CompSci 516
Data Intensive Computing Systems

Lecture 25
Data Mining
and
Mining Association Rules

Instructor: Sudeepa Roy

Announcements

- HW5 due tomorrow 04/20 (Wednesday), 11:55 pm

- Additional office hour – Sudeepa - Thursdays – 3 – 4 pm – D325 (until the exam)

- Review session a few days before the final

Reading Material

Optional Reading:

1. [RG]: Chapter 26

2. "Fast Algorithms for Mining Association Rules"
Agrawal and Srikant, VLDB 1994

19,496 citations on Google Scholar (as of April, 2016 - ~800 increase in eight months)!
One of the most cited papers in CS

- Acknowledgement:

The following slides have been prepared adapting the slides provided by the authors of [RG] and using several presentations of this paper available on the internet (esp. by Ofer Pasternak and Brian Chase)

Data Mining - 1

- Find interesting trends or patterns in large datasets
  - to guide decisions about future activities
  - ideally, with minimal user input
  - the identified patterns should give a data analyst useful and unexpected insights
  - can be explored further with other decision support tools (like data cube)

Data Mining - 2

- Related to
  - exploratory data analysis (Statistics)
  - Knowledge Discovery (KD)
  - Machine Learning
- Scalability is important and a new criterion
  - w.r.t. main memory and CPU
- Additional criteria
  - Noisy and incomplete data (Lecture 24)
  - Iterative process (improve reliability and reduce missing patterns with user inputs)

OLAP vs. Data Mining

- Both analyze and explore data
  - SQL queries (relational algebra)
  - OLAP (multidimensional model)
  - Data mining (most abstract analysis operations)
- Data mining has more flexibility
  - assume complex high level "queries"
  - few parameters are user-definable
  - specialized algorithms are needed

Four Main Steps in KD and DM (KDD)

- Data Selection
  - Identify target subset of data and attributes of interest
- Data Cleaning
  - Remove noise and outliers, unify units, create new fields, use denormalization if needed
- Data Mining
  - extract interesting patterns
- Evaluation
  - present the patterns to the end users in a suitable form, e.g. through visualization

Several DM/KD (Research) Problems

- Discovery of causal rules
- Learning of logical definitions
- Fitting of functions to data
- Clustering
- Classification
- Inferring functional dependencies from data
- Finding “usefulness” or “interestingness” of a rule
  - See the citations in the Agarwal-Srikant paper
  - Some discussed in [RG] Chapter 27

More: Iceberg Queries

```
SELECT P.custid, P.item, SUM(P.qty)
FROM Purchases P
GROUP BY P.custid, P.item
HAVING SUM(P.qty) > 5
```

- Output is much smaller than the original relation or full query answer
- Computing the full answer and post-processing may not be a good idea
- Try to find efficient algorithms with full “recall” and high “precision”

ref. ”Computing Iceberg Queries Efficiently”
Fang et al.
VLDB 1998

Our Focus in this Lecture

- Frequent Itemset Counting
- Mining Association Rules
  - using frequent itemsets
  - Both from the Agarwal-Srikant paper

- Many of the "rule-discovery systems" can use the association rule mining ideas

Mining Association Rules

- Retailers can collect and store massive amounts of sales data
  - transaction date and list of items
- Association rules:
  - e.g. 98% customers who purchase "tires" and "auto accessories" also get "automotive services" done
  - Customers who buy mustard and ketchup also buy burgers
  - Goal: find these rules from just transactional data (transaction id + list of items)

Applications

- Can be used for
  - marketing program and strategies
  - cross-marketing
  - catalog design
  - add-on sales
  - store layout
  - customer segmentation

Notations

- Items $I = \{i_1, i_2, \ldots, i_m\}$
- D : a set of transactions
- Each transaction $T \subseteq I$
  - has an identifier TID
- Association Rule
  - $X \rightarrow Y$
  - $X, Y \subset I$
  - $X \cap Y = \varnothing$

Confidence and Support

- Association rule X→Y
- Confidence c = |Tr. with X and Y|/|Tr. with |X|
  - c% of transactions in D that contain X also contain Y
- Support s = |Tr. with X and Y| / |all Tr.|
  - s% of transactions in D contain X and Y.

