(A Glimpse of) Data Mining

Introduction to Databases

CompSci 316 Fall 2019

DUKE
COMPUTER SCIENCE

Announcements (Wed., Dec. 4)

- Homework 4 X2 due today
- Last Gradiance exercise due Fri.
- Project demos to start this weekend
  - Deadline for signing up is tonight!
  - Schedule will be announced via email by Fri.
  - Last weekly progress update due tonight on Piazza
- Final exam Thu. Dec. 12 2-5pm
  - Open-book, open-notes
  - Comprehensive, but with strong emphasis on the second half of the course
  - Sample final + solution posted on Sakai
- Course evals: earn 2 free points on the final exam
  - Deadline is Mon. Dec. 9

Data mining

- Data → knowledge
- DBMS meets AI and statistics
- Clustering, prediction (classification and regression), association analysis, outlier analysis, evolution analysis, etc.
  - Usually complex statistical "queries" that are difficult to answer → often specialized algorithms outside DBMS
- We will focus on frequent itemset mining, as a sample problem in data mining

Mining frequent itemsets

- Given: a large database of transactions, each containing a set of items
  - Example: market baskets
- Find all frequent itemsets
  - A set of items X is frequent if no less than $s_{min}\%$ of all transactions contain X
  - Examples: {diaper, beer}, {scanner, color printer}

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
…	…

First try

- A naïve algorithm
  - 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 we prune the search space?

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

The Apriori algorithm

Multiple passes over the transactions

- Pass k finds all frequent k-itemsets (i.e., itemsets of size k)
- Use the set of frequent k-itemsets found in pass k to construct candidate $(k+1)$-itemsets to be counted in pass $(k+1)$
  - A $(k+1)$-itemset is a candidate only if all its subsets of size k are frequent

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

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

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

Frequent 1-itemsets

(Itemset {F} is infrequent)

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

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

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

Frequent 1-itemsets

Scan and count

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

Check min. support

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

Frequent 2-itemsets

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

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

Frequent 2-itemsets

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

Candidate 3-itemsets

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

Frequent 3-itemsets

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

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

Frequent 3-itemsets

itemset	count

Candidate 4-itemsets

No more itemsets to count!

