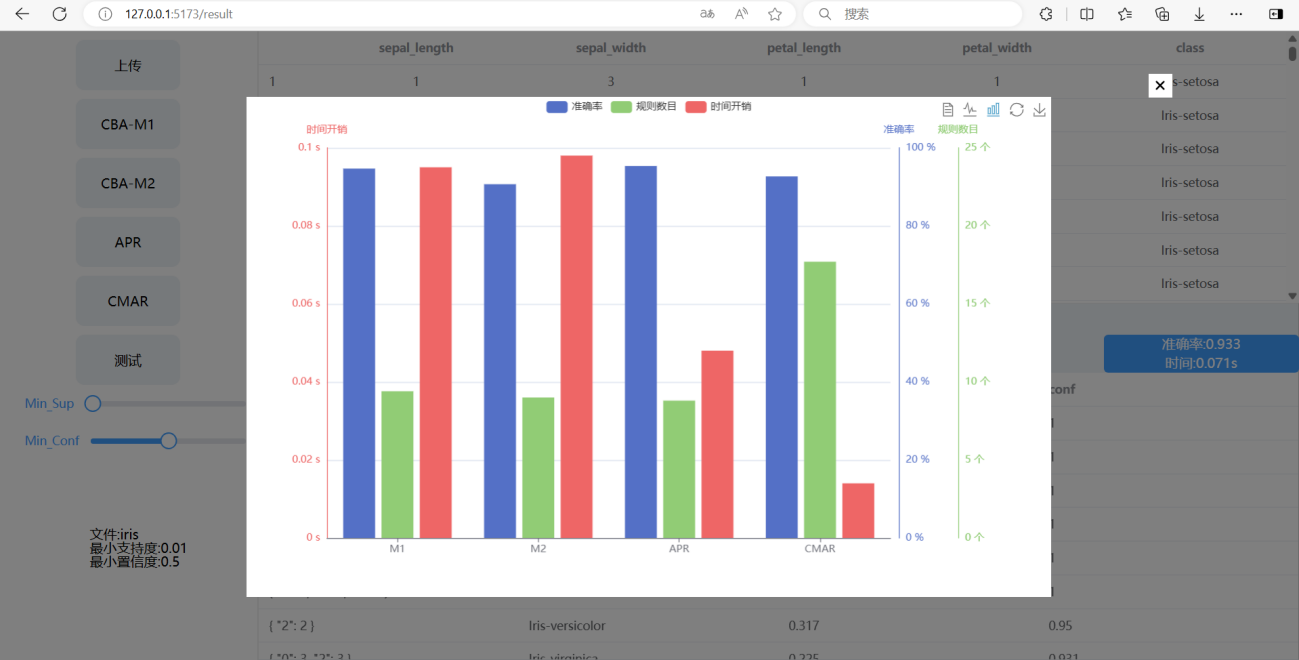


图4.7 四种算法测试结果的柱状图

## 4.2软件测试

软件测试的目的是找出软件中潜在的各种错误和缺陷，通过修正各种错误和缺缺陷提高软件质量。软件实现过程中，对软件进行了白盒测试和黑盒测试，白盒测试则侧重于内部逻辑的正确性，黑盒测试主要关注软件的功能性。在算法实现模块，对每种算法进行了白盒测试，确保所有的代码路径都能被测试到，尤其算法中逻辑较为复杂的部分。在软件页面中进行了黑盒测试，测试过程中设计了一系列测试案例，涵盖所有用户可能的输入和操作路径，以确保软件在所有预定的功能上都能按预期工作。经过测试后，软件执行过程中发现的错误均已改正。

软件初步构建时，测试了软件的边界条件处理能力，比如上传错误类型的文件，以确保软件能妥善处理异常输入；当软件被手动修改跳转到其他路由时，软件能够自动检测异常情况并进行处理。

每种算法实现过程中，编写了测试案例来检测算法运行结果，包括算法执行前的预处理模块，单独采用含缺失值的测试样例，确保每一过程的实现符合预期。在前后端交互过程中，测试了参数信息能否被正确接收等，避免数据传输过程中发生错误。对软件进行结构化测试之后，对对软件的各项功能进行了全面的测试。例如在实现界面中，上传了标准的iris数据集，并使用四种不同的算法模型进行了处理。测试中包括检查算法是否能正确识别和分类iris数据集中的不同种类，同时确保算法的结果展示符合预期。

经过多轮测试，主要发现并修正了以下几个问题。上传大型数据集时，软件响应较慢，造成卡顿。之后在软件中使用懒加载技术，仅在需要时加载数据，减少内存使用；实现CMAR算法时，由于使用卡方计算时的逻辑错误，导致分类结果异常。之后对代码进行审查，查阅文档，重新编写该部分代码；使用滑动栏需修改参数时，页面中显示的参数与实际参数不一致。发现是页面修改的参数没有进行响应式处理，页面参数的改变并没有引起具体值得改变，最终使用ref函数进行响应式处理。点击“测试”按钮后，前端可以接收到后端返回的结果，但是展示部分出现了数据混乱局面，之后通过查看ECharts官网中的教程文档，重新梳理并修改参数，问题得以解决。

通过上述多次测试，不断发现并解决了多种潜在问题，逐步提升了软件的质量和用户体验，最终软件的各项功能都能按照预期运行。

## 4.3 本章小结

本章主要介绍了软件实现和软件测试，通过前端实现的软件界面对软件功能和工作流程进行了详细介绍。基于前文中提到的需求分析和总体设计，实现的软件基本满足了预设功能，能够实现多种关联分类规则算法的选择以及数据可视化。软件中特意添加的测试功能，能够对四种算法分别进行十折交叉验证，利用柱状图和折线图可以清晰地看到不同算法的差异。代码实现过程中不断地进行测试，采用了黑盒测试和白盒测试两种方法，针对发现的不同问题，对相关代码进行了梳理和修改，确保软件能够正常运行，并提升软件的整体性能。

# 第5章 算法分析实验

本文实现的软件中算法部分使用python3.9.18编写，从UCI机器学习数据库中选取10个数据集对程序进行测试，设置的支持度阈值为0.01，置信度阈值为0.5。所有指标均采用10折交叉检验，取10次检验结果的平均值用来评估其总体性能指标。