Support Example

TID	Cereal	Beer	Bread	Bananas	Milk
1	X		X		X
2	X		X	X	X
3		X			X
4	X			X	
5			X		X
6	X				X
7		X		X	
8			X		

- Support(Cereal)
  - 4/8 = .5
- Support(Cereal → Milk)
  - 3/8 = .375

Confidence Example

TID	Cereal	Beer	Bread	Bananas	Milk
1	X		X		X
2	X		X	X	X
3		X			X
4	X			X	
5			X		X
6	X				X
7		X		X	
8			X		

- Confidence(Cereal → Milk)
  - 3/4 = .75
- Confidence(Bananas → Bread)
  - 1/3 = .33333…

X → Y is not a Functional Dependency

For functional dependencies

- F.D. = two tuples with the same value of of X must have the same value of Y
  - X → Y => XZ → Y (concatenation)
  - X → Y, Y → Z => X → Z (transitivity)

For association rules

- X → A does not mean XY→A
  - May not have the minimum support
  - Assume one transaction {AX}
- X → A and A → Z do not mean X → Z
  - May not have the minimum confidence
  - Assume two transactions {XA}, {AZ}

Problem Definition

- Input
  - a set of transactions D
    - Can be in any form – a file, relational table, etc.
  - min support (minsup)
  - min confidence (minconf)

- Goal: generate all association rules that have
  - support >= minsup and
  - confidence >= minconf

Decomposition into two subproblems

- 1. Apriori and AprioriTID:
  - for finding “large” itemsets with support >= minsup
  - all other itemsets are “small”
- 2. Then use another algorithm to find rules X → Y such that
  - Both itemsets X ∪ Y and X are large
  - X → Y has confidence >= minconf
- Paper focuses on subproblem 1
  - if support is low, confidence may not say much
  - subproblem 2 in full version

Basic Ideas - 1

- Q. Which itemset can possibly have larger support: ABCD or AB
  - i.e. when one is a subset of the other?

- Ans: AB
  - any subset of a large itemset must be large
  - So if AB is small, no need to investigate ABC, ABCD etc.

Basic Ideas - 2

- Start with individual (singleton) items {A}, {B}, …
- In subsequent passes, extend the "large itemsets" of the previous pass as "seed"
- Generate new potentially large itemsets (candidate itemsets)
- Then count their actual support from the data
- At the end of the pass, determine which of the candidate itemsets are actually large
  - becomes seed for the next pass
- Continue until no new large itemsets are found

- Benefit: candidate itemsets are generated using the previous pass, without looking at the transactions in the database
  - Much smaller number of candidate itemsets are generated

Apriori vs. AprioriTID

- Both follow the basic ideas in the previous slides
- AprioriTID has the additional property that that the database is not used at all for counting the support of candidate itemsets after the first pass
  - An "encoding" of the itemsets used in the previous pass is employed
  - Size of the encoding becomes smaller in subsequent passes – saves reading efforts
- More later

Notations

- Assume the database is of the form <TID, i1, i2, …> where items are stored in lexicographic order
- TID = identifier of the transaction
- Also works when the database is “normalized”: each database record is <TID, item> pair

k-itemset	An itemset having k items.	
L_k	Set of large k-itemsets (those with minimum support). Each member of this set has two fields: i) itemset and ii) support count.	ACTUAL
C_k	Set of candidate k-itemsets (potentially large itemsets). Each member of this set has two fields: i) itemset and ii) support count.	POTENTIAL Used in both Apriori and AprioriTID
$\overline{C}_k$	Set of candidate k-itemsets when the TIDs of the generating transactions are kept associated with the candidates.	Used in AprioriTID

Algorithm Apriori

$L_1 = \{large\ 1\text{-}itemsets\}$

For $(k = 2;\ L_{k-1} \neq \phi;\ k{+}{+})$ do begin

  C_k = apriori-gen(L_{k-1});

  forall transactions $t \in D$ do begin

    C_t = subset(C_k, t)

    forall candidates $c \in C_t$ do

      $c.count{+}{+}$;

    end

  end

  $L_k = \{c \in C_k \mid c.count \geq minsup\}$

end

$Answer = \bigcup_k L_k$;

Count individual item occurrences

Generate new k-itemsets candidates

count = 0

Find the support of all the candidates

C_t = candidates contained in t

increment count

Take only those with support >= minsup

Apriori-Gen

- Takes as argument L_{k-1} (the set of all large k-1)-itemsets
- Returns a superset of the set of all large k-itemsets by augmenting L_{k-1}

Join step

$L_{k-1} \bowtie L_{k-1}$

insert into C_k

select $p.item_1, p.item_2, p.item_{k-1}, q.item_{k-1}$

from $L_{k-1}p, L_{k-1}q$

where $p.item_1 = q.item_1, \ldots, \quad p.item_{k-2} = q.item_{k-2}, \quad p.item_{k-1} < q.item_{k-1}$

p and q are two large

(k-1)-itemsets identical in all k-2 first items.

