Mining Big Data

Frequent Itemsets

(Chapter 6)

Introduction

- Investigating Similar Items, we wanted to find items that have a large fraction of their baskets in common.
- For Frequent Itemsets, we are interested in the absolute number of baskets that contain a particular set of items.

Market-Basket Model

- The market-based model is used to describe a many-many relationship between two kinds of objects.
- We have items and baskets.
- Each basket consists of a set of items (an itemset)
- Usually, the number of items in a basket is small (much smaller than the total number of items).
- The number of baskets is usually very large (too big to fit in main memory).
- Data is represented as a file consisting of a sequence of baskets.
- In a distributed file system, baskets are objects of the file and each basket is of type “set of items”.

Frequent Itemsets

- A set of items that appears in many baskets is said to be “frequent”.
- We assume that there is a number s, called the support threshold.
- The support of an itemset I is the number of baskets for which I is a subset.
- We say that I is frequent, if its support is at least s.

Example

Given the following 8 baskets (each consisting of a set of items):

1. {Cat, and, dog, bites}
2. {Yahoo, news, claims, a, cat, mated, with, a, dog, and, produced, viable, offspring}
3. {Cat, killer, likely, is, a, big, dog}
4. {Professional, free, advice, on, dog, training, puppy, training}
5. {Cat, and, kitten, training, and, behavior}
6. {Dog, &, Cat, provides, dog, training, in, Eugene, Oregon}
7. {"Dog, and, cat", is, a, slang, term, used, by, police, officers, for, a, male–female, relationship}
8. {Shop, for, your, show, dog, grooming, and, pet, supplies}

Figure 6.1 in Rajaraman/Ullman

s=3: frequent singleton itemsets: {dog}, {cat}, {a}, {training}

Example

- Doubletons: A doubleton cannot be frequent unless both items in the set are frequent.
- There are 10 possible frequent doubletons:

	training	a	and	cat
dog	4, 6	2, 3, 7	1, 2, 8	1, 2, 3, 6, 7
cat	5, 6	2, 3, 7	1, 2, 5	
and	5	2, 7		
a	none			

Figure 6.2 in Rajaraman/Ullman

- For s=3: we get {dog, a}, {dog, and}, {dog, cat}, {cat, a}, {cat, and} as the frequent doubletons.

Applications

- Original application of the market-based model is the analysis of true market baskets.
- Supermarkets/chain stores record every market basket (shopping card).
- Items are the different products and baskets are set of items in a single market basket.
- A major chain might sell 100,000 different items and collect data about millions of market baskets.
- Finding frequent itemsets, the chain can learn what is commonly bought together.

Applications

- We will discover that many people buy milk and bread together (but these are popular items individually)
- We will also discover that many people buy hot dogs and mustard together (allows for clever marketing: lower price for hot dogs and raise price for mustard).
- Diapers and beer: One would hardly expect these items to be related. But, it has been discovered that people buying diapers are also likely to buy beer (explanation: If there is a baby at home, then you are unlikely to go drinking at a bar and you are more likely to bring beer home).

Other Examples

Related Concepts:

- Let items be words and let baskets be documents.
- A basket contains these items that are present in the document.
- We ignore stop words and look for sets of words that appear together in many documents.

Plagiarism:

- Items are documents and baskets are sentences.
- We look for pairs of items that appear together in several baskets (documents that share particular sentences).
- In practice, even one or two sentences in common is a good indicator for plagiarism.

Association Rules

- Information on frequent itemsets is often represented in form of association rules.
- The form of an association rule is I -> j, where I is a set of items and j is an item.
- Implication of the rule is that if all items in I appear in some basket then "j" is "likely" to appear as well.
- We formalize "likely" by defining the confidence of the rule I -> j.
- The confidence of I->j is given by the ratio of the support for I $\cup$ {j} to the support of I.