## 5.1 关联分类算法的准确率比较

在关联分类任务中，准确率是衡量算法性能的一个关键指标。本文使用关联分类规则软件对UCI机器学习数据库中的十个数据集进行了测试。

表5.1 分类结果准确率

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 序号 | 数据集 | 准确率(单位：%) | | | |
| CBA-M1 | CBA-M2 | APR | CMAR |
| 1 | car | 75.584 | 71.757 | 80.344 | 76.43 |
| 2 | diagnosis | 80.833 | 77.5 | 80.833 | 95 |
| 3 | iris | 94.667 | 90.667 | 95.333 | 92.667 |
| 4 | monks | 100 | 94.238 | 100 | 50.419 |
| 5 | mammographic | 82.516 | 76.797 | 82.097 | 74.294 |
| 6 | qualitative\_bankruptcy | 100 | 97.6 | 100 | 98.8 |
| 7 | user-knowledge-modeling | 81.964 | 78.679 | 84.376 | 83.879 |
| 8 | vertebra | 68.71 | 82.581 | 84.194 | 75.484 |
| 9 | yeast | 37.004 | 41.463 | 41.769 | 46.438 |
| 10 | seeds | 88.571 | 88.571 | 90 | 92.381 |

如表5.1所示，APR算法在大多数数据集上表现出较高的准确率，特别是在car、iris、user-knowledge-modeling和vertebra数据集上。而CMAR算法在diagnosis、yeast和seeds数据集上表现最佳，CBA-M1和CBA-M2算法的表现则相对较弱。

APR算法在整体测试中都显现出较高的准确率，表明动态剪枝策略提升效果明显。通过采用动态剪枝，APR算法的分类器能够动态地筛选规则，实时修改类关联规则的优先级，确保留下的都是高质量规则，从而提高了分类器的准确率。CMAR算法基于卡方检验的加权机制，能够在分类过程中考虑规则的重要性和相关性，特别适合分类数据分布不均衡的数据集。这种机制使得CMAR算法在一些类别分布和特征相关性较强的数据集上表现出色。相比之下，CBA-M1算法和CBA-M2算法在规则生成和筛选过程中相对简单，对复杂模式的捕捉能力有限。这也导致它们在处理复杂数据模式时准确率较低。

从准确率方面来看，APR算法和CMAR算法通常表现较好，适用于对分类精度要求较高的场景。APR通过全面的模式捕捉和严格筛选，CMAR通过加权机制有效处理复杂类别分布。

## 5.2 关联分类算法的时间开销比较

表5.2 运行时间开销

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 序号 | 数据集 | 时间开销（单位：s） | | | |
| CBA-M1 | CBA-M2 | APR | CMAR |
| 1 | car | 7.201 | 10.199 | 2.866 | 0.839 |
| 2 | diagnosis | 0.094 | 0.19 | 0.083 | 0.147 |
| 3 | iris | 0.088 | 0.092 | 0.046 | 0.013 |
| 4 | monks | 1.986 | 2.208 | 1.725 | 0.446 |
| 5 | mammographic | 2.181 | 2.258 | 2.118 | 0.089 |
| 6 | qualitative\_bankruptcy | 4.504 | 4.825 | 1.119 | 0.577 |
| 7 | user-knowledge-modeling | 0.004 | 0.005 | 0.004 | 0.073 |
| 8 | vertebra | 4.947 | 4.859 | 3.672 | 0.157 |
| 9 | yeast | 0.057 | 0.058 | 0.056 | 0.776 |
| 10 | seeds | 8.869 | 8.587 | 1.289 | 1.432 |

如表5.2所示，CMAR算法在大多数数据集上表现出较快的计算速度，APR算法次之，CBA-M1和CBA-M2算法的时间开销则相对较大。

CMAR算法的时间效率与其使用的FP-growth算法和CR-tree结构密切相关。FP-growth算法通过构建频繁模式树（FP-tree），能够高效地发现频繁项集，这种结构显著减少了数据扫描次数，并通过压缩数据表示来提升计算效率。同时，CMAR算法利用CR-tree（Classification Rule-tree）来组织和存储生成的分类规则，这种树结构不仅有助于快速查找和匹配规则，还能有效地剪枝，避免不必要的计算。正是这种高效的数据结构和算法设计，使得CMAR在时间开销上表现优异。

相比之下，APR算法虽然在准确率上表现出色，但其规则生成过程中需要遍历大量的候选项集，计算频繁度和置信度。因此，在处理大规模数据时，APR算法的时间开销较高。不过，APR算法通过剪枝和优化策略，依然在多数情况下表现出较好的时间效率。CBA-M1和CBA-M2算法在时间开销上表现较差，主要原因在于其规则生成和筛选策略相对简单直接，缺乏高效的剪枝机制。这使得在处理大规模数据时，生成和筛选规则的时间显著增加。此外，CBA-M2由于其基本计算步骤较多，导致总体时间开销较高。

数据的规模和复杂性也对时间开销产生了显著影响。在较小且简单的数据集（如iris和user）上，各算法的时间差异不大；而在较大且复杂的数据集上，CMAR算法的时间优势更为明显。

从时间开销来看，CMAR算法在多数情况下速度更快，适合处理大规模数据且对速度有要求的任务。其高效的规则筛选机制显著减少了计算量。

## 5.3 关联分类算法的规则数目比较

表5.3 分类器关联规则数目

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 序号 | 数据集 | 规则数目(单位：个) | | | |
| CBA-M1 | CBA-M2 | APR | CMAR |
| 1 | car | 24.6 | 8.2 | 12.9 | 29.7 |
| 2 | diagnosis | 4 | 2.9 | 4 | 62.3 |
| 3 | iris | 9.3 | 9.2 | 9 | 15.6 |
| 4 | monks | 22.3 | 21.2 | 25.8 | 21.1 |
| 5 | mammographic | 63.1 | 15.9 | 31.7 | 12.5 |
| 6 | qualitative\_bankruptcy | 10 | 8.2 | 5.8 | 72.4 |
| 7 | user-knowledge-modeling | 4.9 | 3.2 | 5.9 | 24.6 |
| 8 | vertebra | 15.3 | 9.6 | 11.8 | 74.4 |
| 9 | yeast | 3 | 2 | 3 | 51.1 |
| 10 | seeds | 23 | 18.4 | 16.1 | 140.9 |

如表5.2所示，CBA-M2算法整体上生成的规则数目最少，APR次之，CBA-M1稍多，CMAR生成的规则数目最多，分类器中的类关联规则数目直接影响了模型的复杂性。

CBA-M2算法在规则生成过程中采用了更严格的筛选标准，虽然可能牺牲一部分准确率，但确保了生成规则的简洁性。严格的筛选机制减少了冗余规则，有助于提升模型的可解释性和应用效率。CBA-M1算法在规则生成上相对宽松，导致生成的规则数目较多。宽松的规则生成策略虽然提高了覆盖范围，但也增加了冗余规则，影响了模型的简洁性。APR算法在CBA-M1算法的基础上引入的动态剪枝策略，使得APR在规则数量少于CBA-M1。CMAR算法由于采用了基于卡方检验的加权机制，生成了大量规则。这种机制在提高分类准确率的同时，也增加了规则数目，导致模型复杂度较高。虽然这种方法能够在复杂数据集中取得较高的准确率，但规则数目的增加也带来了模型的解释难度和应用负担。

