大数据原理与技术 Lab5

20337025 崔璨明

阅读报告

我选择了一篇名为“A fast APRIORI implementation”的论文进行阅读和复现，论文在2003发表，作者是Ferenc Bodon。

Apriori算法是一种挖掘频繁项集的算法，其核心思想是利用先验知识，即如果一个项集是频繁的，则它的所有子集也是频繁的。因此，Apriori算法通过逐层扫描数据集并生成候选项集的方式来发现频繁项集。

然而，随着数据集的增大，Apriori算法的执行速度会显著下降，因为它需要生成大量的候选项集并计算它们的支持度。为了解决这个问题，作者在文章中提出了一个优化版本的Apriori算法。

这个算法有几个优化点。首先，它使用哈希树来高效地计算项集的支持度，这样可以避免重复扫描数据集。其次，它使用了剪枝技术来消除不必要的候选项集的生成。最后，它使用了压缩数据结构来存储频繁项集，以减少内存占用。

作者通过实验比较了他们的算法（APRIORI BRAVE）和其他常见的Apriori算法的性能。实验结果表明，他们的算法在大多数情况下都比其他算法执行更快，而且内存占用也更少。

复现报告

作者进行实验时采用的是C++语言来编写程序，且没有公开源码，为了复现的便捷性，我采取了python来编写程序，进行复现。首先原始Apriori算法我们可以通过调库的方式来进行简单实现，具体代码见`apriori.py`文件，论文中的采用了哈希树结构的APRIORI BRAVE算法的实现我参考了[该仓库](https://github.com/asaini/Apriori)，具体代码见`apriori_brave.py`文件。论文中作者采用的数据集为T40I10D100K、T10I4D100K、和kosarak，并将他们的APRIORI BRAVE算法和几个经典的APRIORI算法进行了对比。而在我的复现中，由于这三个数据集过于庞大，程序运行的时间过长，因此我还增加了一个新的数据集“The Stop，Question and Frisk Data”作为复现的数据集。该数据集记录了当纽约警察局官员因怀疑居民参与犯罪而对居民进行询问/搜身时的事件，在经过去除无关数据、消除稀疏性整理后得到`data.csv`文件。

分别运行两个程序，参数设置设定最小支持度0.4，最小置信度0.5，程序降序输出置信度大于0.5的关联规则和算法的执行时间，运行结果如下：

普通的apriori算法：

```
PS D:\大三下资料\大数据\lab5> & E:/anaconda/python.exe d:/大三下资料/大数据/lab5/code/apriori.py

==High-confidence association rules (min_conf=50.0%)

[frisked] => [male] (Conf: 95.51118897299442%, Supp: 63.2732607791184%)
[BLACK] => [male] (Conf: 93.89067524115757%, Supp: 49.8740885103028 5%)
[MEDIUM] => [male] (Conf: 93.8355602918272%, Supp: 50.1324807848118%)
[young adult] => [male] (Conf: 93.64919777135437%, Supp: 48.95219742921584%)
[average] => [male] (Conf: 93.64298640434318%, Supp: 44.19164823614426%)
[male] => [frisked] (Conf: 68.12769669676749%, Supp: 63.2732607791184%)
[male] => [MEDIUM] (Conf: 53.978732935656524%, Supp: 50.1324807848118%)
[male] => [BLACK] (Conf: 53.70051635111876%, Supp: 49.8740885103028 5%)
[male] => [young adult] (Conf: 52.707896163912004%, Supp: 48.95219742921584%)

using time: 205.0097 ms
PS D:\大三下资料\大数据\lab5>
```

论文中的apriori brave算法：

```
PS D:\大三下资料\大数据\lab5> & E:/anaconda/python.exe d:/大三下资料/大数据/lab5/code/apriori_brave.py

==High-confidence association rules (min_conf=50.0%)
[frisked] => [male] (Conf: 95.51118897299442%, Supp: 63.2732607791184%)
[BLACK] => [male] (Conf: 93.89067524115757%, Supp: 49.87408851030285%)
[MEDIUM] => [male] (Conf: 93.8355602918272%, Supp: 50.1324807848118%)
[young adult] => [male] (Conf: 93.64919777135437%, Supp: 48.95219742921584%)
[average] => [male] (Conf: 93.64298640434318%, Supp: 44.19164823614426%)
[male] => [frisked] (Conf: 68.12769669676749%, Supp: 63.2732607791184%)
[male] => [MEDIUM] (Conf: 53.978732935656524%, Supp: 50.1324807848118%)
[male] => [BLACK] (Conf: 53.70051635111876%, Supp: 49.87408851030285%)
[male] => [young adult] (Conf: 52.707896163912004%, Supp: 48.95219742921584%)

using time: 111.1314 ms
PS D:\大三下资料\大数据\lab5>
```

可见两者在“The Stop，Question and Frisk Data”数据集上都得到了正确的结果，但apriori brave算法的执行速度要比普通的apriori算法快了几乎一倍，这和论文中的实验结果一致。

在论文中的数据集上进行实验，由于时间因素，我只设置了`min_supp=0.05`一组实验，得到表格如下：

数据集	**apriori**	**apriori brave**
T40I10D100K	2259.87 s	768.34 s
T10I4D100K	1726.04 s	441.534 s
kosarak	1403.88 s	357.92 s

论文中的实验结果：

min_ supp	Bodon impl.	Borgelt impl.	Goethals impl.	APRIORI_ BRAVE
0.05	8.57	10.53	25.1	8.3
0.030	10.73	11.5	41.07	10.6
0.020	15.3	13.9	53.63	14.0
0.010	95.17	155.83	207.67	100.27
0.009	254.33	408.33	458.7	178.93
0.0085	309.6	573.4	521.0	188.0

可见实验结果和论文中的大致相同，apriori brave算法的执行速度要比普通的apriori算法快。

实验感想

通过这次实验，我能够认真读完这篇论文并能根据指导进行实验结果的复现，这不仅加深了我对所学理论知识的印象，而且锻炼了我的动手能力，让我受益匪浅。