

《数据挖掘》

实验报告本

|  |  |
| --- | --- |
| 班 级： | **计172** |
| 学 号： | **10170811** |
| 姓 名： | **李震** |
| 指导教师： | **张静** |

信息科学与工程学院

2020年 5月

**实验一 Apriori算法研究与实现**

**一、实验目的**

通过编程实现Apriori算法，并在给定数据集上进行测试，完成频繁模式的挖掘。

**二、实验设备和环境**

**win10**

**Codeblocks**

**C++11**

**三、实验内容**

编程实现Apriori算法，并在数据集T10I4D1K.dat、T10I4D10K.dat、T10I4D50K.dat、T10I4D100K.dat上完成测试，测试时各个数据集上的频繁模式，支持度计数阈值如下表所示，请按照数据集从小到大的次序找到各个支持度计数下的频繁项集，并保存在文档中。

|  |  |  |  |
| --- | --- | --- | --- |
| 数据集 | 支持度计数 | | |
| T10I4D1K | 6 | 8 | 10 |
| T10I4D10K | 60 | 80 | 100 |
| T10I4D50K | 300 | 400 | 500 |
| T10I4D100K | 600 | 800 | 1000 |

**四、实验过程、结果及分析**

1.实验过程：

拿到实验数据前，实现了一个状态压缩存储集合的Apriori算法，但是只适用于值域小于60的数据，做实验时将代码改回用c++ STL的set存储集合，适用于任意范围的数据。

关键代码：

状态压缩版Apriori

1. **const** **int** maxn=60;
2. **typedef** **long** **long** LL;
3. map<LL,**int**>mp[maxn],ind;
4. map<LL,**int**>::iterator it,pre;
5. **inline** LL lowbit(LL x){**return** x&(-x);}
6. **inline** **int** count(LL x){
7. **int** ret=0;
8. **for**(;x;x-=lowbit(x))
9. ret++;
10. **return** ret;
11. }
12. **void** Apriori(LL init[],**int** N,**double** conf){
13. ind.clear();
14. **for**(**int** i=0;i<maxn;i++)
15. ind[1LL<<i]=i;
17. **static** vector<LL>tmp;
18. **int** siz;
19. mp[0].clear(),mp[1].clear();
20. **for**(**int** i=0;i<N;i++){
21. **for**(**int** j=init[i];j;j=(j-1)&init[i])
22. mp[0][j]++;
23. **for**(**int** j=init[i];j;j-=lowbit(j))
24. mp[1][lowbit(j)]++;
25. }
26. **for**(**int** i=1;!mp[i].empty();i++){
27. tmp.clear();
28. printf("C%d:\n",i);
29. **for**(it=mp[i].begin();it!=mp[i].end();it++){
30. printf("%I64d %d;",it->first,it->second);
31. **if**(it->second>=N\*conf)
32. tmp.push\_back(it->first);
33. }
34. printf("\nL%d:\n",i);
35. siz=tmp.size();
36. **for**(**int** j=0;j<siz;j++)
37. printf("%I64d ",tmp[j]);
38. printf("\n");
39. mp[i+1].clear();
40. **for**(**int** j=0;j<siz;j++)
41. **for**(**int** k=j+1;k<siz;k++)
42. **if**(count(tmp[j]|tmp[k])==i+1)
43. mp[i+1][tmp[j]|tmp[k]]++;
44. **for**(it=mp[i+1].begin();it!=mp[i+1].end();)
45. **if**(it->second==(i+1)\*i/2){
46. it->second=mp[0][it->first];
47. it++;
48. }
49. **else**{
50. pre=it++;
51. mp[i+1].erase(pre);
52. }
53. }
54. }

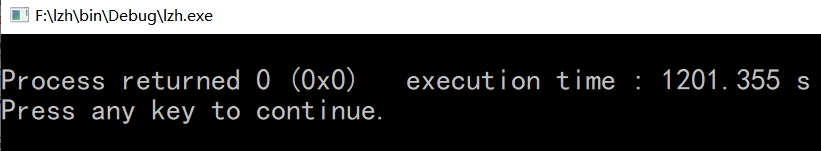
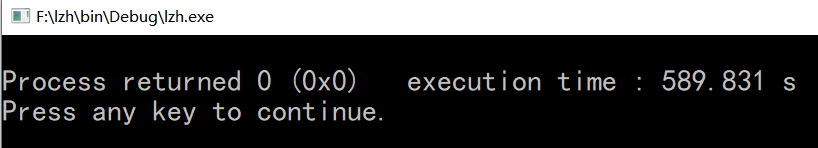
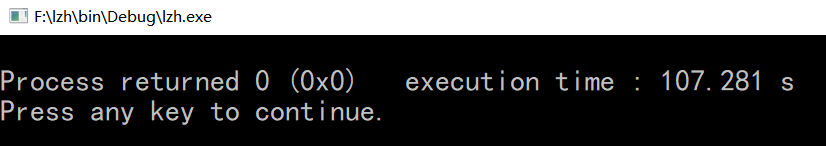
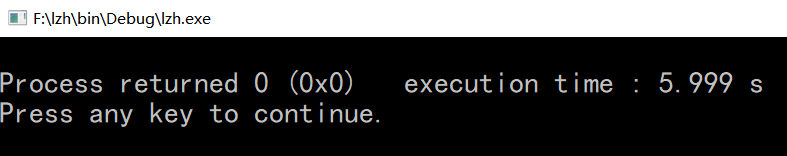
set版Apriori