Join by adding the last item of q to p

Prune step

forall *itemsets* $c \in C_k$ do

forall *(k-1)-subsets s of c* do

if $(s \notin L_{k-1})$ then

delete c *from* C_k

Check all the subsets, remove all candidate with some "small" subset

Apriori-Gen Example - 1

Step 1: Join (k = 4)

Assume numbers 1-5 correspond to individual items

L_3

- {1,2,3}
- {1,2,4}

  {1,3,4}
- {1,3,5}
- {2,3,4}

C_4

- {1,2,3,4}

Apriori-Gen Example - 2

Step 1: Join (k = 4)

Assume numbers 1-5 correspond to individual items

L_3

- {1,2,3}
- {1,2,4}
  {1,3,4}
- {1,3,5}
- {2,3,4}

C_4

- {1,2,3,4}
- {1,3,4,5}

Apriori-Gen Example - 3

Step 2: Prune (k = 4)

- Remove itemsets that can't have the required support because there is a subset in it which doesn't have the level of support i.e. not in the previous pass (k-1)

L_3

- {1,2,3}
- {1,2,4}
  {1,3,4}
- {1,3,5}
- {2,3,4}

C_4

- {1,2,3,4}
- ~~{1,3,4,5}~~

No {1,4,5} exists in L_3
Rules out {1, 3, 4, 5}

Comparisons with previous algorithms (AIS, STEM)

L_{k-1} to C_k

- Read each transaction t
- Find itemsets p in L_{k-1} that are in t
- Extend p with large items in t and occur later in lexicographic order

L_3

- {1,2,3}
- {1,2,4}
- {1,3,4}
- {1,3,5}
- {2,3,4}

C_4

- {1,2,3,4}
- {1,2,3,5}
- {1,2,4,5}
- {1,3,4,5}
- {2,3,4,5}

t = {1, 2, 3, 4, 5}
all 1-5 large items (why?)

5 candidates compared to 2 (after pruning 1) in Apriori

Correctness of Apriori

Check yourself

join

insert into C_k

select $p.item_1, p.item_2, p.item_{k-1}, q.item_{k-1}$

from $L_{k-1} p, L_{k-1} q$

where $p.item_1 = q.item_1, \ldots, \quad p.item_{k-2} = q.item_{k-2}, \quad p.item_{k-1} < q.item_{k-1}$

Show that $C_k \supseteq L_k$

- Any subset of large itemset must also be large
- for each p in L_k, it has a subset q in L_{k-1}
- We are extending those subsets q in Join with another subset q' of p, which must also be large
  - equivalent to extending L_{k-1} with all items and removing those

    whose (k-1) subsets are not in L_{k-1}
- Prune is not deleting anything from L_k

prune

forall *itemsets* $c \in C_k$ do

forall *(k-1)-subsets s of c* do

if $(s \notin L_{k-1})$ then

delete *c from* C_k

Variations of Apriori

- Counting candidates of multiple sizes in one pass
- In the k-th pass
  - Not only update counts of C_k
  - update counts of candidates C'_{k+1}
  - $C'_{k+1} \supseteq C_{k+1}$ since it is generated from L_k
  - Can help when the cost of updating and keeping in memory $C'_{k+1} - C_{k+1}$ additional candidates is less than scanning the database

Problem with Apriori

- Every pass goes over the entire dataset
- Database of transactions is massive
  - Can be millions of transactions added an hour
- Scanning database is expensive
  - In later passes transactions are likely NOT to contain large itemsets
  - Don't need to check those transactions

$L_1 = \{large\ 1\text{-}itemsets\}$
For $(k = 2;\ L_{k-1} \neq \phi;\ k++)$ do begin
  $C_k = \text{apriori-gen}(L_{k-1})$;
  forall transactions $t \in D$ do begin
    $C_t = \text{subset}(C_k, t)$
    forall candidates $c \in C_t$ do
      $c.count++$;
    end
  end
  $L_k = \{c \in C_k | c.count \geq minsup\}$
end
$Answer = \bigcup_k L_k$;

AprioriTid

See the examples in the following slides and then see this slide for an overview

- Also uses Apriori-Gen
- But scans the database D only once.
- Builds a storage set C^*_K
  - “bar” in the paper instead of *
- Members of C^*_K are of the form < TID, $\{X_k\}$ >
  - each X_k is a potentially large k-itemset present in the transaction TID
  - For k=1, C^*_1 is the database D
  - items i as {i}
- If a transaction does not have a candidate k-itemset, C^*_K will not contain anything for that TID
- C^*_K may be smaller than #transactions, esp. for large values of k
  - For smaller values of k, it may be large