Example

1. {Cat, and, dog, bites}
2. {Yahoo, news, claims, a, cat, mated, with, a, dog, and, produced, viable, offspring}
3. {Cat, killer, likely, is, a, big, dog}
4. {Professional, free, advice, on, dog, training, puppy, training}
5. {Cat, and, kitten, training, and, behavior}
6. {Dog, &, Cat, provides, dog, training, in, Eugene, Oregon}
7. {"Dog, and, cat", is, a, slang, term, used, by, police, officers, for, a, male–female, relationship}
8. {Shop, for, your, show, dog, grooming, and, pet, supplies}

Figure 6.1 in Rajaraman/Ullman

Confidence for rule {cat, dog} -> {and} is 2/5.

Confidence for rule {cat} -> {kitten} is 1/6.

Confidence and Interest

- Confidence can only be useful if support for the left side is fairly large.
- The interest of an association rule I -> j is the difference between its confidence and the fraction of baskets that contain j.
- If I has no influence of j, we expect the interest to be 0.
- High interest means that I causes j whereas high negative interest means that I discourages the presence of j.

Examples

- Example for high interest: {diapers} -> beer
- Example for high negative interest {coke} -> pepsi and similarly {pepsi} -> coke

Association Rules with High Confidence

- For now, we assume that we have the frequent itemsets (support at least s).
- We are looking for the association rules I -> j that apply to a reasonable fraction of the baskets.
- The support of I should be reasonably high (let's say 1% of the baskets)
- Confidence of the rule should be high as well (let's say 50%).
- As a result $I \cup \{j\}$ will also have a fairly high support.

Association Rules with High Confidence

- Assume that we found all itemsets of support at least s and know their exact support.
- We want to find within them all association rules that have both high support and high confidence.
- Let J be a set of n items that is found to be frequent.
- There are only n possible association rules involving J, namely J \ {j} -> j for each j in J.
- If J is frequent then J \ {j} must be at least as frequent as it is a subset (support has already been computed).
- The confidence of J \ {j} -> j is given by the ratio of the supports of J \ {j} and J.

Number of Frequent Itemsets

- There should not be too many frequent itemsets and thus not too many candidates for high support.
- If we give a store manager a million association rules, he can not even read them or even act on them.
- It is normal to adjust the support threshold such that we don’t get too many frequent itemsets.

Representation of Market-Basket Data

- We assume that market-based data is stored in a file basket-by-basket
- Data can be stored in a distributed file system and baskets are the objects of the files.
- Example of a file:

$$\underbrace{\{23, 456, 1001\}}_{\text{First basket}}\underbrace{\{3,18,92,145\}}_{\text{Second basket}}\{....$$

Representation of Market-Basket Data

- We assume that files of baskets don’t fit into main memory (cost for reading from disk).
- Once a disk block full of baskets is in the main memory, we can generate all subsets of size k.
- A assume that the average size of baskets is small and that the time to generate all subsets is less than the time to transfer baskets into main memory.
- If n is the number of items in a basket and k is the size of the subsets, then generating all subsets of size k takes time $n^k/(k!)$ (not feasible for large k and n)

Algorithm Analysis

In practice:

- Often we need to find small frequent itemsets, i.e. k<=2.
- If k is large, then it's often possible to eliminate many items that don't take part in a frequent itemset (n drops if k increases).

Algorithm analysis:

- In practice the work, require to examine each basket is often proportional to the size of the file.
- We can measure the running time of a frequent-itemset algorithm by the number of times each disk block is read.
- Our algorithms pass through the basket file(s) and the time is given by the size of the basket file times the number of passes through this file.

Use of Main Memory

- All frequent-itemsets algorithms require us to make many different counts as we pass through the data.
- Example: We have to count the number of times each pair of items occurs in baskets.
  - For n elements about $n^2/2$ space required.
  - If integers take 4 bytes and we got 2 gigabyte then n<33,000 has to hold such that each count fits into main memory.
- If we don’t have enough main memory to store each of the counts, we have to load a page from disk (significant slow down)

Counts

- We want to store the $\binom{n}{2}$ counts such that it's easy to find the count for a pair {i,j}.
- It's more space efficient to represent element by consecutive integers 1, …, n (for n items).
- If they are not represented in that way, we need a hash table that translates items into numbers 1, …, n.
- Each time we see an item, we hash it to its number.
- If we see a new item, we assign the next available number to it.