Example: final answer

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

Frequent 1-itemsets

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

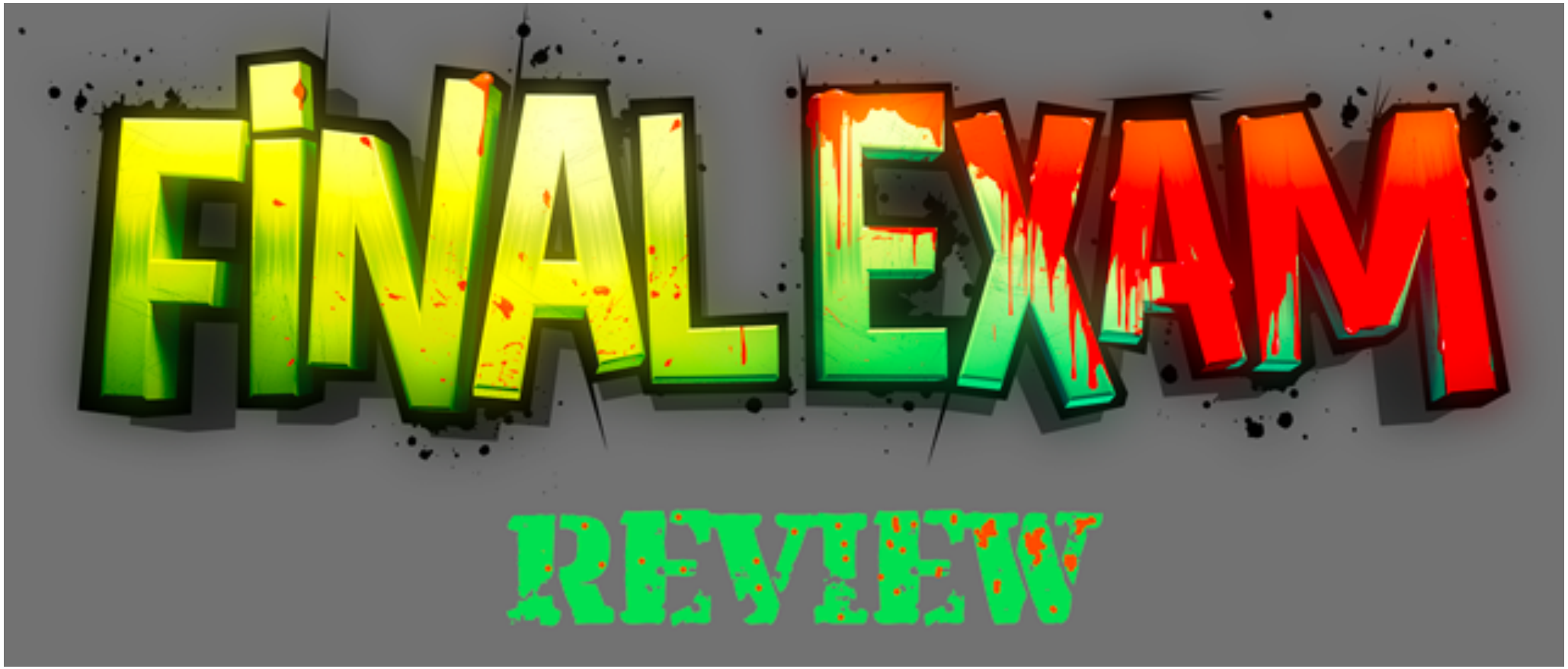
Frequent 2-itemsets

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

Frequent 3-itemsets

Summary

- Only covered frequent itemset counting
- Skipped many other techniques (clustering, classification, regression, etc.)
- Compared with statistics and machine learning: more focus on massive datasets and I/O-efficient algorithms



https://mrpappasteach.files.wordpress.com/2014/06/b8335-final_exam_logo.png

Relational basics

- Relational model + query languages: physical data independence
- Relation algebra (set semantics)
- SQL (bag semantics by default)
- Schema design
  - Entity-relationship design
  - Theory (FD's, MVD's, BNCF, 4NF): help eliminate redundancy

More about SQL

- `NULL` and three-valued logic: nifty but messy
- Bag vs. set: beware of broken equivalences
- `SELECT-FROM-WHERE` (SPJ)
- Grouping, aggregation, ordering
- Subqueries (including correlated ones)
- Modifications
- Constraints: the more you know the better
- Triggers (ECA): “active” data
- Index: reintroduce redundancy for performance
- Transactions and isolation levels

Semi-structured data

- Data models
  - XML: well-formed vs. DTD (or even XML Schema)
  - JSON: may be getting a schema too!
- Query languages:
  - XPath: (branching) path expressions (with conditions)
    - Be careful about the semantics of overloaded operators on sets
  - XQuery: FLWOR, subqueries in return (restructuring output), quantified expressions, aggregation, ordering
  - MongoDB `find()` and `aggregate()`
- Programming: SAX (streaming) vs. DOM (in-memory)
- Relational vs. XML/JSON
  - Tables vs. hierarchies
    - Flat vs. nested
  - Highly structured/typed vs. less
  - Joins vs. path traversals
  - Storing hierarchies as relations: various mapping methods

Physical data organization

- Storage hierarchy (DC vs. Pluto): so count I/Os!
- Hard drives: geometry → three components of access cost; random vs. sequential I/O
- Solid state drives: faster, but still slower than memory and still block-oriented access
- Data layout by row vs. by column
  - Different types of locality; columns easier to compress
- Access paths (indexing)
  - Primary vs. secondary; sparse vs. dense
  - Tree-based indexes: ISAM, B^+-tree
    - Big fan-out: do as much as you can with one I/O
  - Again, reintroduce redundancy to improve performance, but keep in mind the query vs. update cost trade-off

Query processing & optimization

- Processing
  - Scan-, sort-, hash-, and index-based algorithms
    - Do as much as you can with each I/O
    - Manage memory very carefully
  - Pipelined execution vs. materialization
- Optimization (or “goodification”)
  - Heuristics: push selections down; smaller joins first
    - Reduce the size of intermediate results
  - Cost-based
    - Query rewrite: de-correlate and merge query blocks to expand search space
    - Cost estimation: comes down to estimating size of intermediate results; statistics + assumptions
    - Search algorithms: greedy vs. dynamic programming (with interesting orders)

Parallel data processing

- Various performance metrics, sources of parallelism
- "Data Base" (e.g., Teradata) vs. "Big Data" (e.g., MapReduce, Spark) systems, and possible convergence
- Key ideas from Spark
  - Fewer black-box functions, more DB-style operators
  - Optimize both the execution plan (DB-style) and execution code (compiler-style)
  - RDD: use memory across the entire cluster to avoid going to Pluto altogether, but work failures must be handled more intelligently (by tracking lineage)

Transaction processing

- ACID
- Concurrency control
  - Serial and conflict-serializable scheduled
  - Locking-based: 2PL and strict 2PL
- Recovery with logging
  - Steal: requires undo logging
  - No force: requires redo logging
  - WAL: log holds the truth
  - Fuzzy checkpointing