DATA MINING

**Frequent Item Dataset**

**Market-Basket Data** – A general many-to-many mapping between two kinds of things, where the one (the baskets) is a set of the other (the items). The technology focuses on common events, not rare events.

Example:

|  |  |
| --- | --- |
| TID | Items |
| 1 | Bread, Milk |
| 2 | Bread, Diaper, Beer Eggs |
| 3 | Milk, Diaper, Beer, Coke |
| 4 | Bread, Milk, Diaper, Beer |
| 5 | Bread, Milk, Diaper, Coke |

Examples of frequent itemsets s(1) >= 3

{Bread} = 4

{Milk} = 4

{Diaper} = 4

{Beer} = 3

{Diaper, Beer} = 3

{Milk, Bread} = 3

Items = products; basket = sets of products

**Definition**:

**Itemset**: a collection of one or more items

**Support**: count (frequency of occurrence of an itemset), fraction (fraction of transaction that contain an itemset)

**Frequent itemset**: an itemset whose support is greater than or equal to a minsup threshold

**Mining frequent itemsets task**

Input: a set of transactions T, over a set of items I

Output: all itemsets with items in I having support >= minsup threshold

Problem parameters:

N = |T|: number of transactions

d = |I|: number of distinct items

w: max width of transactions

number of possible itemsets? M= 2d

**A Naïve algorithm**

**Brute-force approach**, each itemset is a candidate:

Consider each itemset is a lattice, and count the support of each candidate by scanning he data

Time complexity – O(NMw), Space complexity – O(M)

**Or**:

Scan the data, and for each transaction generate all possible itemsets. Keep a count for each itemset in the data

Time complexity – O(N2w), Space complexity – O(M)

**Computation Model**

Typically, data is kept in flat files rather than in database system.

* Stored on disk
* Stored basket-by-basket
* Expand baskets into pairs

**Computational Model 2**

The true cost of mining disk-resident data is usually the number of disk I/O’s

In practice, association-rule algorithms read the data in passes – all baskets read in turn

Thus, we measure the cost by the number of passes an algorithm takes

**Main-memory bottleneck**

For many frequent-itemsets algorithms, main memory is a critical resource

* As we read baskets, we need to count something, for example, occurrences of pairs.
* The number of different things we can count is limited by main memory.
* Swapping counts in/out is a disaster.

**The Apriori principle**

* If an itemset is frequent, then all of its subsets must also be frequent
* If an itemset is not frequent, then all of its supersets cannot be frequent
* The support of an itemset never exceeds the support of its subsets
* This is known as the anti-monotone property

**The Apriori algorithm**

* Ck = candidate itemsets of size k
* Lk = frequent itemsets of size k
* k = 1, C1 = all items
* While Ck not empty
  + Scan the database to find which itemsets in Ck are frequent and put them into Lk
  + Use Lk to generate a collection of candidate itemsets Ck+1 of size k+1
  + k++

**Candidate generation**

* Basic principle:
  + An itemset of size k+1 is a candidate to be frequent only if all its subsets of size k are known to be frequent
* Main idea:
  + Construct a candidate of size k+1 by combining frequent itemsets of size k
    - If k = 1, take all the pairs of frequent items
    - If k > 1, join pairs of itemsets that differ by just one item
    - For each generated candidate itemset, ensure that all subsets of size k are frequent

Association Rules

Given a set of transactions, find rules that will predict the occurrence of an item based on the occurrences of other items in the transaction.

Definition:

* An implication expression of the form X -> Y where X and Y are itemsets. Example: {Milk, Diaper} -> {Beer}
* Support (s):
  + Fraction of transactions that contain both X and Y
  + The probability P (X, Y) that X and Y occur together
* Confidence (c):
  + Measures how often items in Y appear in transactions that contain X
  + The conditional probability P (Y|X) that Y occurs given that X has occurred.