See the examples in the following slides and then come back to the algorithm

Algorithm AprioriTid

$L_1 = \{large\ 1\text{-}itemsets\}$

$\hat{C}_1 = database\ D;$

For $(k = 2;\ L_{k-1} \neq \phi;\ k{+}{+})$ do begin

$C_k = \text{apriori-gen}(L_{k-1});$

$\hat{C}_k = \phi;$

forall $entries\ t \in \hat{C}_{k-1}$ do begin

$C_t = \{c \in C_k | (c - c[k]) \in t.set-of-items$

$\wedge (c - c[k-1]) \in t.set-of-items\};$

forall candidates $c \in C_t$ do

$c.count{+}{+};$

if $(C_t \neq \varphi)\ then\ \hat{C}_k += < t.TID, C_t >;$

end

end

$L_k = \{c \in C_k | c.count \geq minsup\}$

end

$Answer = \bigcup_k L_k;$

Count item occurrences

The storage set is initialized with the database

Generate new k-itemsets candidates

Build a new storage set

Determine candidate itemsets which are containted in transaction TID

Find the support of all the candidates

Remove empty entries

Take only those with support over minsup

AprioriTid Example

Min support = 2

Database

TID	Items
100	1 3 4
200	2 3 5
300	1 2 3 5
400	2 5

$\overline{C}_1$

TID	Set-of-Itemsets
100	$\{ \{1\}, \{3\}, \{4\} \}$
200	$\{ \{2\}, \{3\}, \{5\} \}$
300	$\{ \{1\}, \{2\}, \{3\}, \{5\} \}$
400	$\{ \{2\}, \{5\} \}$

L_1

Itemset	Support
$\{1\}$	2
$\{2\}$	3
$\{3\}$	3
$\{5\}$	3

Min support = 2

AprioriTid Example

L_1

Itemset	Support
{1}	2
{2}	3
{3}	3
{5}	3

Apriori-gen →

k = 2

C_2

Itemset	Support
{1 2}	
{1 3}	
{1 5}	
{2 3}	
{2 5}	
{3 5}	

Now we need to compute the supports of C_2
without looking at the database D
from C^*_1

AprioriTid Example

Min support = 2

k = 2

C_2

Itemset	Support
{1 2}	1
{1 3}	
{1 5}	
{2 3}	
{2 5}	
{3 5}	

$\overline{C}_1$

TID	Set-of-Itemsets
100	{ {1}, {3}, {4} }
200	{ {2}, {3}, {5} }
300	{ {1}, {2}, {3}, {5} }
400	{ {2}, {5} }

C_{100} = {{1, 3}}
C_{200} = {{2, 3}, {2, 5}, {3, 5}}
C_{300} = {{1, 2}, {1, 3}, {1, 5}, {2, 3}, {2, 5}, {3, 5}}
C_{400} = {{2, 5}}

Only 300 has both {1} and {2}
Support = 1

forall entries $t \in \overline{C}_{k-1}$ **do begin**
  // determine candidate itemsets in C_k contained
  // in the transaction with identifier t.TID
  $C_t = \{c \in C_k \mid (c - c[k]) \in t.\text{set-of-itemsets} \wedge$
    $(c - c[k-1]) \in t.\text{set-of-itemsets}\};$
  forall candidates $c \in C_t$ **do**
    c.count++;
  if $(C_t \neq \emptyset)$ **then** $\overline{C}_k$ += $< t.\text{TID}, C_t >$;
end

Min support = 2

AprioriTid Example

k = 2

C_2

Itemset	Support
{1 2}	1
{1 3}	2
{1 5}	
{2 3}	
{2 5}	
{3 5}	

$\overline{C}_1$

TID	Set-of-Itemsets
100	{ {1}, {3}, {4} }
200	{ {2}, {3}, {5} }
300	{ {1}, {2}, {3}, {5} }
400	{ {2}, {5} }

C_{100} = {{1, 3}}
C_{200} = {{2, 3}, {2, 5}, {3, 5}}
C_{300} = {{1, 2}, {1, 3}, {1, 5}, {2, 3}, {2, 5}, {3, 5}
C_{400} = {{2, 5}}

forall entries $t \in \overline{C}_{k-1}$ **do begin**
  // determine candidate itemsets in C_k contained
  // in the transaction with identifier t.TID
  $C_t = \{c \in C_k \mid (c - c[k]) \in t.\text{set-of-itemsets} \wedge$
    $(c - c[k-1]) \in t.\text{set-of-itemsets}\}$;
  forall candidates $c \in C_t$ **do**
    c.count++;
  if $(C_t \neq \emptyset)$ **then** $\overline{C}_k \mathrel{+}= < t.\text{TID}, C_t >$;
end

Min support = 2

AprioriTid Example

k = 2

C_2

Itemset	Support
{1 2}	1
{1 3}	2
{1 5}	1
{2 3}	2
{2 5}	3
{3 5}	2

$\overline{C}_1$

TID	Set-of-Itemsets
100	{ {1}, {3}, {4} }
200	{ {2}, {3}, {5} }
300	{ {1}, {2}, {3}, {5} }
400	{ {2}, {5} }

