Data Mining

CPS 216

Advanced Database Systems

Data mining

- Data → knowledge
- DBMS meets AI and statistics
- Usually complex statistical "queries" that are difficult to answer

» Warehousing is a must if data needs to be integrated from various sources

» Often done using specialized algorithms outside the DBMS

  – Some recent work on pushing mining inside DBMS (Sarawagi et al., SIGMOD 1998)

Data mining problems

- Clustering: group together similar items and separate dissimilar ones
- Prediction: predict values of some attributes from others based on training data
  - Classification: predict the "class label"
  - Regression: predict a numeric attribute value
- Association analysis: detect attribute-value conditions that occur frequently together
- Outlier analysis, evolution analysis, etc., etc.

Data mining applications

- Business
  - Marketing, finance, investment, insurance…
    - Urban legend: WalMart discovered that people who bought diapers tended to buy beer at the same time
- Science
  - Astronomy, environmental science, genomics…
- Law enforcement
  - Fraud detection, criminal profiling…

Association rule mining

A.k.a. market-basket analysis

- A transaction (market basket) contains a set of items bought together
- Given a lot of transactions, discover rules such as "diaper ⇒ beer" or "digital camera, scanner ⇒ graphics software"

TID	*items*
T001	diaper, milk, candy
T002	milk, egg
T003	milk, beer
T004	diaper, milk, egg
T005	diaper, beer
T006	milk, beer
T007	diaper, beer
T008	diaper, milk, beer, candy
T009	diaper, milk, beer
…	…

Association rules

- An association rule has the form $X \Rightarrow Y$, where X and Y are disjoint itemsets (sets of items)
  - Confidence c%: c% of the transactions that contain X also contain Y
  - Support s%: s% of all transactions contain both X and Y

  » Note: association rules are directional
    - "Diaper ⇒ beer" and "beer ⇒ diaper" mean different things
- Problem: Given a set of transactions, find all association rules with confidence and support greater than or equal to specified thresholds c_{min}% and s_{min}%

Mining association rules

- Step 1: Find frequent itemsets, and count the number of times they appear in transactions
  - An itemset X is frequent if no less than s_{min}% of all transactions contain X
    - That is, count(X) ≥ s_{min}% · total # of transactions
- Step 2: Mine association rules from frequent itemsets

7

Finding frequent itemsets

- First try: a brute-force approach
  - Keep a running count for each possible itemset
  - For each transaction T, and for each itemset X, if T contains X then increment the count for X
  - Return itemsets with large enough counts
- Problem: The number of itemsets is huge!
  - 2^n, where n is the number of items
- Think: How do you prune the search space?

8

The Apriori property

- All subsets of a frequent itemset must also be frequent
  - Because any transaction that contains X must also contains subsets of X

» If you have already verified that X is infrequent, there is no need to count X's supersets because they must be infrequent too

9

The Apriori algorithm

Agrawal & Srikant, VLDB 1994

- Multiple passes over the transactions
- Pass k finds all frequent k-itemsets (itemset of size k)
- Use the set of frequent (k – 1)-itemsets found in the previous pass to narrow the search for k-itemsets

10

Pseudo-code for Apriori

Scan the transactions to find L_1, the set of all frequent 1-itemsets, together with their counts;

for ($k = 2$; $L_{k-1} \neq \varnothing$; k++) {

  Generate C_k, the set of candidate k-itemsets, from L_{k-1}, the set of frequent (k – 1)-itemsets found in the previous step;

  Scan the transactions to count the occurrences of itemsets in C_k;

  Find L_k, a subset of C_k containing k-itemsets with counts no less than (s_{min}% · total # of transactions); }

Return $L_1 \cup L_2 \cup \ldots \cup L_k$;

11

Candidate generation

From L_{k-1} to C_k

- Join: combine frequent (k – 1)-itemsets to form candidate k-itemsets
- Prune: ensure every size-(k – 1) subset of a candidate is frequent

12

Candidate generation: join

- Combine almost-matching pairs of frequent $(k-1)$-itemsets
  - 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$_1$
    AND p.item$_2$ = q.item$_2$ AND …
    AND p.item$_{k-2}$ = q.item$_{k-2}$
    AND p.item$_{k-1}$ < q.item$_{k-1}$;
    - » The last conjunct ensures no duplicates
    - » Can you justify why p and q should almost match?

13

Candidate generation: prune

- Remove candidates with an infrequent size-$(k-1)$ subset
  - for each itemset c in C_k:
    for each item x in c:
    if $c - \{x\}$ is not in L_{k-1} then:
    delete c from C_k

14

Example: pass 1

TID	*items*
T001	A, B, E
T002	B, D
T003	B, C
T004	A, B, D
T005	A, C
T006	B, C
T007	A, C
T008	A, B, C, E
T009	A, B, C
T010	F

Transactions

$s_{min}\% = 20\%$

L_1

itemset	*count*
{A}	6
{B}	7
{C}	6
{D}	2
{E}	2

Itemset { F } is infrequent

15

Example: pass 2

TID	*items*
T001	A, B, E
T002	B, D
T003	B, C
T004	A, B, D
T005	A, C
T006	B, C
T007	A, C
T008	A, B, C, E
T009	A, B, C
T010	F

Transactions

$s_{min}\% = 20\%$

Generate candidates → Scan and count → Check min. support

L_1

itemset	*count*
{A}	6
{B}	7
{C}	6
{D}	2
{E}	2

C_2

itemset
{A, B}
{A, C}
{A, D}
{A, E}
{B, C}
{B, D}
{B, E}
{C, D}
{C, E}
{D, E}

C_2

itemset	*count*
{A, B}	4
{A, C}	4
{A, D}	1
{A, E}	2
{B, C}	4
{B, D}	2
{B, E}	2
{C, D}	0
{C, E}	1
{D, E}	0