从规则数目来看，CBA-M2算法生成的规则数目最少，适用于对模型简洁性要求较高的应用场景。

## 5.4 本章小结

本章使用关联分类规则软件对四种算法的性能进行了分析，包括准确率、时间开销、规则数目三个维度。通过实际数据集的测试，发现测试的几种算法各有优劣。通过合理选择和组合不同算法，可以在具体应用中获得最佳效果。针对实际应用需求，应综合考虑准确率、时间开销和规则简洁性等因素，选择最适合的算法。经过本章的测试，对于不同的算法的表现也有了更加客观的认识。

# 第6章 总结与展望

## 6.1 结论

本文围绕关联分类规则挖掘软件的设计与实现，进行了系统性研究和分析，取得了一些显著的成果。选择并实现了CBA-M1、CBA-M2、APR和CMAR四种经典的关联分类算法。这些算法在关联规则挖掘和分类任务中具有代表性，能够有效地处理不同类型的数据集。特别是在APR算法的实现过程中引入动态剪枝策略，提高了规则筛选的效率和分类准确性。这种优化不仅减少了冗余规则，还显著提升了算法的运行速度和准确性。

软件采用前后端分离的架构进行开发，前端使用Vue3框架，后端使用Flask框架。这种架构设计提高了软件的扩展性和维护性，前端实现了数据的上传与展示、用户操作和算法结果的可视化功能，后端则负责数据处理和算法实现。通过这种设计和模块化开发，软件能够灵活应对用户的不同需求，提供高效的数据处理和结果展示。

软件在界面展示中注重数据可视化和用户体验的优化。通过使用ECharts可视化库，软件能够直观地展示不同算法的运行结果，包括分类准确率、运行时间和规则数目等指标。用户界面设计简洁明了，操作便捷，提升了用户体验。同时，软件提供了参数调节功能，允许用户根据具体需求调整最小支持度和最小置信度参数，从而灵活控制算法的执行。

通过对多个UCI公开数据集的实验测试，本文详细分析了四种算法在准确率、时间开销和规则数目等方面的性能表现。实验结果显示，APR算法在分类准确率高的同时，也拥有较少的时间开销和规则数目，综合性能优异;CMAR算法在时间效率上具有明显优势;而CBA-M2算法在生成规则数目上占优，构造的分类器最为简洁。通过这些分析，为实际应用中的算法选择提供了客观依据和指导。

## 6.2 后续研究工作展望

尽管本论文在关联分类规则挖掘软件实现取得了一定的进展，但仍有许多值得进一步探索和改进的方面。未来的研究工作可以从以下几个方面展开：

1. 算法优化与扩展：

进一步优化现有算法的计算效率和分类效果，特别是针对大规模数据集，探索更高效的数据结构和优化策略，如并行计算和分布式处理，以提升算法的可扩展性和实时性。此外，可以尝试将多种关联分类算法进行组合，形成集成模型，利用不同算法的优势互补，提高分类器的准确性和稳定性。

1. 算法创新：

在原有的关联分类规则算法的基础上进行创新，通过改进现有算法的分类器构造方法，挖掘出高质量的规则，从而提高分类准确率并减少规则的数目。进一步研究结合人工智能和关联规则的结合算法，利用深度学习的特征提取能力和关联规则的可解释性，构建更为高效和准确的分类器。

1. 用户界面和交互设计优化：

改进软件的用户界面和交互设计，提升用户体验。特别是增加数据可视化功能，使用户能够更加直观地理解和分析分类结果。增加更多的参数调节选项，满足用户的个性化需求，使软件在功能和操作上更加灵活和友好。

# 参考文献

1. Liu B, Hsu W, Ma Y. Integrating classification and association rule mining[C]//Proceedings of the fourth international conference on knowledge discovery and data mining. 1998: 80-86.
2. Li W, Han J, Pei J. CMAR: Accurate and efficient classification based on multiple class-association rules[C]//Proceedings 2001 IEEE international conference on data mining. IEEE, 2001: 369-376.
3. Rajab K D. New associative classification method based on rule pruning for classification of datasets[J]. IEEE Access, 2019, 7: 157783-157795.
4. 黎文娟, 周忠眉. 基于大数据与人工智能的关联分类改进算法[J]. 广东通信技术, 2022, 42(01): 63-66.
5. 秦晨普, 张云华. 基于分类修剪的关联分类算法改进[J]. 计算机系统应用, 2019, 28(4): 194-198.
6. 李家辉, 周忠眉. 基于多次学习和关联度的关联分类改进算法[J]. 南京大学学报 (自然科学版), 2019, 55(4): 564-572.
7. 郭明环, 代素香, 吕亚娟. 关联挖掘在图书馆读者分类中的应用探索[J]. 内蒙古科技与经济, 2021, (17): 120-123.
8. 翟悦, 李楠, 于文武. 基于扩展概念格的带约束关联分类规则挖掘方法[J]. 大连交通大学学报, 2021, 42(4): 88-93.
9. Hasanpour H, Meibodi R G, Navi K. Improving rule-based classification using Harmony Search[J]. PeerJ Computer Science, 2019, 5: e188.
10. Thanajiranthorn C, Songram P. Efficient rule generation for associative classification[J]. Algorithms, 2020, 13(11): 299.
11. Dhanalakshmi R, Anitha K, Rukmani Devi D, et al. RETRACTED ARTICLE: Association rule generation and classification with fuzzy influence rule based on information mass value[J]. Journal of ambient intelligence and humanized computing, 2021, 12(6): 6613-6620.
12. Cagliero L, Canale L, Farinetti L, et al. Predicting student academic performance by means of associative classification[J]. Applied Sciences, 2021, 11(4): 1420.
13. Alazaidah R, Samara G, Almatarneh S, et al. Multi-Label Classification Based on Associations[J]. Applied Sciences, 2023, 13(8): 5081.
14. 郭叔瑾, 吴辰文. 基于统计调和平均值的特征加权关联分类算法[J]. 西北大学学报自然科学版, 2018, 48(6): 817-826.
15. 黄再祥, 周忠眉, 何田中, 等. 改进的多类不平衡数据关联分类算法[J]. 模式识别与人工智能, 2015, 28(10): 922-929.
16. 苗世强, 郑晓势. 关联分类算法的研究与实现[J]. 智能计算机与应用, 2018, 8(2): 138-139.
17. 李学明, 杨阳, 秦东霞, 等. 基于频繁闭项集的新关联分类算法 ACCF[J]. 电子科技大学学报, 2012, 41(1): 104-109.