Tasks:

* Input: a set of transactions T, over a set of items I
* Output: all rules with items in I having
  + Support >= minsup threshold
  + Confidence >= minconf threshold

Two-step approach:

* Frequent Itemset Generation
  + Generate all itemsets whose support >= minsup
* Rule Generation
  + Generate high confidence rules from each frequent itemset, where each rule is a partitioning of a frequent itemset into Left-Hand-Side and Right-Hand-Side

Rule generation:

* We have all frequent itemsets, how do we get the rules?
  + For every frequent itemset S, we find rules of the form L -> S – L, where L S, that satisfy the minimum confidence requirements
  + Example: L = {A, B, C, D}
  + Candidate rules: A -> BCD, B -> ACD, C -> ABD, D -> ABC, AB -> CD, AC -> BD, AD -> BC, BC -> AD, BD -> AC, CD -> AB, ABC -> D, BCD -> A
* If |L| = k, then there are – 2 candidate association rules (ignoring L -> and -> L)
* How to efficiently generate rules from frequent itemsets?
  + In general, confidence does not have an anti-monotone property: c (ABC -> D) can be larger or smaller than c (AB -> D)
  + But confidence of rules generated from the same itemset has an anti-monotone property
  + Example: L = {A, B, C, D}: c (ABC -> D) >= c (AB -> CD) >= c (A -> BCD)
  + Confidence is an anti-monotone w.r.t number of items on the right-hand side of the rule

Rule generation for APriori Algorithm:

* Candidate rule is generated by merging two rules that share the same prefix in the right-hand side (RHS)
* Join (CD -> AB, BD -> AC) would produce the candidate rule D -> ABC
* Prune rule D -> ABC if its subset AD -> BC does not have high confidence
* Essentially, we are doing APriori on the RHS

Link Analysis Ranking

How to organize the web:

* Manually curated Web Directories
* Web Search
* Using the web graph
* Link analysis ranking

Definition:

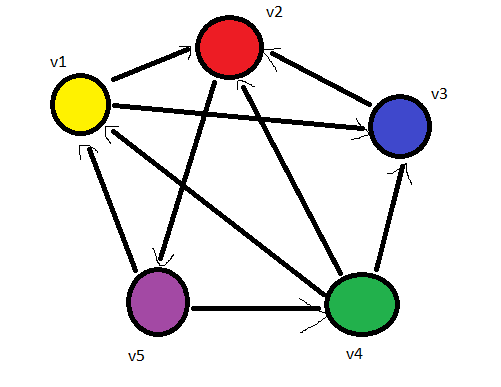
* Use the graph structure in order to determine the relative importance
* Intuition: an edge from node p to node q denotes endorsement
  + Node p endorses/recommends/confirms the authority/centrality/importance of node q

Rank by popularity:

* Rank pages according to the number of incoming edges
* How many edges link to you is not the only important factor, but how important are those edges is necessary to define importance.
* Good authorities are pointed by good authorities

PageRank

* Good authorities should be pointed by good authorities
  + The value of a node is the value of the nodes that point to it
* How do we implement that?
  + Assume that we have a unit of authority to distribute to all nodes
  + Node i gets a fraction of that authority weight
  + Each node distributes the authority value they have to their neighbors
  + The authority value of each node is the sum of the authority fractions it collects from its neighbors
* Example:



Random Walks on Graphs

* The algorithm defines a random walk on the graph
* Rando walk:
  + Start from the node chosen uniformly at random with probability
    - Pick one of the outgoing edges uniformly at random
    - Move to the destination of the edge
    - Repeat

Markov chains

* A Markov chain describes a discrete time stochastic process over a set of states
  + S = {according to a transition probability matrix P = {}
  + = probability of moving to state j when at state i
* Matrix P has the property that the entries of all rows sum to 1. A matrix with this property is called stochastic
* The stochastic process proceeds in steps and moves between the states:
  + State probability distribution: the vector = {} that stores the probability of being at state after t steps
* Memorylessness property: the next state of the chain depends only at the current state and not on the past of the process (first order MC)
  + Higher-order MSs are also possible
* We can compute the vector at step t using a vector-matrix multiplication

HITS Algorithm

* Authority is not necessarily transferred directly between authorities
* Pages have double identity
  + Hub identity
  + Authority identity
* Good hubs point to good authorities
* Good authorities are pointed by good hubs
* Two kinds of weights:
  + Hub weight
  + Authority weight
* The hub weight is the sum of the authority weights of the authorities pointed to by the hub
* The authority weight is the sum of the hub weights that point to this authority

HITS and eigenvectors

* The HITS algorithm is a power-method eigenvector computation
* In vector terms
  + and
  + and
  + Repeated iterations will converge to the eigenvectors
* The authority weight vector a is the eigenvector of
* The hub weight vector h is the eigenvector of
* The vectors a and h are the singular vectors of the matrix A