浙江大学

本科实验报告

课程名称：数据挖掘导论

姓　　名：张溢弛

学　　院：计算机科学与技术学院

专　　业：软件工程

学　　号：3180103772

指导教师：李石坚

2021 年 6 月 7 日

浙江大学 实验报告

课程名称：数据挖掘导论　　实验类型：综合

实验项目名称：决策树分类器的实现与性能对比

学生姓名：张溢弛　专业：软件工程　学号：3180103772

同组学生姓名：无　指导老师：李石坚

实验地点：曹西-503　实验日期：2021 年 6 月 7 日

目录

一 实验基本信息 **III**

1.1 实验要求 III

1.2 实验环境 III

1.3 实验内容 III

二 Aprioi 算法实现 **III**

2.1 算法描述 III

2.1.1 支持度和置信度 III

2.1.2 最小支持度和最小置信度 IV

2.1.3 Aprioi 算法具体步骤 IV

2.2 代码实现 IV

三 数据预处理 **V**

3.1 数据集基本情况 V

3.2 数据预处理过程 V

四 实验结果与分析 **VI**

4.1 不同阈值下的挖掘结果 VI

4.2 结果分析 VIII

五 实验总结 **IX**

一　实验基本信息

1.1　实验要求

基于 Kaggle 上的 Bank-marketing 数据集，自行实现 Aprioi 或 FP-Tree 算法之一（也可以都实现），挖掘该数据集中的频繁模式并分析。

数据集链接为：https://www.kaggle.com/henriqueyamahata/bank-marketing

1.2　实验环境

实验的基本环境如下：

- 操作系统：Windows 10
- 编程环境：Anaconda 4.10+Jupyter Notebook 4.4.0
- 编程语言：Python 3.6.5
- Python 库版本：Pandas 0.23.0，numpy 1.14.3

1.3　实验内容

本次实验中我针对 Kaggle 上的 Bank-marketing 数据集，实现了 Aprioi 算法并对该数据集进行了频繁模式的挖掘。

二　Aprioi 算法实现

2.1　算法描述

Aprioi 算法是一种非常经典的频繁模式和关联规则挖掘算法，最早提出的时候被用在购物商品数据集中来寻找顾客可能的购买模式。我们的目标就是从数据集 D 中挖掘出一系列关联规则 $A \to B$

2.1.1　支持度和置信度

关联规则挖掘中需要一些统计量作为评估的标准，而支持度和置信度就是 Aprioi 算法使用的评价标准，支持度定义为：

$$\text{support}(A \to B) = P(AB) \tag{1}$$

而置信度的定义为：

$$\text{confidence}(A \to B) = P(B|A) = \frac{P(AB)}{P(A)} \tag{2}$$

2.1.2 最小支持度和最小置信度

最小支持度：最小支持度就是人为规定的阈值，表示项集在统计意义上的最低重要性。最小置信度：最小置信度也是人为规定的阈值，表示关联规则最低可靠性。只有支持度与置信度同时达到了最小支持度与最小置信度，此关联规则才会被称为强规则。

2.1.3 Aprioi 算法具体步骤

Aprioi 算法需要数据数据集 D 和支持度阈值 α，具体步骤主要有如下几步：

- 扫描整个数据集，得到所有出现过的数据，作为候选频繁 1 项集。k=1，频繁 0 项集为空集
- 扫描数据计算候选频繁 k 项集的支持度
- 去除候选频繁 k 项集中支持度低于阈值的数据集, 得到频繁 k 项集。如果得到的频繁 k 项集为空，则直接返回频繁 k-1 项集的集合作为算法结果，算法结束。如果得到的频繁 k 项集只有一项，则直接返回频繁 k 项集的集合作为算法结果，算法结束。
- 基于频繁 k 项集，连接生成候选频繁 k+1 项集
- k=k+1 返回第 2 步继续迭代

2.2 代码实现

我在提交的代码文件 apriori 中实现了一个类 Apriori，该类就是 Apriori 算法的核心实现类，按照 Apriori 算法的步骤，我将代码总体框架设计如下：

```
# 实现Apriori算法的类
class Apriori:
    # 展示频繁模式挖掘的结果
    def show_freq_pattern(self, freq_1, freq_2, conf):

    # 对数据集进行扫描
    def scan_dataset(self, dataset, candidate_k, min_support=0.5):

    # 将频繁集进行组合
    def itemset_combination(self, freq_list, k):

    # 进一步寻找频繁的模式
    def pattern_mining(self, freq_set, data, minConf=0.5):

    # 计算置信度并剪枝
    def cal_confidence(self, freq_set, items, data, brl, min_conf=0.5):

    def cal_rules(self, freq_set, items, data, brl, min_conf=0.5):

    # 算法主函数
```

```
21    def apriori(self, dataset, min_support=0.5):
22        # 初始化1项的频繁集
```