Triangular-Matrix Method

- Assume that we have coded items as integers (4 bytes).
- We use a one-dimensional triangular array to store the $\binom{n}{2}$ counters.
- We store in a[k] the count for the pair {i,j} with 1≤i<j≤n where
$$k = (i-1)\left(n - \frac{i}{2}\right) + j - i$$
- This implies that counters for pairs are stored in lexicographically order, i.e {1,2},{1,3}, …, {1,n}, {2,3}, {2,4}, …., {n-1,n}
- Total storage $2n^2$ bytes.

Triples Method

- Depending on the fraction of possible pairs, the Triples Method might be more appropriate.
- Here we store counts as triples [i,j,c], which means that the count of {i,j} is c.
- We can use a hash table with {i,j} as keys and c as its value.
- We can quickly test whether there is a triple for {i,j} and if so, find it.
- Advantage: Does not require to store anything for a pairs whose count is 0.
- Disadvantage: We need 3 integers instead of 1 if count is >0.

Monotonicity of Itemsets

Observation: If I is a frequent itemset then every subset of I is a frequent itemset.

- Let $J \subseteq I$, then the count for J is at least as high as the count for I.
- Given a support threshold s, we say that an itemset is maximal if no superset is frequent.
- We only need to know all maximal subsets (as we know that subsets of them are also frequent)

Tyranny of Counting Pairs

- Number of items is usually not so large that we cannot count all singleton sets in the main memory.
- In order for frequent-itemsets analysis to make sense, the result has to be a small number of sets (otherwise we cannot even read them all).
- In practice the threshold s is set high enough such that there are not too many frequent sets.
- Monotonicity tells us that we expect to find more frequent pair than triples, more triples than quadruples, and so on.

The A-Priori Algorithm

- We consider frequent pairs only now.
- We can count all pairs in the main memory, then doing the counting is easy.
- We each basket, we use a double loop to generate all pairs. If we generate a pair, we increase its counter by one.
- At the end, we report all pairs of counter at least s.

The A-Priori Algorithm

- The previous approach doesn’t work if there are too many pairs to count them in the main memory.
- The A-Priori Algorithm reduces the number of pairs that must be counted at the expense of performing two passes over the data (rather than one pass).

First Pass

- The first pass creates two tables.
- The first table translates item names into integers from 1 to n (as explained before).
- The second table is an array of counts.
  - Ith element counts the occurrences of item number i.
  - Reading the baskets, we translate names to integers and increase the count of a newly found item.

Between the Passes

- After the first pass, we determine which singletons are frequent.
- We can ignore all items that are not frequent as singletons as pairs involving these items can’t be frequent.
- Assume that there are m frequent singletons.
- We create a new numbering 1,…,m for these items.

Second Pass

- During the second pass, we count all the pairs consisting of two frequent items.
- Space required is $2m^2$ (instead of $2n^2$) if we use the triangular matrix.

Second pass:

- For each basket, look in the frequent-items table to see which items are frequent.
- In a double loop, generate all frequent pairs.
- For each frequent pair, add one to its counter.