# 致谢

行文至此，毕业设计已经进入尾声了。从当初的选题、查阅相关资料，到现在完成论文的编写，一路的酸甜苦辣已回荡在脑海之中。在此，首先感谢我的指导老师周应华老师，他在我毕业设计完成的过程中提供了很多建议，对于毕业论文中存在的问题耐心地指导。

同时，我还要感谢在毕业设计期间和我交流并给我建议的同学。正是通过交流和探讨，我才能发现自己在某些方面的不足，才能使毕业设计逐渐的完善。

最后，再次对给予我帮助的导师和同学表示由衷的感谢。

# 附录A 文献翻译

## 一、英文原文

**New Associative Classification Method Based on Rule Pruning for Classification of Datasets**

**KHAIRAN D. RAJAB**

Abstract: In data mining, a rule-based classification approach called Associative Classification (AC) normally builds accurate classifiers from supervised learning data sets. It extracts ‘‘If-Then’’ rules and associates each of the generated rules with two computed parameters; support and confidence. These two parameters are utilized to differentiate the rules’ superiority during the building of a classifier’s step. In current AC algorithms, whenever a rule is inserted into a classifier, all of its corresponding training data is discarded. However, the discarded data actually are used to compute support and confidence of other rules and will affect other lower ranked rules since rules normally have common training data examples. The use of static support and confidence may result in very large less-accurate classifiers.Thus, a procedure that amends other rules’ support and confidence is important. This paper proposes a new procedure named Active Pruning Rules (APR) to overcome the above problem so then the classifiers’ performance - especially predictive accuracy and reducing rule redundancy - will be further improved. The experimental results obtained from a number of University of California Irvine (UCI) data sets and real adult autism classification data set showed that APR is highly competitive to other AC and rule-based classifiers and often produces smaller yet more predictive classifiers.

**Introduction**

Most AC algorithms induce the knowledge (rules) from the input data, and then construct a predictive model that is named the classifier. The two main parameters connected with each rule are support and confidence .

The process of choosing the rules during the building of the classifier is the concern of this article. In this context, the current AC methods evaluate the induced rules on the input data set one by one after sorting them based on a rule ranking procedure, i.e., (rules’ confidence, support, length). Starting from the highest ranked rule downward, each training data will be evaluated and covered by a single rule only. When a training example is covered by a certain rule, it will be discarded and that rule will become part of the classifier. The evaluation process continues until all of the rules have been evaluated or no more data can be removed. Only at this time, is the classifier generated. The classifier will contain only rules that were covered in the training examples and all other rules are removed since they are redundant. This paper investigates a major shortcoming associated with AC algorithms during the process of rule evaluation. Specifically, when removing the training examples connected with a classifier’s rules whenever a rule gets evaluated against a training data set. This problem was raised by in which the authors proposed a multi-label rules mechanism. We argue that when training examples are discarded after a rule is evaluated, this removal affects other lower ranked rules since rules normally have common training data examples (items inside the data). Current AC algorithms in the literature assume that a training example can be used to learn multiple rules. So these algorithms do not eliminate any overlap of

the training data amongst the generated rules, resulting in generating very large classifiers. The aim of this research is to overcome the data overlapping issue in AC mining by developing a new learning method to ensure that rules derived have no data overlapping thus reducing the size of the classifier and maintain predictive accuracy performance.

The above issue occurs during the process of inducing the rules and should be resolved when the algorithm chooses the classifier’s rules. Consider for instance rule R1: ‘‘If y1 and x2 then Class A’’, that has been evaluated against the training data set, assuming that all training data examples of R1 are deleted. All other lower ranked rules having the values (‘‘y1’’ and ‘‘x2’’) in their body are impacted because of R1’s data removal. This means that these rules’ confidence and support and support counts should now be updated to reflect R1’s data removal. When the training data set is frequently updated because of rule example removal, the rule rank should change as well during the process of the rule evaluation. If this is implemented, a fairer classifier that does not rely on static rules rank will result.

A new AC method that we call Active Pruning Rules (APR) is developed to resolve the discussed issue. For each rule inserted into the classifier, the algorithm not only deletes all of its connected training data but also removes said data from any other unevaluated rules. This results in decrementing the confidence and support values for some of the rule and changing the ranking of these rules before the next inline rule is evaluated. This can be viewed as a rule-pruning procedure that discards insufficient rules without having to evaluate them against the training data as with current AC algorithms, saving computing resources. Furthermore, amending the rule rank every time a rule is inserted into the classifier may generate classifiers with no rule redundancy. Most importantly, the algorithm significantly increases the predictive accuracy of the classifier as the results from the experiments section shows.

In the last decade, multiple research studies, which have reported the applicability of AC on various different applications including fraud detection, credit card scoring, bioinformatics, on-line security, medical diagnoses, text categorization and others. The wide spread of this classification approach is primarily due to the simplicity and interpretability of the rules in the classifier besides the high predictive accuracy of its classifiers.

In the literature, many algorithms utilize a variety of knowledge-reasoning methodologies, rule pruning, and class predictions for test data. Since AC methods suffer from the exponential growth of the rules, this article tackles this problem and proposes a novel pruning method hence this section focuses more on the rule pruning procedures rather than on general AC algorithms in the literature. Rule pruning is the key to success in AC and ensures that any classifiers derived are controllable and usable by the end users. In fact, without rule pruning, most generated classification systems will not be useful and might get discarded because of the large number of rules in the classifier. This might lead into difficulties in managing the classification systems and hence the managers and decision makers will not put these systems in practise.

One of the first developed AC algorithms is CBA . This algorithm utilizes the Apriori candidate generation function to discover class association rules from data sets. CBA introduces the database coverage pruning method to choose high predictive rules. This pruning method is similar to the way that classic greedy algorithms find the rules. The database coverage in relation to the pruning assumes that the rules are already discovered and ranked in order of the confidence and support values. Then starting with the top rule, this method evaluates the rule on the training data set for possible classification. If the current rule was able to classify at least a single training example, it will be put into the classifier and all of its classified data examples will be removed. The same test is repeated for all of the remaining potential rules until all of the data examples are classified or all rules have been tested. Any potential rule that was unable to classify at least one training example is discarded. A number of successive AC algorithms have adopted the CBA database coverage method like CBA and ACCF, uCBA, X-class, SPARK-CBA, and others.

