**CSE5243: Class notes on November 20, 2014**

Agenda:

1. Partition I/O
2. Distributed version of (1)
3. Sampling for AR/FPM

Out of core: Data too large to fit in memory

Parallel: Distributed computing

What if dataset is too large to fit in memory?

Requirement: We want the exact same result as we’d get if we had infinite memory

1. Divide data into chunks

Assuming algorithm like Apriori/ECLAT as a blackbox, our input is dataset D and support (δ), we want the patterns in the dataset.

Each chunk’s size is less than the available memory. We feed each chunk and support threshold (δ into the black box algorithm and we get a set of patterns. The support threshold is a value between 0 and 1 (expressed as a fraction of number of transactions).

Let, us partition the dataset into d partitions, D1, D2, … Dd such that



Let, running the black box algorithm independently on each partition, we get frequent patterns P1, P2…Pd, respectively.

What is the relationship between the true frequent patterns P and (P1, P2…Pd) as we get from d chunks (D1, D2, … Dd)?

Suggestions:

1. P is a subset of the union of Pis: 

This is true because to be globally frequent, a pattern must be frequent at least in one partition. But this way we also get some false positives. These are patterns that are frequent in some partition(s) but not frequent with respect to the entire dataset. So in order to get the exact set P, we need a second pass over the data to get the actual counts of patterns in  to get true frequent items. The reason we need the 2nd pass is to remove false positives.

All this is based on the idea that suggestion 1 is true. How do we prove it?

How to prove this?

**Proof by contradiction:**

Let,  but not in . That is,  (1)

Let support threshold be δ = (assume 0.2 w.l.g)

This implies support of x, δx in each Di, i=1…d (data chunks) is less than 0.2|Di| (from (1)). Here |Di| refers to the size of chunk Di, i.e., the number of transaction in chunk Di.

So, δx in Di < 0.2|Di|, for i=1…d.

Summing over all i, we get,



So, we see that support of pattern x is less that 0.2 ((D) < 0.2|D|) which is a contradiction because we assumed x is frequent in D! So (D) must be greater than or equal to 0.2|D|. So x must belong to .

This is a powerful algorithm that can fit nicely into the map-reduce framework as well. In pass1 map, we get frequent items from each chunk. In reduce, combine them into a global union. In pass 2, calculate the count of the items in the entire dataset to eliminate false positives. A final reduction step is needed to get the true frequent patterns.

One possible optimization of the above technique is to take intersection of the frequent patterns in each chunk (). The benefit is, for patterns , we don’t need to count these items in pass 2 because items that are frequent in each chunk are guaranteed to be globally frequent. That is,  so we can exclude them from the counting step.

**Sampling based scaling:**

With dataset D, support threshold δ, running the Black box algorithm we get frequent patterns P.

If we take a sample d from D (), run black box algorithm and get frequent patterns for the sample, Pd.

How do we get P from Pd?

**Positive Border and Negative Border of lattice:**

Assume lattice for items A, B, C, D:

ABCL

ABC ACD ABD BCD

AB AC AD BC BD CD

A B C D

**Transaction dataset:**

**Transaction# Items**

**------------------------------------------**

T1 A B C

} Sample

T2 B C D

T3 A C D

-------------------

T4 A C D

T5 A C D

T6 A B C

Sample T1-T3. Support δ = 0.5 (50%)

Evaluate A B C D (all freq in sample)

Evaluate AB’ AC AD’ BC BD’ CD (items with ‘ are not freq in sample)

None of the bigger candidates are possible. These are the border itemsets. The frequent ones are **positive border** and infrequent ones are **negative border**.

Now look at the entire dataset:

AB is not frequent.

AC is frequent.

AD is frequent (**Switch from negative border to positive set**) **<- needs special case.**

BC is frequent

BD is not frequent

CD is frequent

With AD becoming freq, we now are able to generate a bigger candidate **ACD** since AC was already frequent.

Now counting ACD we find it to be frequent.

The benefit of sample is that we need to examine far less candidates. The critical part is the **switch** between negative and positive borders. It reduces I/O by working with a small sample that fits into memory. We still need one pass over the entire dataset but we need to count for fewer candidates.

**Positive border**: **Maximal** elements wrt the sample (all freq. are in positive set, but only maximal items are part of positive border)

**Negative border**: Items that are deemed infrequent in sample.

When an item frequent in sample becomes infrequent in dataset, it goes to negative border.

These can be extended to sequence mining as well.

**Proof Exercises for pattern constraints:**

Practice Session: proving that some constraint is **monotone/anti-monotone** etc.

Example:

Sum(S.price) <= V is **anti-monotone**

**Proof**:

Let 

Let, S violates Constraint C but S’ does not

Since S violates constraint, Sum(S.price) > V.

If price is non negative (>= 0) and S’ is a superset of S (), Sum(S’.price) must be greater than V. I.e., Sum(S’.price) > V which is a contradiction! Because we assumed S’ does not violate C so we assumed Sum(S’.price) <= V. Hence, Sum(S.price) <= V is anti-monotone.

So we prove the constraint to be anti-monotone.