**Appendix A**

**Source Code**

**Source code of apriori.java**

1. import java.io.BufferedReader;
2. import java.io.FileReader;
3. import java.io.IOException;
4. import java.text.DecimalFormat;
5. import javax.swing.JFrame;
6. import javax.swing.JOptionPane;
7. import javax.swing.JTextArea;
8. class S\_Itemset
9. {
10. class Symb
11. {
12. int code;
13. Symb next;
14. Symb(int code)
15. {
16. this.code=code;
17. next=null;
18. }
19. }//end: class Symb
21. Symb head=null;
22. Symb rp;
24. void separate(int code,int items)
25. {
26. int c;
27. Symb p=head;
28. Symb pp=null;
29. do
30. {
31. c=code%items;
32. code/=items;
33. if(p==null)
34. {
35. p=new Symb(c);
36. if(head == null)
37. {
38. head=p;
39. }
40. else
41. {
42. pp.next=p;
43. }
44. }
45. else
46. {
47. p.code=c;
48. }
49. pp=p;
50. p=p.next;
51. }while(code > 0);
52. if(p != null) p.code=-1;
53. rp=head;
54. }//end: void separate(int,int)
56. int read()
57. {
58. if(head==null) return -1;
59. if(rp == null || rp.code == -1) return -1;
60. int c=rp.code;
61. rp=rp.next;
62. return c;
63. }//end: int read()
65. void reset()
66. {
67. if(head != null) rp=head;
68. }//end: void reset()
70. int length()
71. {
72. reset();
73. int i=0;
74. while(read() != (-1)) i++;
75. reset();
76. return i;
77. }//end: void length()
79. }//end: class S\_Itemset
80. @SuppressWarnings("serial")
81. public class apriori extends JFrame
82. {
83. static class Rule
84. {
85. float conf;
86. int antecedent;
87. int consequent;
88. int sup\_a;
89. int sup\_c;
90. int sup\_ac;
91. float liftRatio;
92. Rule next;
93. }//end: class Rule
95. private static String dset;
96. private static Rule r=null;
97. private static String[] header;
98. private static int trans=0;
99. private static int items;
100. private static float minsup;
101. private static float minconf;
102. private static byte k=1;
104. private static Pineapple.Slice next(Pineapple.Slice sl,char mode)
105. {
106. int v=(mode=='C')? (k-1):(k);
107. while(sl.next != null)
108. {
109. sl=sl.next;
110. if(sl.data.k == v) return sl;
111. if(sl.data.k == k) break;
112. }
113. return null;
114. }//end: Pineapple.Slice next(Pineapple.Slice,char)
116. private static int max(int code)
117. {
118. int c;
119. do
120. {
121. c=code%items;
122. code/=items;
123. }while(code > 0);
124. return c;
125. }//end: int max(int)
127. private static boolean candidate(Pineapple mem)
128. {
129. k++;
130. Pineapple.Slice sl=mem.linker;
131. while(sl != null && sl.data.k != k-1)
132. sl=sl.next;
133. if(sl == null) return false;
135. Pineapple.Slice nsl;
136. S\_Itemset sobj=new S\_Itemset();
137. int max,c,code;
138. boolean flag=false;
140. do
141. {
142. max=max(sl.data.code);
143. nsl=sl;
144. while(true)
145. {
146. nsl=next(nsl,'C');
147. if(nsl == null) break;
148. else
149. {
150. sobj.separate(nsl.data.code,items);
151. while((c=sobj.read()) != (-1))
152. {
153. if(c > max)
154. {
155. code=(sl.data.code)+(c\*(int)Math.pow(items,k-1));
156. if(mem.search(code) == null)
157. {
158. mem.insert(code,k,0);
159. flag=true;
160. }
161. }
162. }
163. }
164. }
165. }while((sl=next(sl,'C')) != null);
166. return flag;
167. }//end: boolean candidate(Pineapple)
169. private static void prune(Pineapple mem)
170. {
171. S\_Itemset sobj1=new S\_Itemset();
172. S\_Itemset sobj2=new S\_Itemset();
173. int pow,i,j,c1,c2,code;
174. Pineapple.Slice sl=mem.linker;
176. while(sl.data.k != k)
177. sl=sl.next;