1. #include<bits/stdc++.h>
2. **using** **namespace** std;
3. **void** readIn(**const** **char**\***FILE**,vector<set<**int**> >&S){
4. freopen(**FILE**,"r",stdin);
5. set<**int**>s;
6. S.clear();
7. **static** **char** str[1024];
8. **while**(gets(str+1)){
9. s.clear();
10. **int** len=strlen(str+1),tmp;
11. **for**(**int** i=0;i<len;i++){
12. **if**((!i||str[i]==' ')&&isdigit(str[i+1])){
13. sscanf(str+i+1,"%d",&tmp);
14. s.insert(tmp);
15. }
16. }
17. S.push\_back(s);
18. }
19. }
21. **void** Apriori(**const** **char**\***FILE**,vector<set<**int**> >s,**int** supp){
22. freopen(**FILE**,"w",stdout);
23. vector<set<**int**> >C,L;
24. vector<**int**>cnt;
25. C.clear(),cnt.clear();
26. **for**(**int** i=0;i<1000;i++){
27. set<**int**>tmp;
28. tmp.clear(),tmp.insert(i);
29. C.push\_back(tmp);
30. cnt.push\_back(0);
31. }
32. /\*for(int i=0;i<1000;i++)
33. printf("%d ",cnt[i]);
34. puts("");\*/
35. map<set<**int**>,**int**>F;
36. **for**(**int** T=1;;T++){
37. **for**(**int** i=0;i<C.size();i++){
38. **for**(**int** j=0;j<s.size();j++){
39. **bool** flag=1;
40. **for**(set<**int**>::iterator it=C[i].begin();it!=C[i].end();it++)
41. **if**(s[j].find(\*it)==s[j].end()){
42. flag=0;
43. **break**;
44. }
45. **if**(flag)
46. cnt[i]++;
47. }
48. }
49. L.clear();
50. **for**(**int** i=0;i<C.size();i++)
51. **if**(supp<=cnt[i])
52. L.push\_back(C[i]);
53. **if**(!L.size())**break**;
54. printf("L%d %d:\n",T,L.size());
55. **for**(**int** i=0;i<L.size();i++){
56. **for**(set<**int**>::iterator it=L[i].begin();it!=L[i].end();it++)
57. printf("%d ",\*it);
58. printf("\n");
59. }
60. F.clear();
61. **for**(**int** i=0;i<L.size();i++)
62. **for**(**int** j=i+1;j<L.size();j++){
63. set<**int**>tmp=L[i];
64. tmp.insert(L[j].begin(),L[j].end());
65. **if**(tmp.size()==T+1)
66. F[tmp]++;
67. }
68. C.clear(),cnt.clear();
69. **for**(map<set<**int**>,**int**>::iterator it=F.begin();it!=F.end();it++)
70. **if**(it->second==(T+1)\*T/2){
71. C.push\_back(it->first);
72. cnt.push\_back(0);
73. }
74. }
75. }
76. **int** main(){
77. vector<set<**int**> >S;
78. readIn("T1014D1K.dat",S);
79. Apriori("T1014D1K\_6.txt",S,6);
80. Apriori("T1014D1K\_8.txt",S,8);
81. Apriori("T1014D1K\_100.txt",S,10);
82. **return** 0;
83. }

实验结果：

对于一个数据集，运行一次算出三种支持度计数的频繁项集。

耗时如下（顺序是1K,10K,50K,100K）



结果分别保存在

T1014D1K\_6.txt

T1014D1K\_8.txt

T1014D1K\_10.txt

T1014D10K\_60.txt

T1014D10K\_80.txt

T1014D10K\_100.txt

T1014D50K\_300.txt

T1014D50K\_400.txt

T1014D50K\_500.txt

T1014D100K\_600.txt

T1014D100K\_800.txt

T1014D100K\_1000.txt

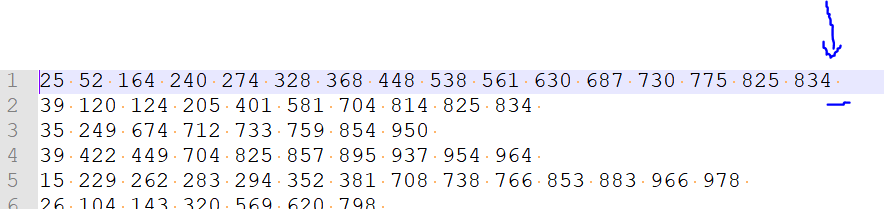
分析：当支持度计数最小时，每个数据集都能跑出一些频繁五项集，随着支持度计数增大，约束变强，频繁项集数量减少。总体来说算法运行时间可以接受，对答案进行抽查，结果基本正确。