Main Memory during passes

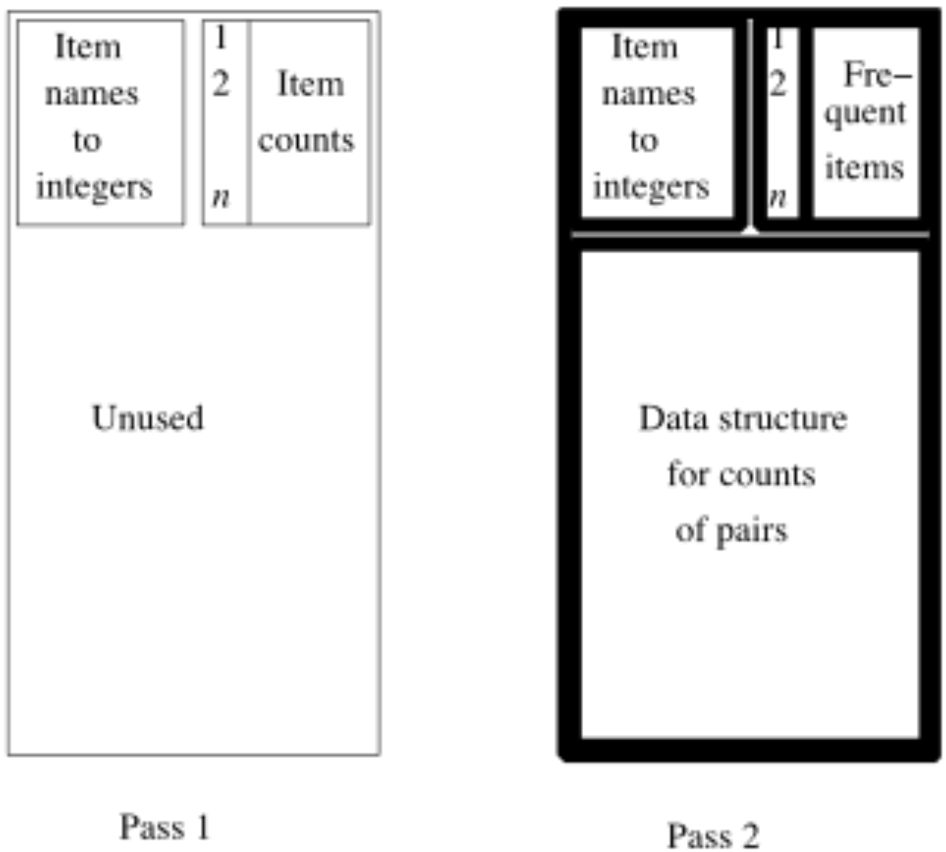

Figure 6.3 in Rajaraman/Ullman

A-Priori for All Frequent Itemsets

- We can generalize the approach to find all frequent itemsets.
- We do one pass for each set-size k.

For each k, there are two sets of itemsets:

- C_k: set of candidate itemsets of size k (itemsets that we must count to figure out whether they are frequent)
- L_k: set of truly frequent itemsets of size k

Schematic View

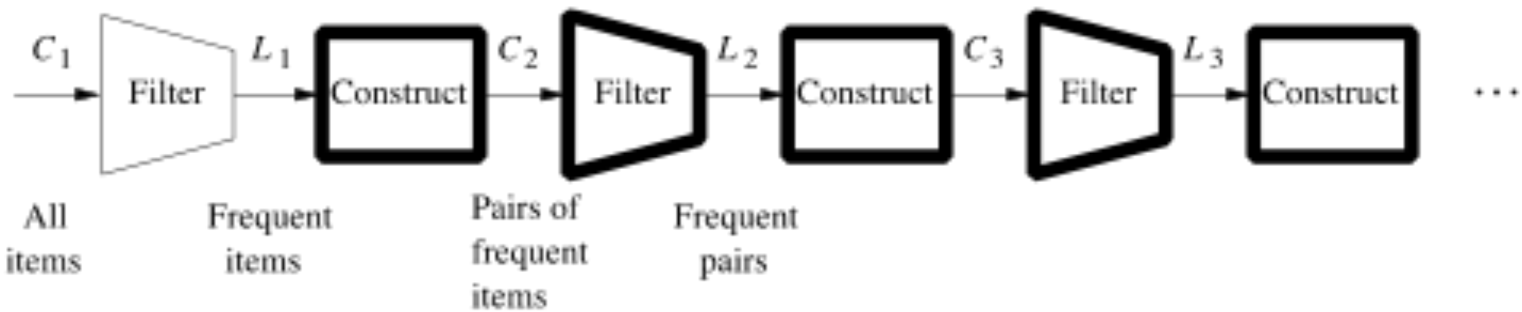

Figure 6.4 in Rajaraman/Ullman

Constructing frequent itemsets by constructing candidate sets and filtering the truly frequent items for increasing values of k.