L_2

itemset	*count*
{A, B}	4
{A, C}	4
{A, E}	2
{B, C}	4
{B, D}	2
{B, E}	2

16

Example: pass 3

TID	*items*
T001	A, B, E
T002	B, D
T003	B, C
T004	A, B, D
T005	A, C
T006	B, C
T007	A, C
T008	A, B, C, E
T009	A, B, C
T010	F

Transactions

$s_{min}\% = 20\%$

Generate candidates → Scan and count → Check min. support

L_2

itemset	*count*
{A, B}	4
{A, C}	4
{A, E}	2
{B, C}	4
{B, D}	2
{B, E}	2

C_3

itemset
{A, B, C}
{A, B, E}
~~{A, C, E}~~
~~{B, C, D}~~
~~{B, C, E}~~
~~{B, D, E}~~

C_3

itemset	*count*
{A, B, C}	2
{A, B, E}	2

L_3

itemset	*count*
{A, B, C}	2
{A, B, E}	2

17

Example: pass 4

TID	*items*
T001	A, B, E
T002	B, D
T003	B, C
T004	A, B, D
T005	A, C
T006	B, C
T007	A, C
T008	A, B, C, E
T009	A, B, C
T010	F

Transactions

$s_{min}\% = 20\%$

Generate candidates

L_3

itemset	*count*
{A, B, C}	2
{A, B, E}	2

C_4

itemset
~~{A, B, C, E}~~

C_4 and L_4 are empty

18

Example: final answer

L_1

itemset	count
{A}	6
{B}	7
{C}	6
{D}	2
{E}	2

L_2

itemset	count
{A, B}	4
{A, C}	4
{A, E}	2
{B, C}	4
{B, D}	2
{B, E}	2

L_3

itemset	count
{A, B, C}	2
{A, B, E}	2

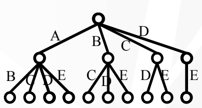
Mining rules from frequent itemsets

- for each frequent itemset l:
  for each nonempty proper subset s of l:
  if confidence = count(l) ⁄ count(s) $\geq c_{\min}$% then:
  output $s \Rightarrow (l - s)$;
- Example: rules from l = {A, B, E} are
  - A, B ⇒ E (confidence 2/4 = 50%)
  - A, E ⇒ B (confidence 2/2 = 100%)
  - B, E ⇒ A (confidence 2/2 = 100%)
  - A ⇒ B, E (confidence 2/6 = 33%)
  - B ⇒ A, E (confidence 2/7 = 29%)
  - E ⇒ A, B (confidence 2/2 = 100%)

itemset	count
{A}	6
{B}	7
{C}	6
{D}	2
{E}	2
{A, B}	4
{A, C}	4
{A, E}	2
{B, C}	4
{B, D}	2
{B, E}	2
{A, B, C}	2
{A, B, E}	2

Data structure for counting itemsets

- Problem: Given a transaction, determine which itemsets in C_k it contains
- Idea: Build a tree structure from C_k
- Example: C_2 = { {A, B}, {A, C}, {A, D}, {A, E}, {B, C}, {B, D}, {B, E}, {C, D}, {C, E}, {D, E} }

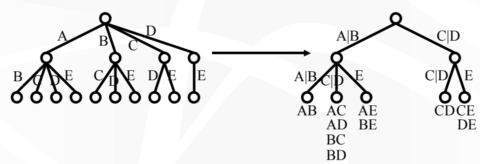


Transaction: A, D, E, F

- Remember all nodes visited
- If item matches any outgoing edge, follow it
- At a leaf node, increment count

Hash tree

- What if the tree may too big or imbalanced?
- Merge edges using a hash function
- Leaves must now store the exact itemset



A and B hash to the same key
C and D hash to the same key

Other tricks and extensions

- Transaction reduction
  - If a transaction does not contain any frequent k-itemset, remove it from further consideration
    - » AprioriTid, AprioriHybrid, from the same paper
- Dynamic itemset counting
  - Why only introduce candidate itemsets at the end of a scan? Start counting them whenever there is enough support from smaller itemsets
  - Fewer passes over data
    - » Brin et al., SIGMOD 1997
- Parallelization, sampling, incremental mining, etc.

End-semester logistics

- Project
  - Demo: You should have received an email about your scheduled slot
  - Report: Due on the day of the final exam
- Final exam
  - Thursday, December 13, 9:00am – 12:00pm
  - In this room
  - Comprehensive, emphasis on the latter half
  - Open book, open notes
- TA and instructor office hours
  - Same as regular office hours

Some points to remember from 216

- Declarativeness is good
  - Relational model, relational algebra, SQL, …
- Redundancy is bad
  - Normal forms, decomposition, …
- Redundancy is good (for performance, as long as you can hide it)
  - Replication, warehousing, materialized views, indexes, …
- One more level of indirection solves lots of things
  - Secondary indexes, wrappers, …
- Query optimizer is really "query goodifier"
  - Assumptions and heuristics to narrow the search space…
- Think beyond tables
  - Bitmap indexes, wavelet histograms, data cube, MOLAP, …

25

Next semester

- CPS 296.1: Advanced topics in databases
  - Data mining
  - XML and Web data processing
  - Incremental and approximate query evaluation
  - Materialized views for caching and warehousing

26