**Result**

Thefigure illustrates that the proposed algorithm was able to reduce the classifier size of the AC algorithms yet did not negatively influence the predictive accuracy. This can be attributed to the new pruning performed by APR during the rule evaluation. APR always sustains strong rules by pruning any weak rules that do not maintain an acceptable support and confidence value during the rule evaluation step. In other words, removing training data instances from other rules whenever a rule becomes part of the classifier surely minimizes rule redundancy, and therefore has resulted in a more concise set of rules that maintain high predictive accuracy during the test data classification phase. It is obvious from the number of rule results that the CBA algorithm produced the largest classifiers despite performing multiple pruning runs followed by C4.5. The algorithm that derived the least numbers of rules is RIPPER since it follows a separate-and-conquer strategy while inducing the rules beside multiple levels of rule pruning. RIPPER normally employs validation set pruning and testing set pruning to only keep perfect rules (rules with low data coverage yet high expected accuracy). However, despite the fewer number of rules generated by RIPPER, this algorithm’s classifiers often suffer from low predictive accuracy when compared to AC and decision tree approaches.

AC methods suffer from a large number of rules, which is a problem inherited from association rule mining. This problem limits the use of AC in application domains due to the very large classifiers that might be derived, which is hard to control by the end users. This article proposed a novel rule pruning procedure in AC; APR that amends the rules’ position in the classifier every time that a rule is inserted rather than relying on the initial rules’ frequency computed from the training data set. This results in only keeping rules with actual data coverage in the classifier and discards any rule whose data frequency has dropped below the acceptable minimum frequency. The influence of this new procedure was apparent on the classifiers size and on the predictive accuracy of the APR classifiers. The results have been obtained from experimenting on a number of data sets from the University of Irvine’s (UCI) data repository & real adult ASD dataset and using known AC, covering and decision tree algorithms. The experimental results showed that APR is very competitive with reference to the predictive accuracy in relation to the C4.5, RIPPER, and CBA algorithms. In fact, the new classifiers procedure of APR showed an improvement in the classification accuracy when compared to those of C4.5, RIPPER, and CBA. Furthermore, APR regularly generated fewer rules yet had a more efficient training procedure than CBA due to the new pruning that discards any weak rules while the classifier is progressing. In the near future, we intend to apply APR on unstructured application data such as text mining to

seek any possible challenges or improvements.

In near future, the proposed AC will be extended into two directions. The first issue on how to deal with unstructured features related to medical diagnosis particularly in ASD and other behavioural applications. APR will need particular extension in dealing with variables with sparse possible values or variables that are need transformation before utilizing them during the training step. The second possible area of extending the current work is to integrate APR with ensembles learning approach in which multiple classifiers can be used to come up with a collective decision for assigning the class table.

## 二、英文翻译

**基于规则修剪的数据集关联分类新方法**

摘要:在数据挖掘中，一种基于规则的分类方法——关联分类(Associative classification, AC)通常从监督学习数据集中构建准确的分类器。它提取“If-Then”规则，并将每个生成的规则与两个计算参数相关联;支持和信心。在分类器步骤的构建过程中，使用这两个参数来区分规则的优越性。在当前的AC算法中，每当将规则插入分类器时，其对应的所有训练数据都会被丢弃。然而，丢弃的数据实际上是用来计算其他规则的支持度和置信度的，并且会影响其他排名较低的规则，因为规则通常有共同的训练数据示例。使用静态支持和置信度可能会导致非常大的不太准确的分类器。因此，修正其他规则的支持和信任的程序是重要的。为了克服上述问题，本文提出了一种新的方法——主动剪枝规则(Active Pruning Rules, APR)，从而进一步提高了分类器的性能，特别是预测精度和减少规则冗余。来自加州大学欧文分校(UCI)数据集和真实成人自闭症分类数据集的实验结果表明，APR与其他基于AC和规则的分类器相比具有很强的竞争力，并且通常产生更小但更具预测性的分类器。

**介绍**

大多数AC算法从输入数据中归纳出知识(规则)，然后构造一个预测模型，称为分类器。与每条规则相关的两个主要参数是支持度和置信度。本文所关注的是分类器构建过程中规则的选择过程。在这种情况下，现有的AC方法根据规则排序过程，即(规则置信度、支持度、长度)，对输入数据集上的诱导规则进行排序后，逐一评估。从排名最高的规则开始向下，每个训练数据将仅由单个规则评估和覆盖。当一个训练样例被某个规则覆盖时，它将被丢弃，该规则将成为分类器的一部分。评估过程将继续进行，直到评估完所有规则或无法删除更多数据为止。只有在这个时候，分类器才会生成。分类器将只包含训练示例中涵盖的规则，而所有其他规则都被删除，因为它们是冗余的。本文研究了AC算法在规则评估过程中的一个主要缺陷。具体来说，当根据训练数据集对规则进行评估时，删除与分类器规则相关的训练示例。针对这一问题，作者提出了一种多标签规则机制。我们认为，当在评估规则后丢弃训练样例时，这种删除会影响其他排名较低的规则，因为规则通常具有公共训练数据样例(数据中的项)。目前文献中的交流算法假设一个训练样例可以用来学习多个规则。所以这些算法并没有消除任何重叠生成规则中的训练数据，导致生成非常大的分类器。本研究的目的是通过开发一种新的学习方法来克服AC挖掘中的数据重叠问题，以确保导出的规则没有数据重叠，从而减少分类器的大小并保持预测精度性能。

上述问题发生在规则的归纳过程中，需要在算法选择分类器规则时加以解决。例如，考虑规则R1:“如果y1和x2则为A类”，它已经对训练数据集进行了评估，假设R1的所有训练数据示例都被删除。所有其他排名较低的规则在其主体中具有值(“y1”和“x2”)，因为R1的数据删除而受到影响。这意味着这些规则的置信度和支持度计数现在应该被更新，以反映R1的数据删除。当训练数据集由于规则示例的移除而频繁更新时，在规则评估过程中，规则的等级也应该发生变化。如果实现了这一点，将产生一个不依赖于静态规则排名的更公平的分类器。

为了解决这一问题，提出了一种新的交流方法——主动剪枝规则(APR)。对于插入到分类器中的每个规则，算法不仅删除其所有连接的训练数据，而且还从任何其他未评估的规则中删除所述数据。这将导致降低某些规则的置信度和支持值，并在评估下一个内联规则之前更改这些规则的排名。这可以看作是一个规则修剪过程，它可以丢弃不充分的规则，而不必像当前的AC算法那样根据训练数据对它们进行评估，从而节省了计算资源。此外，每次将规则插入分类器时修改规则秩可能会生成没有规则冗余的分类器。最重要的是，该算法显著提高了分类器的预测精度，如实验部分的结果所示。