C_{100} = {{1, 3}}
C_{200} = {{2, 3}, {2, 5}, {3, 5}}
C_{300} = {{1, 2}, {1, 3}, {1, 5}, {2, 3}, {2, 5}, {3, 5}}
C_{400} = {{2, 5}}

forall entries $t \in \overline{C}_{k-1}$ **do begin**
    // determine candidate itemsets in C_k contained
    // in the transaction with identifier t.TID
    $C_t = \{c \in C_k \mid (c - c[k]) \in t.\text{set-of-itemsets} \wedge$
        $(c - c[k-1]) \in t.\text{set-of-itemsets}\};$
    forall candidates $c \in C_t$ **do**
        c.count++;
    if $(C_t \neq \emptyset)$ **then** $\overline{C}_k$ += $< t.\text{TID}, C_t >$;
end

AprioriTid Example

Min support = 2

How C^*_2 looks

k = 2

C_2

Itemset	Support
{1 2}	1
{1 3}	2
{1 5}	1
{2 3}	2
{2 5}	3
{3 5}	2

$\overline{C}_2$

TID	Set-of-Itemsets
100	{ {1 3} }
200	{ {2 3}, {2 5}, {3 5} }
300	{ {1 2}, {1 3}, {1 5}, {2 3}, {2 5}, {3 5} }
400	{ {2 5} }

C_{100} = {{1, 3}}
C_{200} = {{2, 3}, {2, 5}, {3, 5}}
C_{300} = {{1, 2}, {1, 3}, {1, 5}, {2, 3}, {2, 5}, {3, 5}}
C_{400} = {{2, 5}}

forall entries $t \in \overline{C}_{k-1}$ **do begin**
// determine candidate itemsets in C_k contained
// in the transaction with identifier t.TID
$C_t = \{c \in C_k \mid (c - c[k]) \in t.\text{set-of-itemsets} \wedge (c - c[k-1]) \in t.\text{set-of-itemsets}\}$;
forall candidates $c \in C_t$ **do**
c.count++;
if $(C_t \neq \emptyset)$ **then** $\overline{C}_k$ += < t.TID, C_t >;
end

AprioriTid Example

Min support = 2

k = 2

C_2

Itemset	Support
{1 2}	1
{1 3}	2
{1 5}	1
{2 3}	2
{2 5}	3
{3 5}	2

L_2

Itemset	Support
{1 3}	2
{2 3}	2
{2 5}	3
{3 5}	2

How L_2 looks
(entries above threshold)

The supports are in place
Can compute L_2 from C_2

AprioriTid Example

Min support = 2

k = 3

L_2

Itemset	Support
{1 3}	2
{2 3}	2
{2 5}	3
{3 5}	2

Apriori-gen →

C_3

Itemset	Support
{2 3 5}	

Next step

Min support = 2

AprioriTid Example

C_3

Itemset	Support
{2 3 5}	

$\overline{C}_2$

TID	Set-of-Itemsets
100	{ {1 3} }
200	{ {2 3}, {2 5}, {3 5} }
300	{ {1 2}, {1 3}, {1 5}, {2 3}, {2 5}, {3 5} }
400	{ {2 5} }

Look for transactions containing {2, 3} and {2, 5}

Add <200, {2,3,5}> and <300, {2,3,5}> to C^*_3

forall entries $t \in \overline{C}_{k-1}$ **do begin**

// determine candidate itemsets in C_k contained

// in the transaction with identifier t.TID

$C_t = \{c \in C_k \mid (c - c[k]) \in t.\text{set-of-itemsets} \wedge (c - c[k-1]) \in t.\text{set-of-itemsets}\};$

forall candidates $c \in C_t$ **do**

$c.\text{count}{+}{+};$

if $(C_t \neq \emptyset)$ **then** $\overline{C}_k \mathrel{+}= < t.\text{TID}, C_t >;$

end

Min support = 2

AprioriTid Example

C_3

Itemset	Support
{2 3 5}	2

$\overline{C}_3$

TID	Set-of-Itemsets
200	{ {2 3 5} }
300	{ {2 3 5} }

L_3

Itemset	Support
{2 3 5}	2

C^*_3 has only two transactions (we started with 4)

L_3 has the largest itemset

C_4 is empty

Stop

Optional: read the correctness proof, buffer managements, data structure from the paper

Discovering Rules (from the full version of the paper)

Naïve algorithm:

- For every large itemset p
  - Find all non-empty subsets of p
  - For every subset q
    - Produce rule q $\rightarrow$ (p-q)
    - Accept if support(p) / support(q) >= minconf

Checking the subsets

- For efficiency, generate subsets using recursive DFS. If a subset q does not produce a rule, we do not need to check for subsets of q

Example

Given itemset : ABCD

If ABC → D does not have enough confidence

then AB → CD does not hold

Reason

- For efficiency, generate subsets using recursive DFS. If a subset q does not produce a rule, we do not need to check for subsets of q

For any subset q' of q:

Support(q') >= support(q)

confidence (q' → (p-q'))

= support(p) / support(q')

<= support(p) / support(q)

