# Association analysis

**Give definitions of monotone, antimonotone, and convertible monotone and convertible antimonotone constraints**

A constrain C is a function that returns true or false whether the itemset satisfies C or not

* Monotone constraint: If C(A)=true then C(B)=true where B is any itemset such that
* Antimonotone constraint: If C(A)=true then C(B)=true where B is any itemset such that
* Convertible monotone: If there exists an ordering R such that C(A)=true where A respects R, then C(B)=true for any itemset B that respects R and where A is a suffix of B
* Convertible antimonotone: If there exists an ordering R such that C(A)=true where A respects R, then C(B)=true for any itemset B that respects R and B is a suffix of A

A constraint that is both convertible monotone and antimonotone is strongly convertible.

**Give examples of different kinds of constraints**

Monotone constraints:

* where S is a set of prices (non-negative) and v is some value

Antimonotone constraints:

* where S is a set of prices (non-negative) and v is some value

**Give an example of a convertible monotone constraint that is not monotone.**

with respect to decreasing price order

**Give an example of a convertible antimonotone constraint that is not antimonotone.**

with respect to increasing price order

**Explain the Apriori property is**

Every subset of a frequent itemset is also frequent. Every superset of an infrequent itemset is also infrequent. The Apriori property speeds up the process when considered in association algorithms. It an itemsets is frequent, then all of its subsets will also be frequent, so we don’t have to check for that. Also, if an itemset is infrequent, we don’t have to check if all supersets are infrequent since we know that they are and can instead prune the infrequent itemsets.

**Describe the FP growth algorithm**

FP growth generates frequent patterns without generating candidates and therefore saves time and space. It only needs two scans of the database.

1. Scan the database to find all frequent 1-itemsets.
2. Sort the frequent 1-itemsets in each transaction in support descending order. All items in the database are sorted after the order in this transaction list. Then it outputs the frequent 1-itemsets and the infrequent ones are removed
3. An FP-tree is created. The FP-tree starts with an empty root node { }. Then the database is scanned again and the items are added to the tree in the support descending order decided before. A branch is created for each transaction and a count is made. To save space in the tree, we start with the most frequent item in the transaction. If the items in a transaction share the same prefix as a branch already in the tree they are added to the count is incremented.
4. After creating the FP-tree, it is mined to find the frequent itemsets instead of mining the original database. Start from each frequent 1-itemset and create its X-conditional database, that contains all prefix paths to X in the FP-tree. Mining is preformed recursively on the tree, so when we have found all frequent patterns with X as a prefix , we backtrack and restart the process for some other frequent 1-itemset. This is done until all conditional databases have been mined, when only the root is left in the FP-tree. Then we have found all frequent itemsets.

**What is the main advantage of the FP growth algorithm over the Apriori algorithm?**

FP growth doesn’t perform candidate generation as Apriori does, and therefore saves time and space. Since the algorithm mines the FP-tree, it only needs two scans of the database.

**Explain how to incorporate a monotone constraint into the FP growth algorithm**

When X in a X-conditional database fulfills the constraint, we don’t have to check this condition anymore for the X-conditional databases. The condition is not checked anymore for itemsets with X as a suffix since the condition is monotone and all supersets will also fulfill this constraint. Only minsup have to be checked.

**Explain how you incorporate an antimonotone constraint into the FP growth algorithm**

If X in a conditional-database does not fulfill the constraint, then all supersets with X as a suffix can be pruned since they will not fulfill it either. Then we don’t have to do any conditional databases for X as a suffix.

**Describe the Apriori algorithm**

1. Scan the database and find all frequent 1-itemsets. This is done by generating candidate items of size 1, . If itemsets in is at least minsup, they are considered frequent and go to the large 1-itemsets while the infrequent ones are pruned.
2. Then, it performs a self-join of to generate , candidate 2-itemsets by combining items with the same prefix. In this case, prefix is empty so all combinations are generated. The database is scanned and the support is counted. The infrequent 2-itemsets with support lower than minsup are pruned and the frequent ones go to , which contains all large 2-itemsets. Then a self-join is performed for by combining those items with the same prefix to get , and so forth.
3. This is performed until no more k-itemsets can be generated. Finally the algorithm outputs the frequent itemsets

**Explain how you incorporate an antimonotone constraint into the apriori algorithm**

Only large k-itemsets that satisfy the additional antimonotone constraint are outputted. The candidate k-itemsets that doesn’t satisfy the constraint are pruned (just as the ones not satisfying minsup, which is another antimonotone constrain). Only those itemsets in that satisfy both minsup and the additional constraint goes to .

**Explain how to incorporate a monotone constraint into the Apriori algorithm**

It is trickier to incorporate a monotone constrain into the algorithm. If itemsets don’t satisfy a monotone constraint, we can’t prune them, since the constrain might be true for some superset.

If an itemset fulfills the monotone constraint, we don’t have to check if the supersets of that itemset fulfills it, since we already know it does because of the Apriori property. They go directly to the output (given that they satisfy minsup).

**Sketch a proof of the correctness of the Apriori algorithm**

We prove Apriori is correct by induction

1. Trivial case: . Apriori is correct for (can be seen by line 1 in pseudocode)
2. Induction hypothesis: Apriori is correct up to ,
3. Proof that by induction:  
   * Assume contradiction that some itemset but ,   
     i.e. that is not contained in
   * Induction hypothesis says that
   * To get , we self-join (apriori-gen)  
     Since is a large itemset () it is not pruned from and therefore   
     i.e.   
     This is a contradiction to our assumption and therefore, Apriori is correct for k.