A-Priori for All Frequent Itemsets

- Start with C1 (all singleton sets) and construct L1 (frequent singleton sets)
- C2: set of size 2 where both items are in L1 (we don't construct it explicitly)
- L2: count all candidate pairs and determine which appear at least s times.
- C3: set of size 3 where the three subsets of size 2 are in L2.
- L3: Pass through the basket and for any triple of C3 found, increase its counter

General:

- C_k: all itemsets of size k where every k-1 of which is an itemset in L_{k-1}.
- L_k: pass through the baskets and count all (and only) the itemsets of size k that are in C_k. Those itemsets that have a counter of at least s are in L_k.

Example 6.8

- Suppose basket consists of items 1 through 10.
- L1: Of these 1 through 5 have been found to be frequent items
- L2: Pairs {1,2}, {2,3}, {3,4}, {4,5} have been found to be frequent pairs.
- We eliminate the non-frequent items, leaving only 1 through 5.
- 1 and 5 appear only in one frequent pair and cannot contribute to frequent triples. So, we have to consider only subsets of {2,3,4}. (only one triplet).
- {2,4} is a subset of {2,3,4} that is not in L2. Hence {2,3,4} is not in C3 (not a candidate)

Handling Larger Datasets in Main Memory

- A-priori Algorithm is fine as long as the step with the greatest main memory requirement (typically counting the elements of C2) has enough memory.
- Several algorithms have been proposed to cut down the size of C2.

Algorithm of Park, Chen, and Yu (PCY)

- There may be much unused main memory in the first pass.
- We often don't need more than 10% of the main memory.
- The PCY Algorithm uses this space for an array of integers that generalizes the idea of a Bloom filter.