= confidence (q → (p-q))

optional slide

Simple Algorithm

l = p

a = q

forall *large itemsets* l_k, $k \geq 2$ *do*

$genrules(l_k, l_k)$

procedure genrules (l_k:*large k-itemset*, a_m: *large m-itemset*)

$A = \{(m-1)\text{-itemset } a_{m-1} \mid a_{m-1} \subset a_m\}$;

forall $a_{m-1} \in A$ do begin

$conf = support(l_k)/support(a_{m-1})$

if ($conf \geq minconf$) then begin

output *the rule* $a_{m-1} \Rightarrow (l_k - a_{m-1})$;

if $(m - 1 > 1)$ then

call $genrules(l_k, a_{m-1})$;

end

end

Check all the large itemsets

Check all the subsets

Check confidence of new rule

Output the rule

Continue the depth-first search over the subsets.

If not enough confidence, the DFS branch cuts here

Faster Algorithm

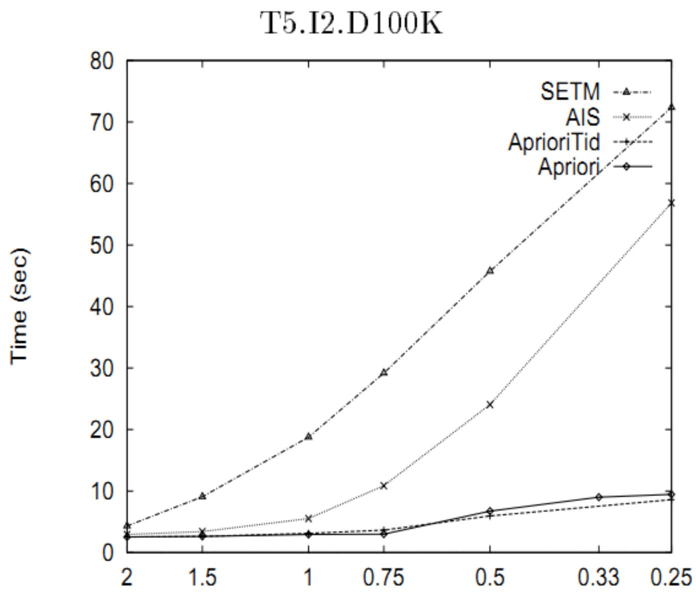
- If (p-q) → q holds than all the rules
(p-q') → q' must hold
  - where q' ⊆ q and is non-empty

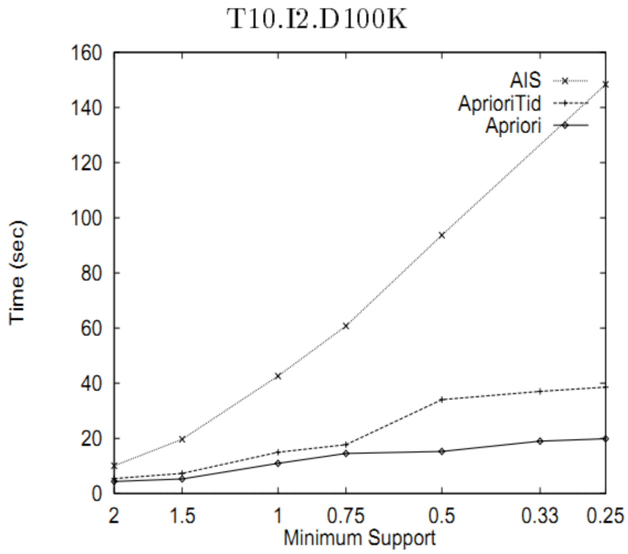
Example:
If AB → CD holds,
then so do ABC → D and ABD → C

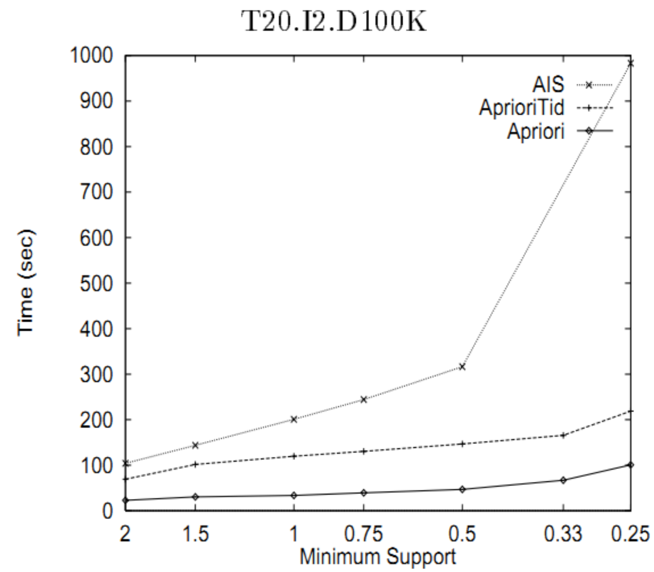
Idea
- Start with 1-item consequent and generate larger consequents
- If a consequent does not hold, do not look for bigger ones
- The candidate set will be a subset of the simple algorithm