178. do
179. {
180. sobj1.separate(sl.data.code,items);
181. pow=(int)Math.pow(2,k);
182. for(i=0;i<pow;i++) // Check all subsets of k-1 for this sl
183. {
184. sobj2.separate(i,2);
185. j=0;
186. while((c2=sobj2.read()) != (-1))
187. {
188. if(c2 == 1) j++;
189. }
190. if(j == k-1)
191. {
192. j=0;
193. code=0;
194. sobj1.reset();
195. sobj2.reset();
196. while(true)
197. {
198. c1=sobj1.read();
199. c2=sobj2.read();
200. if(c2 == 1)
201. {
202. code+=c1\*Math.pow(items,j++);
203. if(j == k-1) break;
204. }
205. }
206. if(mem.fetch(code).k == 0)
207. {
208. sl.data.k=0;
209. break;
210. }
211. }
212. }
213. }while((sl=sl.next) != null);
214. }//end: void prune(Pineapple)
216. private static void supportK1(Pineapple mem)
217. {
218. Pineapple.Slice sl=mem.linker;
220. do
221. {
222. if(sl.data.support < minsup)
223. {
224. sl.data.k=0;
225. }
226. }while((sl=sl.next) != null);
227. }//end: void supportK1(Pineapple)
229. private static void support(Pineapple mem)
230. {
231. Pineapple.Slice fsl=mem.linker;
232. while(fsl != null && fsl.data.k != k)
233. fsl=fsl.next;
234. if(fsl == null) return;
235. Pineapple.Slice sl;
236. String line;
237. S\_Itemset sobj=new S\_Itemset();
238. int c;
239. boolean flag;
241. try
242. {
243. BufferedReader dataset=new BufferedReader(new FileReader(dset));
244. line=dataset.readLine();
245. while(line != null)
246. {
247. sl=fsl;
248. while(sl != null)
249. {
250. flag=false;
251. sobj.separate(sl.data.code,items);
252. while((c=sobj.read()) != (-1))
253. {
254. if(line.charAt(c\*2) == '0')
255. {
256. flag=true;
257. break;
258. }
259. }
260. if(!flag) sl.data.support++;
261. sl=next(sl,'S');
262. }
263. line=dataset.readLine();
264. }
265. dataset.close();
266. }catch(IOException e){System.out.println(e.getMessage());System.exit(1);}
268. sl=fsl;
269. while(sl != null)
270. {
271. if(sl.data.support < minsup)
272. {
273. sl.data.k=0;
274. }
275. sl=next(sl,'S');
276. }
277. }//end: void support(Pineapple)
279. private static void print(Pineapple mem)
280. {
281. Pineapple.Slice sl=mem.linker;
282. S\_Itemset sobj=new S\_Itemset();
283. String s;
284. int c;
285. boolean flag=false;
286. while(sl != null)
287. {
288. if(sl.data.k == k)
289. {
290. if(!flag) { System.out.println("F"+k+"\\\t"+k+"-itemsets\n"); flag=true; }
291. sobj.separate(sl.data.code,items);
292. s="";
293. while((c=sobj.read()) != (-1))
294. {
295. s+=(char)(c+97)+" ";
296. }
297. System.out.println("< "+s+"> code: "+sl.data.code+" k: "+sl.data.k+" support: "+sl.data.support);
298. }
299. if(sl.data.k == k+1) break;
300. sl=sl.next;
301. }
302. if(flag) System.out.println();
303. }//end: void print(Pineapple)
305. private static void getRules(Pineapple mem,int para1,int para2)
306. {
307. Pineapple.Slice sl=mem.linker;
308. S\_Itemset sobj1=new S\_Itemset();
309. S\_Itemset sobj2=new S\_Itemset();
310. S\_Itemset sobj=new S\_Itemset();
311. int pow,i,c1,c2,sideL,powL,sideR,powR;
312. float conf;
313. boolean flag=false;
314. Rule p;
315. String s;
317. while((sl=sl.next) != null)
318. {
319. if(sl.data.k == 1) continue;
321. sobj1.separate(sl.data.code,items);
322. pow=(int)Math.pow(2,sl.data.k);
323. for(i=0;i<pow;i++)
324. {
325. if(i==0 || i==(pow-1)) continue;
326. sobj2.separate(i,2);
327. sideL=powL=sideR=powR=0;
328. sobj1.reset();
329. while((c1=sobj1.read()) != (-1))
330. {
331. c2=sobj2.read();
332. if(c2 != 1) sideL+=c1\*Math.pow(items,powL++);