具体的运行结果和分析在实验报告的第四部分。

三　数据预处理

3.1　数据集基本情况

Bank-marketing 数据集是一个用来预测客户是否会认购定期存款的数据集，该数据集以电话通信为基础，包含一系列用户的基本信息：

- age：用户的年龄
- job：用户的工作，有蓝领、经理、退休、自由职业、学生、失业等多种类别
- marital：婚姻状况，包括离婚，单身，未知等多个类别
- education：受教育情况
- default：是否有信用违规的情况
- housing：是否有房贷
- loan：是否有负债
- contact：联系方式
- y：是否会认购定期存款

以及其他若干信息特征，很显然这个数据集中的数据有数值，有字符，也有布尔类型，这很明显是无法直接使用 Aprioi 算法进行频繁模式挖掘的，因此我对这个数据集进行了一定的预处理，处理的方式如下。

3.2　数据预处理过程

Aprioi 算法最早是用在商品集预测的任务里面的，在这个问题中商品用不同的数字来代替，我认为在本次作业中，我们也可以将 Bank-marketing 数据集中的栏目看成是一系列的商品，并用不同的数字来表示这些特征，对于 Boolean 类型的特征来说这是比较容易完成的，而对于数值类型和字符串类型的特征，我才去了如下处理办法：

- 数值类型的特征采用区间划分的方式，比如年龄，小于 30 用 0 表示，30-50 用 1 表示，而 50 以上用 2 表示
- 字符串类型使用了一定的划分方式，比如 job 按照是否有正式的职业进行了一定的划分，学生、待业人员、退休等分到了一个类别中，marital 也按照是否已婚进行了划分

按照这种划分方式，我实现了数据集的预处理和转换，最终将原始的 Bank-marketing 数据集转换成了类似于商品集的列表，这一部分的代码在提交的 preprocess.py 中可以看到，同时我筛选出了数据集中最终结果为 yes(也就是认购了存款) 的记录，得到了最终的处理后的数据格式如下：

```
In [5]: from preprocess import *

        dataset = data_preprocess()
        for data in dataset:
            print(data)
```

```
[2, 4, 6, 8, 9, 12]
[2, 4, 5, 8, 9, 12]
[2, 4, 5, 8, 10, 12]
[2, 4, 5, 8, 9, 12]
[2, 4, 5, 8, 9, 12]
[2, 4, 5, 8, 9, 11]
[2, 4, 5, 8, 9, 12]
[0, 3, 6, 8, 9, 11]
[2, 4, 5, 8, 9, 12]
[2, 4, 5, 8, 10, 12]
[2, 4, 5, 8, 10, 12]
[2, 4, 5, 8, 9, 12]
[2, 4, 5, 8, 10, 11]
[2, 4, 5, 8, 9, 12]
[2, 4, 6, 8, 10, 12]
[0, 3, 6, 8, 9, 11]
[2, 4, 5, 8, 9, 12]
[2, 4, 6, 8, 9, 12]
[2, 4, 5, 8, 9, 12]
```

这里的数字分别代表了如下含义：

- 0 表示年龄 30 以下，1 表示年龄 30-50，2 表示 50 以上
- 3 表示无业，4 表示有正规职业
- 5 表示已婚，6 表示未婚
- 7 表示有信用违规记录，8 表示没有
- 9 表示有房子和房贷，10 表示没有
- 11 表示有负债，12 表示没有负债

四　实验结果与分析

4.1　不同阈值下的挖掘结果

完成了数据集的预处理和算法代码的编写之后，我使用了不同的支持度阈值来计算，因为数据量比较大，所以即使使用 0.8，0.9 等阈值，得到的频繁集还是非常丰富，这也说明 Bank-marketing 数据集中确实存在着一些频繁出现的用户模式。阈值是 0.5 时候的结果：

```
from apriori import Apriori
apriori = Apriori()
freq_sets, support = apriori.apriori(dataset)
rules = apriori.pattern_mining(freq_sets, support, 0.5)
```

```
[5] -----> [8] confidence: 1.0
[8] -----> [5] confidence: 0.5456896551724137
[4] -----> [2] confidence: 0.6610666666666667
[2] -----> [4] confidence: 0.9404400606980272
[2] -----> [8] confidence: 1.0
[8] -----> [2] confidence: 0.5681034482758621
[4] -----> [8] confidence: 1.0
[8] -----> [4] confidence: 0.8081896551724138
[9] -----> [8] confidence: 1.0
[8] -----> [9] confidence: 0.540301724137931
[4] -----> [12] confidence: 0.8552
[12] -----> [4] confidence: 0.8104624715693707
[12] -----> [8] confidence: 1.0
[8] -----> [12] confidence: 0.852801724137931
[12] -----> [8, 4] confidence: 0.8104624715693707
[4] -----> [8, 12] confidence: 0.8552
[8] -----> [4, 12] confidence: 0.6911637931034482
[4] -----> [8, 2] confidence: 0.6610666666666667
[2] -----> [8, 4] confidence: 0.9404400606980272
[8] -----> [2, 4] confidence: 0.5342672413793104
```