在过去十年中，多项研究表明了AC在各种不同应用中的适用性，包括欺诈检测、信用卡评分、生物信息学、在线安全、医疗诊断、文本分类等。这种分类方法的广泛传播主要是由于分类器中的规则的简单性和可解释性以及其分类器的高预测精度。

在文献中，许多算法利用各种知识推理方法、规则修剪和测试数据的类预测。由于AC方法受到规则指数增长的影响，本文解决了这一问题，并提出了一种新的修剪方法，因此本节更多地关注规则修剪过程，而不是文献中一般的AC算法。规则修剪是AC中成功的关键，并确保任何分类器最终可以控制和使用衍生数据。事实上，如果不进行规则修剪，大多数生成的分类系统将没有用处，并且可能会因为分类器中存在大量规则而被丢弃。这可能会导致管理分类系统的困难，因此管理人员和决策者不会将这些系统付诸实践。

最早开发的交流算法之一是CBA。该算法利用Apriori候选生成函数从数据集中发现类关联规则。CBA引入了数据库覆盖修剪方法来选择高质量规则。这种修剪方法类似于经典贪婪算法查找规则的方式。与修剪相关的数据库覆盖率假设已经发现了规则，并按照置信度和支持值的顺序进行排序。然后，该方法从顶部规则开始，在训练数据集上评估可能的分类规则。如果当前规则能够对至少一个训练样例进行分类，则将其放入分类器中，并删除其所有分类数据样例。对所有剩余的潜在规则重复相同的测试，直到对所有数据示例进行分类或对所有规则进行测试为止。任何无法对至少一个训练样本进行分类的潜在规则都将被丢弃。相继出现的许多AC算法都采用了CBA数据库覆盖方法，如CBA和ACCF、uCBA、X-class、SPARK-CBA等。

**结果**

从图中可以看出，该算法能够减少AC算法的分类器大小，但不会对预测精度产生负面影响。这可以归因于APR在规则求值期间执行的新修剪。在规则评估步骤中，APR总是通过修剪任何不能维持可接受的支持和置信度值的弱规则来维持强规则。换句话说，每当规则成为分类器的一部分时，从其他规则中删除训练数据实例肯定会最小化规则冗余，因此会产生一组更简洁的规则，在测试数据分类阶段保持较高的预测准确性。从规则结果的数量可以明显看出，尽管CBA算法在C4.5之后执行了多次修剪运行，但它产生了最大的分类器。推导出规则数最少的算法是RIPPER，因为它在引入规则时遵循了分而治之的策略，并在多级规则修剪之外引入规则。RIPPER通常采用验证集修剪和测试集修剪，只保留完美的规则(数据覆盖率低但期望精度高的规则)。然而，尽管RIPPER生成的规则数量较少，但与AC和决策树方法相比，该算法的分类器的预测精度往往较低。

AC方法存在大量的规则，这是关联规则挖掘遗留下来的问题。这个问题限制了AC在应用程序域中的使用，因为可能派生出非常大的分类器，这是最终用户难以控制的。本文提出了一种新的AC规则剪枝程序APR，每次插入规则时修改规则在分类器中的位置，而不是依赖于从训练数据集中计算的初始规则频率。这将导致在分类器中只保留具有实际数据覆盖的规则，并丢弃数据频率低于可接受的最小频率的任何规则。这一新程序对分类器的大小和APR分类器的预测精度有明显的影响。结果是通过对欧文大学(UCI)数据存储库和真实成人ASD数据集的大量数据集进行实验，并使用已知的AC，覆盖和决策树算法获得的。实验结果表明，相对于C4.5、RIPPER和CBA算法，APR在预测精度方面具有很强的竞争力。事实上，与C4.5、RIPPER和CBA的分类器相比，APR的新分类器程序的分类精度有所提高。此外，APR定期生成更少的规则，但具有比CBA更有效的训练过程，因为在分类器进行过程中，新的修剪会丢弃任何弱规则。在不久的将来，我们打算将APR应用于非结构化应用程序数据，如文本挖掘寻找任何可能的挑战或改进。

在不久的将来，提议的AC将扩展至两个方向。第一个问题是如何处理与医学诊断有关的非结构化特征，特别是在自闭症谱系障碍和其他行为应用中。在处理具有稀疏可能值的变量或在训练阶段使用之前需要转换的变量时，APR需要特别的扩展。扩展当前工作的第二个可能领域是将APR与集成学习方法集成在一起，其中可以使用多个分类器来提出分配类表的集体决策。

# 附录B

## 一、源程序

import pandas as pd

from server.CBANB.read import read

from server.CBANB.pre\_processing import pre\_process

from server.CBANB.cba\_rg import rule\_generator

from server.CBANB.cba\_cb\_m1 import classifier\_builder\_m1

from server.CBANB.cba\_cb\_m2 import classifier\_builder\_m2

import time

import random

def get\_accuracy(apr,test):

temp=[]

actual=[x[-1] for x in test]

count=0

for i in range(len(test)):

flag1=True

for j in range(len(apr.rule\_list)):

flag=True

for item in apr.rule\_list[j].condition\_set:

if test[i][item]!=apr.rule\_list[j].condition\_set[item]:

flag=False

break

if flag:

temp.append(apr.rule\_list[j].class\_label)

if temp[-1]==actual[i]:

count+=1

flag1=False

break

if flag1:

temp.append(apr.default\_class)

if temp[-1]==actual[i]:

count+=1

res=count/len(test)

return res

def cba\_m1\_prune(file, minsup, minconf):

data, attributes, value\_type = read('./dataset/' + file + '.csv')

random.shuffle(data)

dataset = pre\_process(data, attributes, value\_type)

train\_ratio = 0.8

train\_size = int(len(dataset) \* train\_ratio)

random.shuffle(dataset)

training\_dataset = dataset[:train\_size]

test\_dataset = dataset[train\_size:]

start\_time = time.time()

cars = rule\_generator(training\_dataset, minsup, minconf)

classifier\_m1 = classifier\_builder\_m1(cars, training\_dataset)

cars.prune\_rules(training\_dataset)

cars.rules = cars.pruned\_rules

classifier\_m1.print()

print(classifier\_m1.default\_class)

accuracy = get\_accuracy(classifier\_m1, test\_dataset)

end\_time = time.time()

cost = end\_time - start\_time

all\_rules = classifier\_m1.all\_rules

return {'accuracy': accuracy, 'cost': cost,'rules': all\_rules,'default': classifier\_m1.default\_class,'nums':len(all\_rules)}

def cross\_validate\_m1(file, minsup, minconf):