Performance

- Support decreases => time increases
- AprioriTID is “almost” as good as Apriori, BUT Slower for larger problems
  - C^*_k does not fit in memory and increases with #transactions

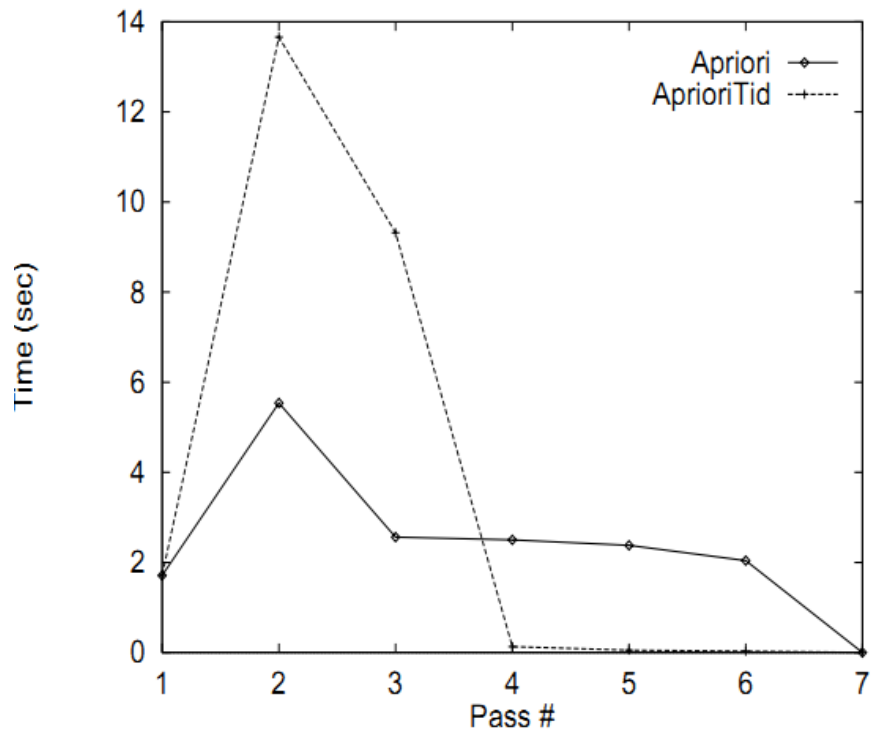






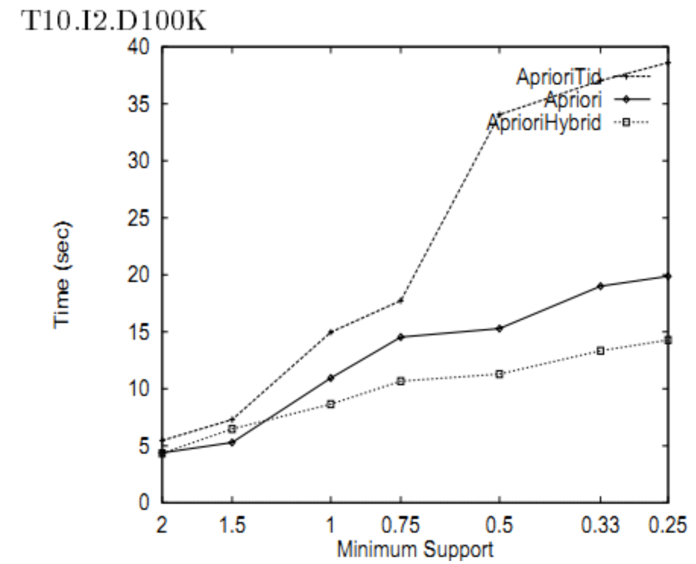
Performance

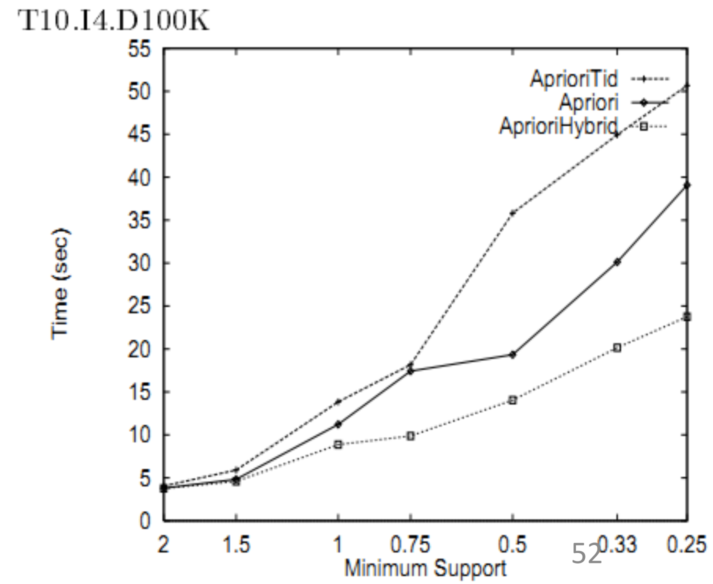
- AprioriTid is effective in later passes
  - Scans C^*_k instead of the original dataset
  - becomes small compared to original dataset
- When fits in memory, AprioriTid is faster than Apriori



