Association Rule Mining

Instructor: Jesse Davis

Slides from: Chris Clifton,
Pedro Domingos, Jeff Ullman

Annoucements

- No class next week
  - Office hours next Tuesday 5:30-7:30/8
- Homework 3 is graded
- Homework 4 is due next Tuesday by midnight

Outline

- Homework 3 review
- Association rule mining
- Take away messages from class

Problem 1

- Accuracy 98-99% after several dozen iterations
- Generally slower than NB but higher accuracy

Problem 1: 2 BIG (RELATED) MISTAKES

- Setting bias by hand (e.g., $w_0 x_0 = 0$)
  - Every input vector should have the same x0 (say, 1)
  - Weight w0 should be *learned* like any other weight

- Not normalizing feature values to range [0,1].
  - Notice that if w0*x0 is fixed at 0 then $\Sigma w_i x_i > 0$ iff $n\Sigma w_i x_i > 0$, so normalization would indeed be unnecessary
  - If w0*x0 != 0 you must normalize to ensure that model generalizes!

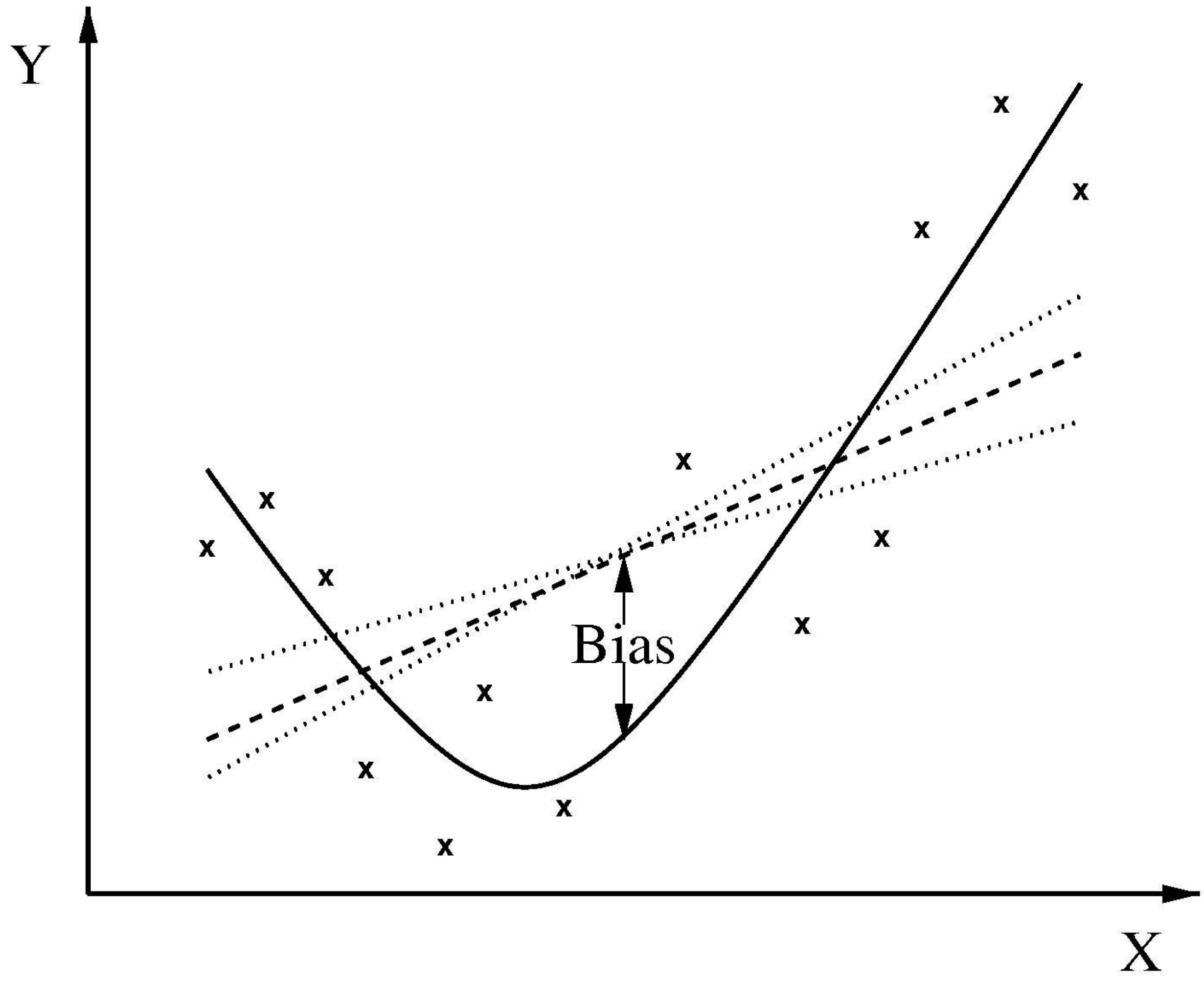
Bagging vs. Boosting

- Both techniques will improve performance of decision stumps

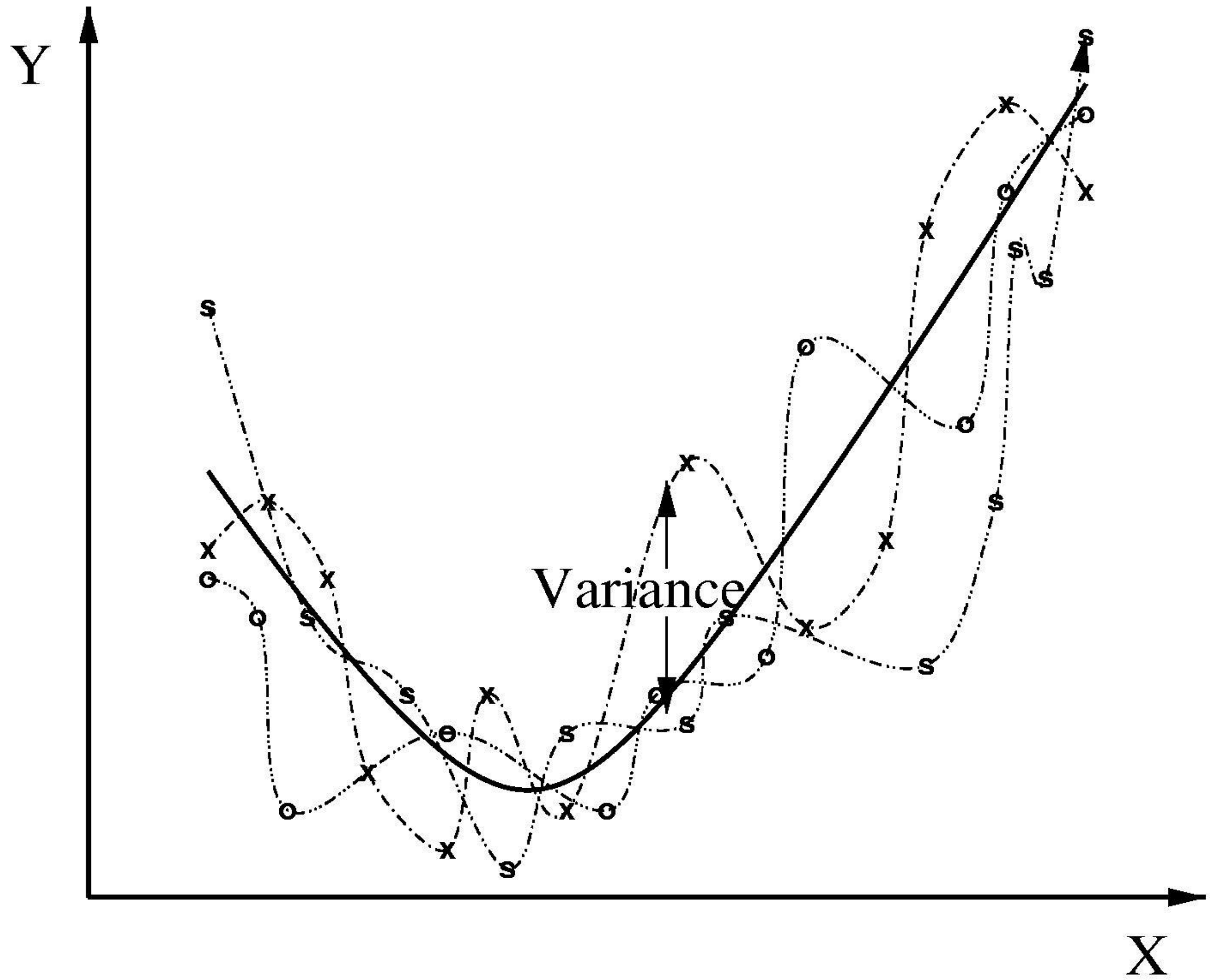
- Boosting should help more because it is better at reducing the ‘bias’ portion of error in addition to variance portion of error