data, attributes, value\_type = read('./dataset/' + file + '.csv')

random.shuffle(data)

dataset = pre\_process(data, attributes, value\_type)

block\_size = int(len(dataset) / 10)

split\_point = [k \* block\_size for k in range(0, 10)]

split\_point.append(len(dataset))

total\_time = 0

total\_car\_number = 0

total\_classifier\_rule\_num = 0

accuracy\_total = 0

for k in range(len(split\_point)-1):

print("\nRound %d:" % k)

training\_dataset = dataset[:split\_point[k]] + dataset[split\_point[k+1]:]

test\_dataset = dataset[split\_point[k]:split\_point[k+1]]

start\_time = time.time()

cars = rule\_generator(training\_dataset, minsup, minconf)

cars.prune\_rules(training\_dataset)

cars.rules = cars.pruned\_rules

cars.print\_pruned\_rule()

classifier\_m1 = classifier\_builder\_m1(cars, training\_dataset)

accuracy = get\_accuracy(classifier\_m1, test\_dataset)

end\_time = time.time()

total\_time += end\_time - start\_time

accuracy\_total += accuracy

total\_car\_number += len(cars.rules)

total\_classifier\_rule\_num += len(classifier\_m1.rule\_list)

accuracy = accuracy\_total / 10 \* 100

num\_rules = int(total\_classifier\_rule\_num) / 10

cost = total\_time / 10

car\_number = total\_car\_number / 10

return {'accuracy': round(accuracy,3), 'num\_rules': num\_rules, 'cost':round(cost,3)}

def cba\_m2\_prune(file, minsup, minconf):

data, attributes, value\_type = read('./dataset/' + file + '.csv')

dataset = pre\_process(data, attributes, value\_type)

train\_ratio = 0.8

train\_size = int(len(dataset) \* train\_ratio)

random.shuffle(dataset)

training\_dataset = dataset[:train\_size]

test\_dataset = dataset[train\_size:]

start\_time = time.time()

cars = rule\_generator(training\_dataset, minsup, minconf)

classifier\_m2 = classifier\_builder\_m2(cars, training\_dataset)

cars.prune\_rules(training\_dataset)

cars.rules = cars.pruned\_rules

classifier\_m2.print()

accuracy = get\_accuracy(classifier\_m2, test\_dataset)

end\_time = time.time()

cost = end\_time-start\_time

all\_rules = classifier\_m2.all\_rules

print(len(classifier\_m2.rule\_list),len(all\_rules))

return {'accuracy': accuracy, 'cost': cost,'rules': all\_rules,'default':classifier\_m2.default\_class,'nums':len(all\_rules)}

def cross\_validate\_m2(file, minsup, minconf):

data, attributes, value\_type = read('./dataset/' + file + '.csv')

random.shuffle(data)

dataset = pre\_process(data, attributes, value\_type)

block\_size = int(len(dataset) / 10)

split\_point = [k \* block\_size for k in range(0, 10)]

split\_point.append(len(dataset))

total\_time = 0

total\_car\_number = 0

total\_classifier\_rule\_num = 0

accuracy\_total = 0

for k in range(len(split\_point)-1):

print("\nRound %d:" % k)

training\_dataset = dataset[:split\_point[k]] + dataset[split\_point[k+1]:]

test\_dataset = dataset[split\_point[k]:split\_point[k+1]]

start\_time = time.time()

cars = rule\_generator(training\_dataset, minsup, minconf)

cars.prune\_rules(training\_dataset)

cars.rules = cars.pruned\_rules

cars.print\_pruned\_rule()

# print(cars.all\_rules)

classifier\_m2 = classifier\_builder\_m2(cars, training\_dataset)

accuracy = get\_accuracy(classifier\_m2, test\_dataset)

end\_time = time.time()

total\_time += end\_time - start\_time

accuracy\_total += accuracy

total\_car\_number += len(cars.rules)

total\_classifier\_rule\_num += len(classifier\_m2.rule\_list)

accuracy = accuracy\_total / 10 \* 100

num\_rules = int(total\_classifier\_rule\_num) / 10

cost = total\_time / 10

return {'accuracy': round(accuracy,3), 'num\_rules': num\_rules, 'cost':round(cost,3)}

def get\_preprocess(file):

df = pd.read\_csv('./dataset/' + file + '.csv')

data = df.values.tolist()

attributes = df.columns.tolist()

value\_type = df.dtypes.iloc

divided = pre\_process(data, attributes, value\_type)

divided.insert(0,attributes)

return divided

import random

import time

from server.APR.apr\_cb\_m1 import is\_satisfy, sort\_dict, classifier\_builder\_m1

from server.APR.apr\_rg import rule\_generator

from server.APR.pre\_processing import pre\_process

from server.APR.read import read

def get\_accuracy(apr,test):

temp=[]

actual=[x[-1] for x in test]

count=0

for i in range(len(test)):

flag1=True

for j in range(len(apr.rule\_list)):

flag=True

for item in apr.rule\_list[j].condition\_set:

if test[i][item]!=apr.rule\_list[j].condition\_set[item]:

flag=False

break

if flag:

temp.append(apr.rule\_list[j].class\_label)

if temp[-1]==actual[i]:

count+=1

flag1=False

break

if flag1:

temp.append(apr.default\_class)

if temp[-1]==actual[i]:

count+=1

res=count/len(test)

return res

def apr(file, minsup, minconf):

data, attributes, value\_type = read('./dataset/' + file + '.csv')

dataset = pre\_process(data, attributes, value\_type)

train\_ratio = 0.8

train\_size = int(len(dataset) \* train\_ratio)

random.shuffle(dataset)

training\_dataset = dataset[:train\_size]

test\_dataset = dataset[train\_size:]

start\_time = time.time()

cars = rule\_generator(training\_dataset, minsup, minconf)

arr=list(cars.rules\_list)

max=-1

for i in range(len(arr)):

if len(arr[i].condition\_set)>max:

max=len(arr[i].condition\_set)

T=[[] for i in range(max)]

for i in range(len(arr)):

T[len(arr[i].condition\_set)-1].append(arr[i])

u=[]

for i in range(len(T)):

