TDT 4300 Data mining - Assignment 2

Arve Nygård, Tobias Linkjendal

26.02.2014

Project description

Code up and test three different apriori algorithms, and a rule-generation algorithm.

Frequent item set generation

Brute-force method: No pruning until all candidates have been generated. Cross join F_{k-1} with F_1.

$F_{k-1} \times F_1$: Same as brute force, check for support, and prune infrequent candidates before generating next level.

$F_{k-1} \times F_{k-1}$: Cross join F_{k-1} with itself to allow for even more aggressive pruning.

Rule generation

The key here was recursion. The method to be implemented was handed rule candidates of the form `ABC=>D`, for every combination of `A`,`B`,`C` and `D`. Our job was to recurse to make sure we calculated the rules of the form `AB=>CD`, and `A=>BCD` as well. This was done by moving the last element of the LHS item set onto the RHS item set before having the method call itself.

The assignment was a bit unclear on how to handle min-confidence. We ran with a hard-coded min-conf set to `0`, and a confidence check determining wether to keep recursing or not. This was done both to keep our results in line with the (presumed) expected results, as well as to show that we were exploiting the anti-monotone property of the arrows position for rules (i.e. we are doing “pruning” here as well).

We were disappointed that the big data set did not have different results even if you failed to do the recursion properly. Ideally it would be fun if the large data set was designed so that we had more levels in play as well.

Scalability

Brute force: The brute-force algorithm definitely scaled the worst. Every combination of item sets are generated for each level. O(nˆ2)-complexity regardless of the actual data.

$F_{k-1} \times F_1$: Significant improvement in running time over brute force. Allowed us to skip generation of a huge amount of candidates, by virtue of pruning the infrequent item sets before we generated the next set.

$F_{k-1} \times F_{k-1}$: Slight improvement in running time over $F_{k-1} \times F_1$. We suspect the data set did not allow for this algorithm to really run circles around the previous one. Allowed us to skip even some more generation of candidates, by not having to cross join with the original items (some of which turned out to be infrequent when combined with other items.)

Output - Small data set

Frequent itemset geeration: Brute-force method

Level	Generated Candidates	Kept frequent itemsets
1	6	4
2	15	4
3	20	1
4	15	0
5	6	0
6	1	0

Frequent itemset geeration: $F_{k-1} \times F_1$

Level	Generated Candidates	Kept frequent itemsets
1	6	4
2	6	4
3	4	1
4	1	0

Frequent itemset geeration: $F_{k-1} \times F_{k-1}$

Level	Generated Candidates	Kept frequent itemsets
1	6	4
2	6	4
3	3	1
4	0	0

Generated rules

Rule	Confidence	Support
[diapers] => [beer]	0.6	0.5
[beer] => [diapers]	1.0	0.5
[milk] => [bread]	0.7999999999999999	0.6666666666666666
[bread] => [milk]	0.7999999999999999	0.6666666666666666
[diapers] => [bread]	0.7999999999999999	0.6666666666666666
[bread] => [diapers]	0.7999999999999999	0.6666666666666666
[milk] => [diapers]	0.7999999999999999	0.6666666666666666
[diapers] => [milk]	0.7999999999999999	0.6666666666666666
[diapers, milk] => [bread]	0.75	0.5
[diapers] => [bread, milk]	0.6	0.5
[bread, milk] => [diapers]	0.75	0.5
[bread] => [diapers, milk]	0.6	0.5
[bread, diapers] => [milk]	0.75	0.5
[bread] => [diapers, milk]	0.6	0.5

Table 4: Rules for small data set

Output - Big data set

Brute-force method

Unfeasible. Did not finish within 30 minutes on a desktop machine. No data gathered.

$F_{k-1} \times F_1$

Level	Generated Candidates	Kept frequent itemsets
1	122	17
2	136	15
3	47	0

$F_{k-1} \times F_{k-1}$

Level	Generated Candidates	Kept frequent itemsets
1	122	17
2	136	15
3	30	0

Generated rules

Rule	Confidence	Support
[fruit] => [baking needs]	0.6414584740040514	0.4105445116681072
[baking needs] => [fruit]	0.6797853309481217	0.4105445116681072
[vegetables] => [baking needs]	0.6582235731171902	0.4211322385479689
[baking needs] => [vegetables]	0.6973166368515206	0.4211322385479689
[milk-cream] => [baking needs]	0.6488601565158216	0.412057044079516
[baking needs] => [milk-cream]	0.6822898032200359	0.412057044079516
[bread and cake] => [baking needs]	0.6579579579579579	0.47342264477095936
[baking needs] => [bread and cake]	0.7838998211091234	0.47342264477095936
[bread and cake] => [biscuits]	0.6255255255255255	0.4500864304235091
[biscuits] => [bread and cake]	0.7996161228406911	0.4500864304235091
[fruit] => [bread and cake]	0.7849426063470628	0.5023768366464996
[bread and cake] => [fruit]	0.6981981981981982	0.5023768366464996
[frozen foods] => [bread and cake]	0.7835848362164151	0.4600259291270527
[bread and cake] => [frozen foods]	0.6393393393393393	0.4600259291270527
[vegetables] => [bread and cake]	0.7760891590678825	0.496542783059637
[bread and cake] => [vegetables]	0.69009009009009	0.496542783059637
[milk-cream] => [bread and cake]	0.7951684246342293	0.5049697493517719
[bread and cake] => [milk-cream]	0.7018018018018019	0.5049697493517719
[juice-sat-cord-ms] => [bread and cake]	0.758830694275274	0.40384615384615385
[bread and cake] => [juice-sat-cord-ms]	0.5612612612612612	0.40384615384615385
[fruit] => [frozen foods]	0.6282916948008103	0.40211754537597233
[frozen foods] => [fruit]	0.6849466323150533	0.40211754537597233
[vegetables] => [frozen foods]	0.6355960824045931	0.4066551426101988
[frozen foods] => [vegetables]	0.6926757453073242	0.4066551426101988
[vegetables] => [fruit]	0.745356298547788	0.4768798617113224
[fruit] => [vegetables]	0.7451046590141797	0.4768798617113224
[milk-cream] => [fruit]	0.6934331405239877	0.4403630077787381
[fruit] => [milk-cream]	0.6880486158001351	0.4403630077787381
[vegetables] => [milk-cream]	0.6838905775075987	0.43755401901469315
[milk-cream] => [vegetables]	0.6890098673018032	0.43755401901469315

Table 7: Rules for big data set