检查的方法是在频繁项集中抽取，统计它在事物集中一共出现了多少次。代码如下：

1. #include<bits/stdc++.h>
2. **using** **namespace** std;
3. **int** check(**const** **char**\***FILE**,set<**int**>&S){
4. **int** ret=0;
5. freopen(**FILE**,"r",stdin);
6. set<**int**>s;
7. **static** **char** str[1024];
8. **while**(gets(str+1)){
9. s.clear();
10. **int** len=strlen(str+1),tmp;
11. **for**(**int** i=0;i<len;i++){
12. **if**((!i||str[i]==' ')&&isdigit(str[i+1])){
13. sscanf(str+i+1,"%d",&tmp);
14. s.insert(tmp);
15. }
16. }
17. **bool** occ=1;
18. **for**(set<**int**>::iterator it=S.begin();it!=S.end();it++)
19. **if**(s.find(\*it)==s.end()){
20. occ=0;
21. **break**;
22. }
23. ret+=occ;
24. }
25. **return** ret;
26. }
27. **int** main(){
28. set<**int**>s;
29. s.clear();
30. s.insert(27);s.insert(58);s.insert(354);s.insert(357);s.insert(480);s.insert(752);
31. **int** ans=check("T1014D1K.dat",s);
32. printf("%d\n",ans);
33. **return** 0;
34. }

2.困难和解决办法：

数据预处理有些困难，数据没有给出行数，可以判断读到文件末尾或者人工标注；没有给出每行数据的个数，可以选择每次读一个字符。如果是数字，则连接到上一个数字末尾，如果是换行符，就表示当前集合结束；如果是其他字符，则当前数字结束。这种处理方法类似于实现一个自动机，虽然效率很高，但是比较麻烦，而且注意到每行最后的字符是空格然后换行，就使得逻辑判断更复杂。（使用notepad++显示所有字符可以看清楚）



所以我换了种方法：直接读入整行字符串然后用sscanf()读入每个数字，代码简短很多。

3.存在问题

对于100K的数据，运行了约20分钟，算法效率有待提升。

代码使用STL的set<int>容器存储集合，使用vector<set<int> >存储集合列表，依赖c++11和编译命令-O2优化，不然效率会低一些。

4.收获和心得体会

动手实现了Apriori算法，对算法过程完全熟悉。值得一提的是，在频繁项集数量不大时，有一种优化比较有效：已知求时，不去枚举然后枚举的子集，而是将中的元素两两组合，判断合并出的集合大小是否是，如果是，将的出现次数。最后枚举所有出现过的，如果出现次数是，就说明属于。

5.算法的另一种实现

使用set<int>容器存储集合不够优雅，集合中项集较多时，合并集合效率低，仍有优化的空间，考虑使用C++ STL 的另一种容器bitset进行实现。bitset用来存储存储较长的01序列，注意到数据的值域为[0,999]，所以可以用一个长度为1000的bitset表示一个集合，第i位为1表示集合中不存在i这个数，第i位为0则不存在。bitset的优势在于可以对两个bitset快速做位运算，集合并就是按位或，集合交就是按位与。将Apriori算法中的集合全部用bitset<1000>维护，重新进行实验，