阈值是 0.8 的时候的结果：

```
freq_sets, support = apriori.apriori(dataset)
rules = apriori.pattern_mining(freq_sets, support, 0.8)
```

```
[5] -----> [8] confidence: 1.0
[2] -----> [4] confidence: 0.9404400606980272
[2] -----> [8] confidence: 1.0
[4] -----> [8] confidence: 1.0
[8] -----> [4] confidence: 0.8081896551724138
[9] -----> [8] confidence: 1.0
[4] -----> [12] confidence: 0.8552
[12] -----> [4] confidence: 0.8104624715693707
[12] -----> [8] confidence: 1.0
[8] -----> [12] confidence: 0.852801724137931
[12] -----> [8, 4] confidence: 0.8104624715693707
[4] -----> [8, 12] confidence: 0.8552
[2] -----> [8, 4] confidence: 0.9404400606980272
```

阈值是 0.9 的时候的结果：

```
freq_sets, support = apriori.apriori(dataset)
rules = apriori.pattern_mining(freq_sets, support, 0.9)
```

```
[5] ----->> [8] confidence: 1.0
[2] ----->> [4] confidence: 0.9404400606980272
[2] ----->> [8] confidence: 1.0
[4] ----->> [8] confidence: 1.0
[9] ----->> [8] confidence: 1.0
[12] ----->> [8] confidence: 1.0
[2] ----->> [8,4] confidence: 0.9404400606980272
```

阈值是 1 时候的结果：

```
freq_sets, support = apriori.apriori(dataset)
rules = apriori.pattern_mining(freq_sets, support, 1)
```

```
[5] ----->> [8] confidence: 1.0
[2] ----->> [8] confidence: 1.0
[4] ----->> [8] confidence: 1.0
[9] ----->> [8] confidence: 1.0
[12] ----->> [8] confidence: 1.0
```

同时我也对不购买存款的用户进行了一定的频繁模式挖掘，其结果如下：

```
freq_sets, support = apriori.apriori(dataset_2)
rules = apriori.pattern_mining(freq_sets, support, 0.9)
```

```
[2] ----->> [4] confidence: 0.9582149943586352
[2] ----->> [8] confidence: 0.999883282107147
[9] ----->> [8] confidence: 0.9999475588651738
[5] ----->> [4] confidence: 0.9221289515984998
[4] ----->> [8] confidence: 0.9999402967252754
[8] ----->> [4] confidence: 0.91659597756191
[5] ----->> [8] confidence: 0.9998660475084836
[12] ----->> [4] confidence: 0.9160507375012104
[12] ----->> [8] confidence: 0.9999031727076139
[2] ----->> [8,4] confidence: 0.9581371824300665
[12] ----->> [8,4] confidence: 0.9159861859729529
[5] ----->> [8,4] confidence: 0.9220396499374889
[2,12] ----->> [8,4] confidence: 0.9582893104873905
```

4.2 结果分析

上面的算法运行结果是用数字来表示某些特征的，在分析数据的阶段我们要将数值转化成对应的特征描述，我们可以从上面的运行结果中挖掘到如下信息 (要注意我分析的数据全部都是选择了认购定期存款的用户)：

- 大多数已婚人士一般都没有信用违规记录，并且也更倾向于认购定期存款 (5->8 的频繁模式)
- 大多数频繁模式反映出，没有信用违规记录的人往往更倾向于认购定期存款 (有很多 x->8 的频繁模式，而 8 代表没有信用违规记录)
- 同时,没有信用违规记录的人一般都有稳定的工作,并且没有个人负债 ([8]——»[4,12])，这几个要素往往互相关联，出现的非常频繁，这从多个阈值下的运行结果中都可以看出来，而**已婚、工作稳定，没有负债和信用违规记录的人也更容易购买定期存款服务**
- 年轻人往往更不喜欢定期存款，我们的频繁模式挖掘过程中没有看到任何关于年轻人的频繁模式 (缺少 0 和 1 相关的频繁模式)，说明在购买定期存款的人群中，年轻人非常少
- 有房贷的用户往往没有信用违规记录，这可能和失信用户难以贷款有关，同时这类人群也更倾向于定期存款

五　实验总结

本次实验中，我学习并实现了 Aprioi 算法，并对 Bank-marketing 数据集进行了一定的预处理，掌握了一定的数据预处理方法和算法实现能力，同时对频繁模式挖掘算法的结果进行了一定的分析，可以说收获比较大。