333. else sideR+=c1\*Math.pow(items,powR++);
334. }
336. conf=(float)mem.fetch(sl.data.code).support/(float)mem.fetch(sideL).support;
337. if(conf >= minconf)
338. {
339. flag=true;
340. p=new Rule();
341. p.conf=conf;
342. p.antecedent=sideL;
343. p.consequent=sideR;
344. p.sup\_a=mem.fetch(sideL).support;
345. p.sup\_c=mem.fetch(sideR).support;
346. p.sup\_ac=mem.fetch(sl.data.code).support;
347. p.liftRatio=conf/((float)mem.fetch(sideR).support/trans);
348. if(r == null)
349. p.next=null;
350. else
351. p.next=r;
352. r=p;
354. s="";
355. sobj.separate(sideL,items);
356. while((c1=sobj.read()) != (-1))
357. {
358. s+=(char)(c1+97)+" ";
359. }
360. s+="--> ";
361. sobj.separate(sideR,items);
362. while((c1=sobj.read()) != (-1))
363. {
364. s+=(char)(c1+97)+" ";
365. }
366. System.out.println("\t"+s+"\t\tconf= "+conf);
367. }
368. }
369. }
370. if(flag) System.out.println("\nNo more rules can be generated.");
371. else System.out.println("No association rules.");
372. System.out.println("--------------------------------------------------\n");
374. sortRules();
375. int[] colsize={5,7,28,28,10,10,12,10};
376. TabledString report=new TabledString(colsize,3);
377. report.addSpace();
378. report.addData(1,"Rule#");
379. report.addData(2,"Conf. %");
380. report.addData(3,"Antecedent(a)");
381. report.addData(4,"Consequent(c)");
382. report.addData(5,"Support(a)");
383. report.addData(6,"Support(c)");
384. report.addData(7,"Support(aUc)");
385. report.addData(8,"Lift Ratio");
386. report.addLineOf('-'); report.addSpace();
387. DecimalFormat df = new DecimalFormat("#.##");
388. if(r == null) report.addString("No association rules.");
389. p=r;
390. i=0;
391. while(p != null)
392. {
393. i++;
394. if(p != r) { report.addRow(); report.addSpace(); }
395. report.addData(1,String.valueOf(i));
396. report.addData(2,df.format(p.conf\*100));
397. sobj.separate(p.antecedent,items);
398. pow=sobj.length(); s="";
399. while((c1=sobj.read()) != (-1))
400. {
401. s+=header[c1];
402. if(--pow != 0)
403. s+=", ";
404. }
405. report.addData(3,s+" =>");
406. sobj.separate(p.consequent,items);
407. pow=sobj.length(); s="";
408. while((c1=sobj.read()) != (-1))
409. {
410. s+=header[c1];
411. if(--pow != 0)
412. s+=", ";
413. }
414. report.addData(4,s);
415. report.addData(5,String.valueOf(p.sup\_a));
416. report.addData(6,String.valueOf(p.sup\_c));
417. report.addData(7,String.valueOf(p.sup\_ac));
418. report.addData(8,df.format(p.liftRatio));
419. p=p.next;
420. }
421. report.addLineOf('-');
422. s="min support: "+df.format(minsup)+" = ("+para1+"%)\n";
423. s+="min confidence: "+df.format(minconf)+" = ("+para2+"%)\n";
424. s+="dataset file: "+dset+"\n";
425. s+="transactions: "+trans+"\n";
426. s+="items: "+items+"\n\n";
427. s+=report.getString();
428. s+="Click anywhere and press Ctrl+A then Ctrl+C, and paste the results on a file.";
429. JOptionPane.showMessageDialog(null,new JTextArea(s),"Association Rules :",1);
431. }//end: void getRules(Pineapple,int,int)
433. private static void readHeader()
434. {
435. String filename="";
436. for(int i=0;i<dset.length();i++)
437. {
438. if(dset.charAt(i) == '.') break;
439. filename+=dset.charAt(i);
440. }
441. filename+=".dh";
443. header=new String[items];
444. String line;
445. int i;
446. try
447. {
448. i=0;
449. BufferedReader file=new BufferedReader(new FileReader(filename));
450. while((line=file.readLine()) != null)
451. {
452. header[i++]=line;