需要注意的是，bitset没有重载小于号，不能作为STL map的键值，因此改用STL unordered\_map。

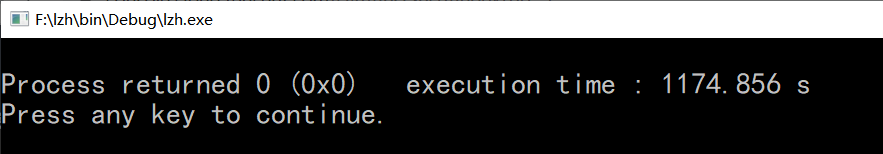
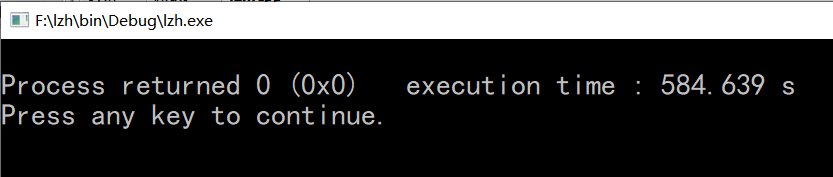
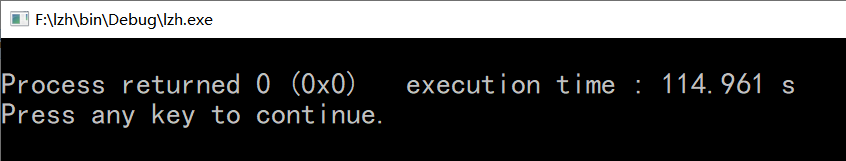
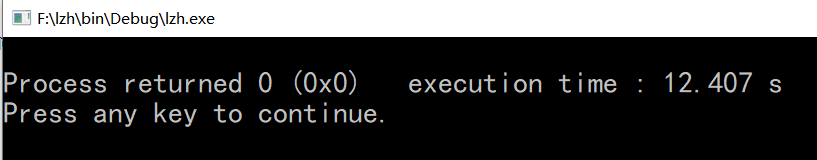
代码如下。

bitset版Apriori

1. #include<bits/stdc++.h>
2. **using** **namespace** std;
3. **const** **int** maxn=1000;
4. **void** readIn(**const** **char**\***FILE**,vector<bitset<maxn> >&S){
5. freopen(**FILE**,"r",stdin);
6. S.clear();
7. **static** **char** str[1024];
8. **while**(gets(str+1)){
9. bitset<maxn>s;
10. **int** len=strlen(str+1),tmp;
11. **for**(**int** i=0;i<len;i++){
12. **if**((!i||str[i]==' ')&&isdigit(str[i+1])){
13. sscanf(str+i+1,"%d",&tmp);
14. s[tmp]=1;
15. }
16. }
17. S.push\_back(s);
18. }
19. }
21. **void** Apriori(**const** **char**\***FILE**,vector<bitset<maxn> >s,**int** supp){
22. freopen(**FILE**,"w",stdout);
23. vector<bitset<maxn> >C,L;
24. vector<**int**>cnt;
25. C.clear(),cnt.clear();
26. **for**(**int** i=0;i<maxn;i++){
27. bitset<maxn>tmp;
28. tmp[i]=1;
29. C.push\_back(tmp);
30. cnt.push\_back(0);
31. }
32. /\*for(int i=0;i<1000;i++)
33. printf("%d ",cnt[i]);
34. puts("");\*/
35. unordered\_map<bitset<maxn>,**int**>F;
36. **for**(**int** T=1;;T++){
37. **for**(**int** i=0;i<C.size();i++)
38. **for**(**int** j=0;j<s.size();j++)
39. **if**((C[i]&s[j])==C[i])
40. cnt[i]++;
41. L.clear();
42. **for**(**int** i=0;i<C.size();i++)
43. **if**(supp<=cnt[i])
44. L.push\_back(C[i]);
45. **if**(!L.size())**break**;
46. printf("L%d %d:\n",T,L.size());
47. **for**(**int** i=0;i<L.size();i++){
48. **for**(**int** j=0;j<maxn;j++)
49. **if**(L[i][j])
50. printf("%d ",j);
51. printf("\n");
52. }
53. F.clear();
54. **for**(**int** i=0;i<L.size();i++)
55. **for**(**int** j=i+1;j<L.size();j++){
56. bitset<maxn>tmp=(L[i]|L[j]);
57. **if**(tmp.count()==T+1)
58. F[tmp]++;
59. }
60. C.clear(),cnt.clear();
61. **for**(unordered\_map<bitset<maxn>,**int**>::iterator it=F.begin();it!=F.end();it++)
62. **if**(it->second==(T+1)\*T/2){
63. C.push\_back(it->first);
64. cnt.push\_back(0);
65. }
66. }
67. }
68. **int** main(){
69. vector<bitset<maxn> >S;
70. readIn("T1014D10K.dat",S);
71. Apriori("T1014D10K\_60.txt",S,60);
72. Apriori("T1014D10K\_80.txt",S,80);
73. Apriori("T1014D10K\_100.txt",S,100);
74. **return** 0;
75. }

输出结果与set版Apriori顺序不同，内容相同。

耗时如下（顺序是1K,10K,50K,100K）



进行对比

耗时是指在三种支持度计数下分别运算三次的总耗时，单位为秒。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 耗时 | 1K | 10K | 50K | 100K |
| set | 5.999 | 107.281 | 589.831 | 1201.355 |
| bitset | 12.407 | 114.961 | 584.639 | 1174.856 |

折线图看起来不明显，所以使用柱状图。

可以看出，随着数据量增大，bitset更有优势。

在集合合并运算时，bitset的效率与项数无关，与值域有关，set反之。而且，数据集中每个事务的项数较少（每行的数字个数较少），可以预见，每行数字增多时，bitset优势更加明显。