Schema for Memory of PCY Algorithm

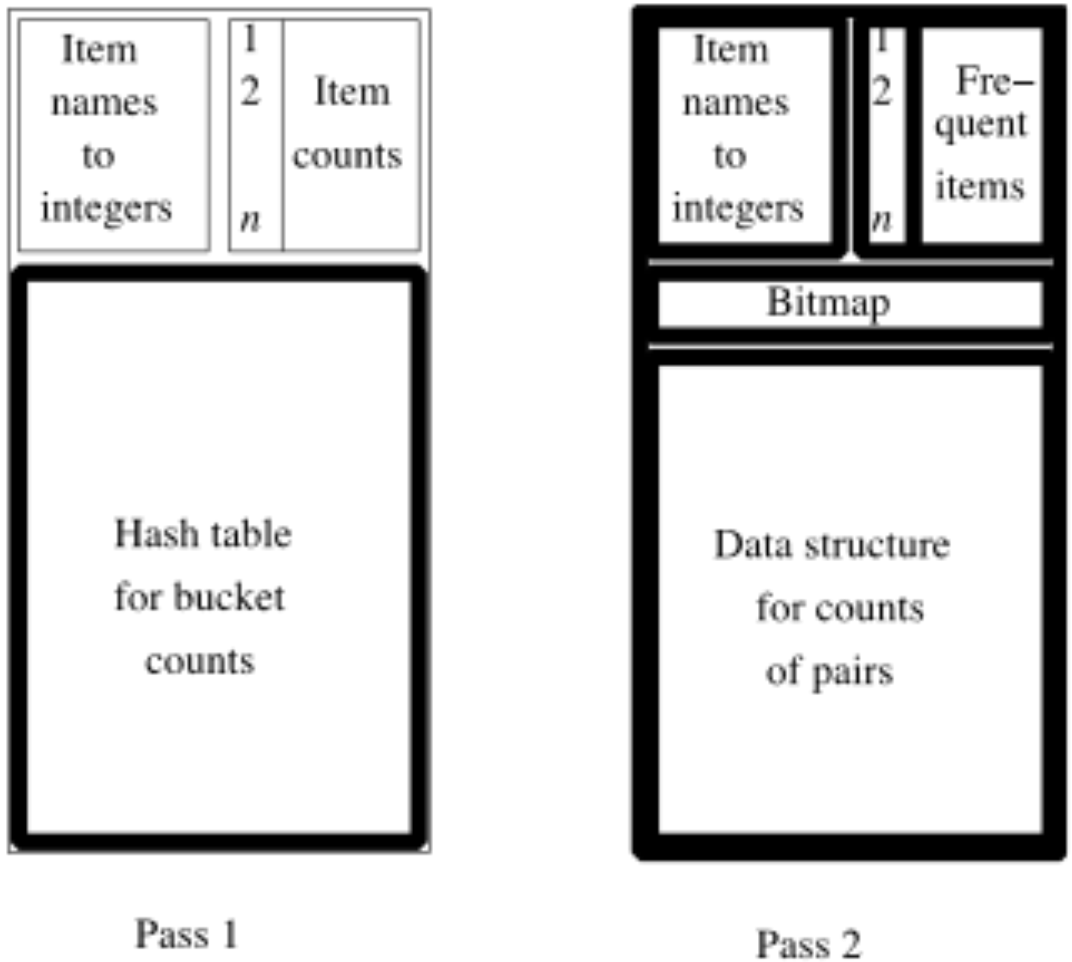

Figure 6.5 in Rajaraman/Ullman

PCY Algorithm

- Pass 1

```
FOR (each basket) {
    FOR (each item in the basket)
      add 1 to item's count;
    FOR (each pair of items) {
      hash the pair to a bucket;
      add 1 to the count for that bucket
    }
}
```

Observations

- A bucket that a frequent pair hashes to is surely frequent

- The count for a bucket is less than the support s

Between passes

- Replace the buckets by a bit-vector (bitmap)
- 4 byte int is replaced by bits, so it is 1/32

PCY Algorithm

- Think of this array as a hash table whose buckets hold integers (not set of keys).
- Pairs of items are hashed to buckets in the table.
- During the first pass, we hash each item and add one to the counter.
- Furthermore, we generate all pairs using a double loop.
- We hash each pair and add 1 to the bucket where the pair hashes.
- At the end of the first pass, each bucket has a count and if the count is at least s, we call is a frequent bucket.
- Note that we can ignore all pairs whose bucket count is less than s (infrequent buckets) as the pair can not be frequent.

Set of candidate pairs C2 contains {i,j} if

1. i and j are frequent items and
2. {i,j} hashes to a frequent bucket

Condition 2. is distinguishes PCY from A-Priori.

PCY Algorithm

- Between the passes the hash table is summarized as a bitmap (1 if bucket if frequent, 0 otherwise).
- 32 bits are replaced by 1.
- PCY can handle some datasets in the main memory where the A-Priori Algorithm would run out of main memory.
- The pairs that PCY avoids would be placed randomly in the triangular matrix.
- We use the triple method in PCY and only gain if PCY avoids counting at least 2/3 of the frequent pairs.
- Finding frequent pairs for larger sets is essentially the same as for the A-Priori Algorithm.

Example

- Assume we have 1 gigabyte available for the hash table in the first pass.
- Suppose that the data file has a billion baskets, each with 10 items.
- A bucket is an integer (4 bytes) and we can maintain a quarter of a billion buckets.
- The number of pairs in all baskets is $4.5*10^{10}$. Same for the number of counts.
- Average count is 180.
- If we set s=180, we might expect few buckets to be infrequent.
- If we set s=1000, we expect a great majority of buckets to be infrequent. Greatest possible number of frequent buckets is 45 million (out of 250 million).