453. }
454. file.close();
455. }catch(IOException e){System.out.println(e.getMessage());System.exit(1);}
457. }//end: void readHeader()
459. private static void sortRules()
460. {
461. if(r == null) return;
462. int flag;
463. Rule prev,p1,p2;
464. while(true)
465. {
466. flag=0;
467. prev=null;
468. p1=r;
469. p2=r.next;
470. while(p2!=null)
471. {
472. if(p1.liftRatio < p2.liftRatio)
473. {
474. flag=1;
475. p1.next=p2.next;
476. p2.next=p1;
477. if(prev==null)
478. {
479. prev=p2;
480. r=p2;
481. }
482. else
483. {
484. prev.next=p2;
485. prev=p2;
486. }
487. p2=p1.next;
488. }
489. else
490. {
491. prev=p1;
492. p1=p2;
493. p2=p2.next;
494. }
495. }
496. if(flag==0) break;
497. }
499. }//end: void sortRules()
501. private static void getAssociationRules(int para1,int para2,String para3)
502. {
503. dset=para3;
504. int i,j;
505. String line;
506. Pineapple mem=null;
507. Pineapple.Itemset itemset;
508. System.out.println("\nsupport: "+para1+"%");
509. System.out.println("confidence: "+para2+"%");
510. System.out.println("dataset: "+dset);
512. try
513. {
514. BufferedReader dataset=new BufferedReader(new FileReader(dset));
515. line=dataset.readLine();
516. items=line.length()/2;
517. readHeader();
518. mem=new Pineapple(items);
519. do
520. {
521. trans++;
522. for(i=0,j=0;j<items;j++,i+=2)
523. {
524. if(line.charAt(i) == '1')
525. {
526. itemset=mem.search(j);
527. if(itemset == null)
528. mem.insert(j,k,1); else itemset.support++;
529. }
530. }
531. }while((line=dataset.readLine()) != null);
532. dataset.close();
533. }catch(IOException e){System.out.println(e.getMessage());System.exit(1);}
535. minsup=(float)trans/100\*para1;
536. supportK1(mem);
538. System.out.println("trans:"+trans);
539. System.out.println("Items: "+items);
540. System.out.println("--------------------------------------------------\n");
541. mem.sort();
542. print(mem);
544. while(candidate(mem))
545. {
546. if(k > 2) prune(mem);
547. support(mem);
548. print(mem);
549. }
550. System.out.println("No more itemsets can be generated.");
551. System.out.println("--------------------------------------------------\n");
553. minconf=(float)1/100\*para2;
554. getRules(mem,para1,para2);
556. }//end: void getAssociationRules(int,int,String)
558. public static void main(String[] args)
559. {
560. switch(args.length)
561. {
562. case 3:
563. getAssociationRules(Integer.valueOf(args[0]),Integer.valueOf(args[1]),args[2]);
564. System.exit(0);
566. default:
567. System.out.println("apriori [min\_support] [min\_confidence] [dataset\_file.ds]");
568. }
569. }//end: main()
571. }//end: class apriori

**Source code of Pineapple.java**

1. class Pineapple
2. {
3. class Itemset // 9 bytes
4. {
5. int code;
6. byte k;
7. int support;
8. Itemset(int value1,byte value2,int value3)
9. {
10. code=value1;
11. k=value2;
12. support=value3;
13. }
14. }//end: class Itemset
15. class Port // 5 bytes
16. {
17. byte number;
18. Slice path;
19. Port next;
20. Port(byte number,Slice path)
21. {
22. this.number=number;
23. this.path=path;
24. }
25. }//end: class Port
26. class Slice // 6 bytes
27. {
28. Itemset data=null;
29. Port ports=null;
30. Slice next=null;
31. }//end class Slice
32. int csys;
33. int count=0;
34. Slice crust[];
35. Slice linker=null;
36. Slice tail=null;
37. Pineapple(int csys)
38. {
39. crust= new Slice[csys];
40. this.csys=csys;
41. for(int i=0;i<csys;i++)
42. crust[i]=null;