T[i]=sort\_dict(T[i])

for j in T[i]:

u.append(j)

classifier = classifier\_builder\_m1(training\_dataset,minsup,len(training\_dataset),u)

classifier.print()

accuracy = get\_accuracy(classifier,test\_dataset)

end\_time = time.time()

cost = end\_time - start\_time

return {'accuracy': accuracy, 'cost': cost, 'rules': classifier.all\_rules,'default':classifier.default\_class,'nums':len(classifier.all\_rules)}

def cross\_validate\_apr(file, minsup, minconf):

data, attributes, value\_type = read('./dataset/' + file + '.csv')

random.shuffle(data)

dataset = pre\_process(data, attributes, value\_type)

block\_size = int(len(dataset) / 10)

split\_point = [k \* block\_size for k in range(0, 10)]

split\_point.append(len(dataset))

total\_time = 0

total\_car\_number = 0

total\_classifier\_rule\_num = 0

total\_accuracy = 0

for k in range(len(split\_point)-1):

print("\nRound %d:" % k)

training\_dataset = dataset[:split\_point[k]] + dataset[split\_point[k+1]:]

test\_dataset = dataset[split\_point[k]:split\_point[k+1]]

start\_time = time.time()

cars = rule\_generator(training\_dataset, minsup, minconf)

arr=list(cars.rules\_list)

max=-1

for i in range(len(arr)):

if len(arr[i].condition\_set)>max:

max=len(arr[i].condition\_set)

T=[[] for i in range(max)]

for i in range(len(arr)):

T[len(arr[i].condition\_set)-1].append(arr[i])

u=[]

for i in range(len(T)):

T[i]=sort\_dict(T[i])

for j in T[i]:

u.append(j)

classifier= classifier\_builder\_m1(training\_dataset,minsup,len(training\_dataset),u)

classifier.print()

accuracy = get\_accuracy(classifier, test\_dataset)

end\_time = time.time()

total\_time += end\_time - start\_time

total\_accuracy += accuracy

total\_car\_number += len(cars.rules)

total\_classifier\_rule\_num += len(classifier.rule\_list)

accuracy = total\_accuracy / 10 \*100

total\_rules = total\_classifier\_rule\_num / 10

cost = total\_time / 10

return {'accuracy': round(accuracy,3), 'num\_rules': total\_rules, 'cost':round(cost,3)}

import json

from server.CMAR.CMAR\_Classifier import \*

from server.CMAR.CR\_Tree import \*

from server.CMAR.cbaLib.pre\_processing import \*

from server.CMAR.cbaLib.validation import \*

from server.CMAR.CMAR\_Classifier import get\_acc

def str2numerical(data, value\_type):

size = len(data)

columns = len(data[0])

for i in range(size):

for j in range(columns-1):

if value\_type[j] == 'float64' and data[i][j] != '?':

data[i][j] = float(data[i][j])

return data

def read(data\_path):

df = pd.read\_csv(data\_path)

data = df.values.tolist()

attributes = df.columns.tolist()

value\_type = df.dtypes.iloc

data = str2numerical(data, value\_type)

return data, attributes, value\_type

def get\_cmar\_result(file,minSup,minConf):

data, attributes, value\_type = read('./dataset/' + file + '.csv')

random.shuffle(data)

dataset = pre\_process(data, attributes, value\_type)

# print(attributes)

train\_ratio = 0.9

train\_size = int(len(dataset) \* train\_ratio)

minSup = int(train\_size \* minSup)

training\_dataset = dataset[:train\_size]

test\_dataset = dataset[train\_size:]

train\_dataset\_to\_feed = []

for data in training\_dataset:

train\_dataset\_to\_feed.append(convert\_data\_to\_dataentry(data, attributes))

print("dataentry example is")

train\_dataset\_to\_feed[0].display()

test\_dataset\_to\_feed = []

for data in test\_dataset:

dataentry = convert\_data\_to\_dataentry(data, attributes)

test\_dataset\_to\_feed.append(dataentry)

start = time.time()

rules = get\_rules(train\_dataset\_to\_feed, minSup,minConf)

cr\_rule\_list = rules

print('rule example is')

root, header\_table = createCRtree(cr\_rule\_list)

tree\_pruned\_rules = root.getAllRules(header\_table)

retained\_rules, default\_label = pruneByCoverage(train\_dataset\_to\_feed, tree\_pruned\_rules)

classifier = CMARClassifier(tree\_pruned\_rules, default\_label, train\_dataset\_to\_feed, len(train\_dataset\_to\_feed))

end = time.time()

cost = end - start

accuracy,rules = get\_acc(classifier, test\_dataset\_to\_feed,attributes)

seen = {}

result = []

for item in rules:

key = json.dumps(item[0])

if key not in seen:

seen[key] = True

result.append(item)

rules = result

return {'accuracy': accuracy, 'cost': cost,'rules':rules,'default':classifier.default\_label,'nums':len(rules)}

def cross\_validate\_cmar(file,minSup,minConf):

data, attributes, value\_type = read('./dataset/' + file + '.csv')

random.shuffle(data)

dataset = pre\_process(data, attributes, value\_type)

block\_size = int(len(dataset) / 10)

split\_point = [k \* block\_size for k in range(0, 10)]

split\_point.append(len(dataset))

total\_acc = 0

total\_time = 0

total\_rules\_nums = 0

train\_size = int(len(dataset) \* 0.9)

minSup = int(train\_size \* minSup)

for k in range(len(split\_point) - 1):

print("\nRound %d:" % k)

training\_dataset = dataset[:split\_point[k]] + dataset[split\_point[k + 1]:]

test\_dataset = dataset[split\_point[k]:split\_point[k + 1]]

train\_dataset\_to\_feed = []

for data in training\_dataset:

train\_dataset\_to\_feed.append(convert\_data\_to\_dataentry(data, attributes))

print("dataentry example is")

train\_dataset\_to\_feed[0].display()

test\_dataset\_to\_feed = []

for data in test\_dataset:

dataentry = convert\_data\_to\_dataentry(data, attributes)

test\_dataset\_to\_feed.append(dataentry)

start = time.time()

rules = get\_rules(train\_dataset\_to\_feed, minSup, minConf)

cr\_rule\_list = rules

root, header\_table = createCRtree(cr\_rule\_list)

tree\_pruned\_rules = root.getAllRules(header\_table)

retained\_rules, default\_label = pruneByCoverage(train\_dataset\_to\_feed, tree\_pruned\_rules)

classifier = CMARClassifier(tree\_pruned\_rules, default\_label, train\_dataset\_to\_feed, len(train\_dataset\_to\_feed))

accuracy, rules = get\_acc(classifier, test\_dataset\_to\_feed, attributes)

end = time.time()

total\_time = end - start

seen = {}

result = []

for item in rules:

key = json.dumps(item[0])

if key not in seen:

seen[key] = True

result.append(item)

rules = result

total\_time = total\_time + (end - start)

total\_acc += accuracy

total\_rules\_nums += len(rules)

return {'accuracy': round(total\_acc / 10 \* 100,3), 'num\_rules': total\_rules\_nums / 10, 'cost': round(total\_time / 10,3)}