AprioriHybrid

- Use Apriori in initial passes
- Switch to AprioriTid when it can fit in memory
- Switch happens at the end of the pass
  - Has some overhead to switch
- Still mostly better or as good as apriori





optional slide

From Apriori Algorithm

Subset Function - 1

- Candidate itemsets in C_k are stored in a hash-tree (like a B-tree)
  - interior node = hash table
  - each bucket points to another node at the level below
  - leaf node = itemsets
  - recall that the itemsets are ordered
  - root at level 1 (top-most)
  - All nodes are initially leaves
  - When the number of itemsets in a leaf-node exceeds a threshold, convert it into an interior node
- To add an itemset c, start from the root and go down the tree until reach a leaf

Given a transaction t and a candidate set C_k, compute the candidates in C_k contained in t

L_1 = {*large 1-itemsets*}
For ($k = 2; L_{k-1} \neq \phi ; k++$) do begin
    C_k = apriori-gen(L_{k-1});
    forall transactions $t \in D$ do begin
        C_t = subset(C_k, t)
        forall candidates $c \in C_t$ do
            $c.count++$;
        end
    end
    $L_k = \{ c \in C_k | c.count \geq minsup \}$
end
$Answer = \bigcup_k L_k$;

optional slide

Subset Function - 2

- To find all candidates contained in a transaction t
  - if we are at a leaf
    - find which itemsets are contained in t
    - add references to them in the answer set
  - if we are at an interior node
    - we have reached it by hashing an item i
    - hash on each item that comes after i in t
    - repair
  - if we are at the root, hash on every item in t

$L_1 = \{large\ 1\text{-}itemsets\}$
For $(k = 2;\ L_{k-1} \neq \phi;\ k++)$ do begin
    $C_k = \text{apriori-gen}(L_{k-1})$;
    forall transactions $t \in D$ do begin
        $C_t = \text{subset}(C_k, t)$
        forall candidates $c \in C_t$ do
            $c.count++$;
        end
    end
    $L_k = \{ c \in C_k | c.count \geq minsup\}$
end
$Answer = \bigcup_k L_k$;

optional slide

Subset Function - 3

- Why does it work?
- For any itemset c in a transaction t
  - the first item of c must be in t
  - by hashing on each item in t, we ensure that we only ignore itemsets that start with an item not in t
  - similarly for lower depths
  - since the itemset is ordered, if we reach by hashing on i, we only need to consider items that occur after i

L_1 = *{large 1-itemsets}*
For $(k = 2;\ L_{k-1} \neq \phi;\ k++)$ do begin
  C_k = apriori-gen(L_{k-1});
  forall transactions $t \in D$ do begin
    C_t = subset(C_k, t)
    forall candidates $c \in C_t$ do
      $c.count++$;
    end
  end
  $L_k = \{c \in C_k \mid c.count \geq minsup\}$
end
$Answer = \bigcup_k L_k$;

Conclusions

(of 516, Spring 2016)

Take-Aways

- DBMS Basics
- DBMS Internals
- Overview of Research Areas
- Hands-on Experience in DB systems

DB Systems

- Traditional DBMS
  - PostGres, SQL
- Large-scale Data Processing Systems
  - Spark/Scala, AWS
- New DBMS/NOSQL
  - MongoDB

- In addition
  - XML, JSON, JDBC, Python/Java

DB Basics

- SQL
- RA/Logical Plans
- RC
- Datalog
  – Why we needed each of these languages

- Normal Forms

DB Internals and Algorithms

- Storage
- Indexing
- Operator Algorithms
  - External Sort
  - Join Algorithms
- Cost-based Query Optimization
- Transactions
  - Concurrency Control
  - Recovery

Large-scale Processing and New Approaches

- Parallel DBMS
- Distributed DBMS
- Map Reduce
- NOSQL

Advanced/Research Topics

(In various levels of details)

- Data Warehouse/OLAP/Data Cube
- Data Privacy
- View Selection
- Data Provenance
- Probabilistic Databases
- Crowdsourcing
- Could not cover many more….

Hope some of you will further explore Database Systems/Data Management/Data Analysis/Big Data…

Thank you ☺