43. }//end: Pineapple(int)
44. void insert(int value1,byte value2,int value3)
45. {
46. int val=value1;
47. Slice p2s; Port p2p;
48. int way=val%csys; val/=csys;
49. if(crust[way]==null)
50. {
51. p2s=new Slice();
52. crust[way]=p2s;
53. }
54. else
55. {
56. p2s=crust[way];
57. }
58. while(val>0)
59. {
60. way=val%csys; val/=csys;
61. p2p=p2s.ports;
62. while(p2p!=null && p2p.number!=way)
63. {
64. p2p=p2p.next;
65. }
66. if(p2p==null)
67. {
68. p2p=new Port((byte)way,new Slice());
69. p2p.next=p2s.ports;
70. p2s.ports=p2p;
71. }
72. p2s=p2p.path;
73. }
74. p2s.data=new Itemset(value1,value2,value3);
76. if(tail!=null)
77. tail.next=p2s;
78. tail=p2s;
79. if(linker==null)
80. linker=p2s;
82. count++;
83. }//end: void insert(int,byte,int)
84. Itemset search(int val)
85. {
86. Slice p2s; Port p2p;
87. int way=val%csys; val/=csys;
88. if(crust[way]==null)
89. return null;
90. else
91. p2s=crust[way];
92. while(val>0)
93. {
94. way=val%csys; val/=csys;
95. p2p=p2s.ports;
96. while(p2p!=null && p2p.number!=way)
97. {
98. p2p=p2p.next;
99. }
100. if(p2p==null)
101. return null;
102. else
103. p2s=p2p.path;
104. }
105. return p2s.data;
106. }//end: Itemset search(int)
107. Itemset fetch(int val)
108. {
109. Slice p2s; Port p2p;
110. int way=val%csys; val/=csys;
111. p2s=crust[way];
112. while(val>0)
113. {
114. way=val%csys; val/=csys;
115. p2p=p2s.ports;
116. while(p2p.number!=way)
117. {
118. p2p=p2p.next;
119. }
120. p2s=p2p.path;
121. }
122. return p2s.data;
123. }//end: Itemset fetch(int)
124. void sort()
125. {
126. int flag;
127. Slice prev,p1,p2;
128. while(true)
129. {
130. flag=0;
131. prev=null;
132. p1=linker;
133. p2=linker.next;
134. while(p2!=null)
135. {
136. if(p1.data.code>p2.data.code)
137. {
138. flag=1;
139. p1.next=p2.next;
140. p2.next=p1;
141. if(prev==null)
142. {
143. prev=p2;
144. linker=p2;
145. }
146. else
147. {
148. prev.next=p2;
149. prev=p2;
150. }
151. p2=p1.next;
152. }
153. else
154. {
155. prev=p1;
156. p1=p2;
157. p2=p2.next;
158. }
159. }
160. if(flag==0) break;
161. }
163. p1=linker;
164. while(p1.next!=null)
165. {
166. p1=p1.next;
167. }
168. tail=p1;
170. }//end: void sort()
172. }//end: class Pineapple

**Source code of TabledString.java**

1. public class TabledString
2. {
3. private int[] array;
4. private int space;
5. private String string;
7. TabledString(int[] array,int space)
8. {
9. this.array=array;
10. this.space=space;
11. string="";
13. }//end: TabledString(int[],int)
15. public void addLineOf(char c)
16. {
17. int i,j;
18. string+="\n";
19. for(j=1;j <= space;j++) string+=c;
20. for(i=0;i < array.length;i++)
21. {
22. for(j=1;j <= array[i];j++)
23. string+=c;
24. for(j=1;j <= space;j++) string+=c;
25. }
26. string+="\n";
28. }//end: void addLineOf(char)
30. public void addRow()
31. {
32. string+="\n";
34. }//end: void addRow()
36. public void addSpace()
37. {
38. for(int i=1;i <= space;i++)
39. if(i == space/2)
40. string+="|";
41. else
42. string+=" ";
44. }//end: void addSpace()
46. public void addData(int colnum,String s)
47. {
48. int i=0;
49. while(i < array[colnum-1])
50. {
51. if(i < s.length())
52. string+=s.charAt(i);
53. else
54. string+=" ";
55. i++;
56. }
57. addSpace();
59. }//end: void addData(int,String)
61. public void addString(String s)
62. {
63. string+=s;
65. }//end: void addString(String)
67. public String getString()
68. {
69. return string;
71. }//end: String getString()
73. }//end: class TabledString