1. class AssociationRule {
2. constructor() {
3. this.X = new Itemset();
4. this.Y = new Itemset();
5. this.Support = 0.0;
6. this.Confidence = 0.0;
7. }
8. toString() {
9. return this.X.toStringNoSupport() + ' => ' + this.Y.toStringNoSupport() +
10. ' (Support: ' + this.Support.toFixed(2) + '%, ' +
11. ' Confidence: ' + this.Confidence.toFixed(2) + '%)';
12. }
13. }
14. class AprioriMining {
15. static doApriori(db, supportThreshold) {
16. let I = db.getUniqueItems();
17. let L = new ItemsetCollection(); // Resultant large itemsets
18. let Li = new ItemsetCollection(); // Large itemset in each iteration
19. let Ci = new ItemsetCollection(); // Pruned itemset in each iteration
20. // First iteration (1-item itemsets)
21. for (var i = 0; i < I.length; i += 1) {
22. Ci.push(Itemset.from([I[i]]));
23. }
24. // Next iterations
25. let k = 2;
26. while (Ci.length != 0) {
27. // Set Li from Ci (pruning)
28. Li.clear();
29. for (var index in Ci) {
30. let itemset = Ci[index];
31. itemset.Support = db.findSupport(itemset);
32. if (itemset.Support >= supportThreshold) {
33. Li.push(itemset);
34. L.push(itemset);
35. }
36. }
37. // Set Ci for next iteration (find supersets of Li)
38. Ci.clear();
39. let subsets = Bit.findSubsets(Li.getUniqueItems(), k); // Get k-item subsets
40. subsets.forEach(set => Ci.push(set));
41. k += 1;
42. }
43. return L;
44. }
45. static mine(db, L, confidenceThreshold) {
46. let allRules = [];
48. for (var i in L) {
49. let itemset = L[i];
50. let subsets = Bit.findSubsets(itemset, 0); // Get all subsets
51. for (var j in subsets) {
52. let subset = subsets[j];
53. let confidence = (db.findSupport(itemset) / db.findSupport(subset)) \* 100.0;
54. if (confidence >= confidenceThreshold) {
55. let rule = new AssociationRule();
56. subset.forEach(i => rule.X.push(i));
57. itemset.removeItemset(subset).forEach(i => rule.Y.push(i));
58. rule.Support = db.findSupport(itemset);
59. rule.Confidence = confidence;
60. if (rule.X.length > 0 && rule.Y.length > 0) {
61. allRules.push(rule);
62. }
63. }
64. }
65. }
66. return allRules;
67. }
68. class Itemset extends Array {
69. constructor() {
70. super();
71. this.Support = 0.0;
72. }
74. includesItemset(itemset) {
75. for (var i = 0; i < itemset.length; i += 1) {
76. var item = itemset[i];
77. if (!this.includes(item)) {
78. return false;
79. }
80. }
81. return true;
82. }
83. removeItemset(itemset) {
84. var removed = new Itemset();
85. for (var i = 0; i < this.length; i += 1) {
86. var item = this[i];
87. if (!itemset.includes(item)) {
88. removed.push(item);
89. }
90. }
91. return removed;
92. }
93. toStringNoSupport() {
94. return '{' + this.join(', ') + '}';
95. }
96. toString() {
97. return '{' + this.join(', ') + '} (support: ' + this.Support + '%)';
98. }
99. }
100. }