- Bagging is better for handling variance

Bias



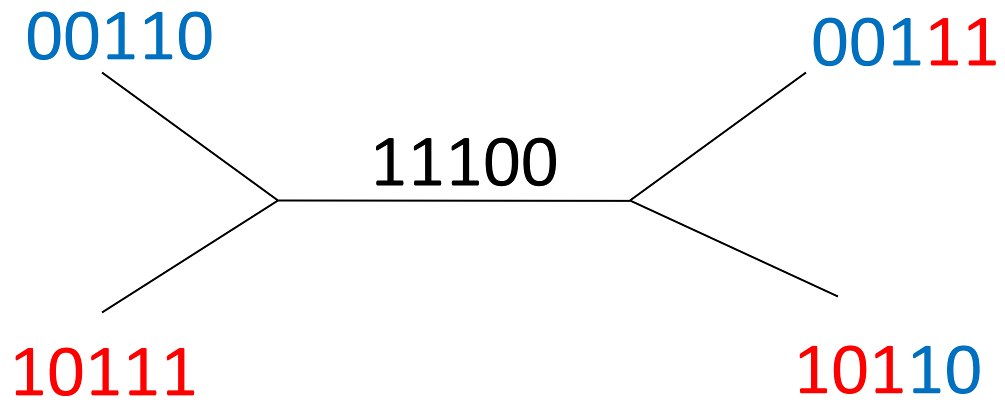
Variance



Bagging vs. Boosting - Errors

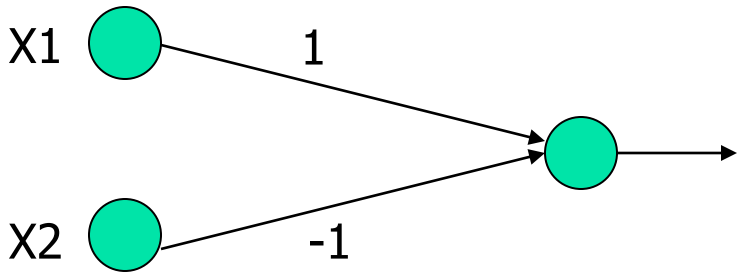
- Error 1: Bagging would help more
- Error 2: Boosting would help more
  - Explained why boosting is good
  - Didn't explain why bagging would be worse

GA Crossover



X1 ^ !X2

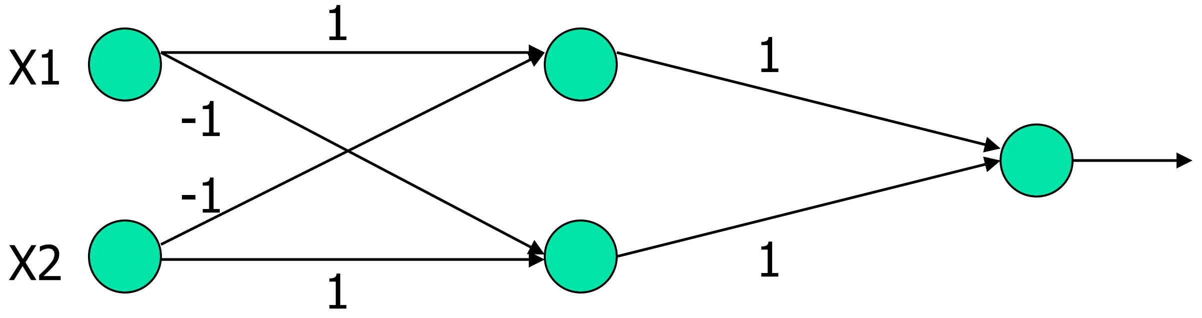
	Input	Output
a)	0 0	0
b)	0 1	0
c)	1 0	1
d)	1 1	0



If sum inputs > 0, then output is 1, else 0

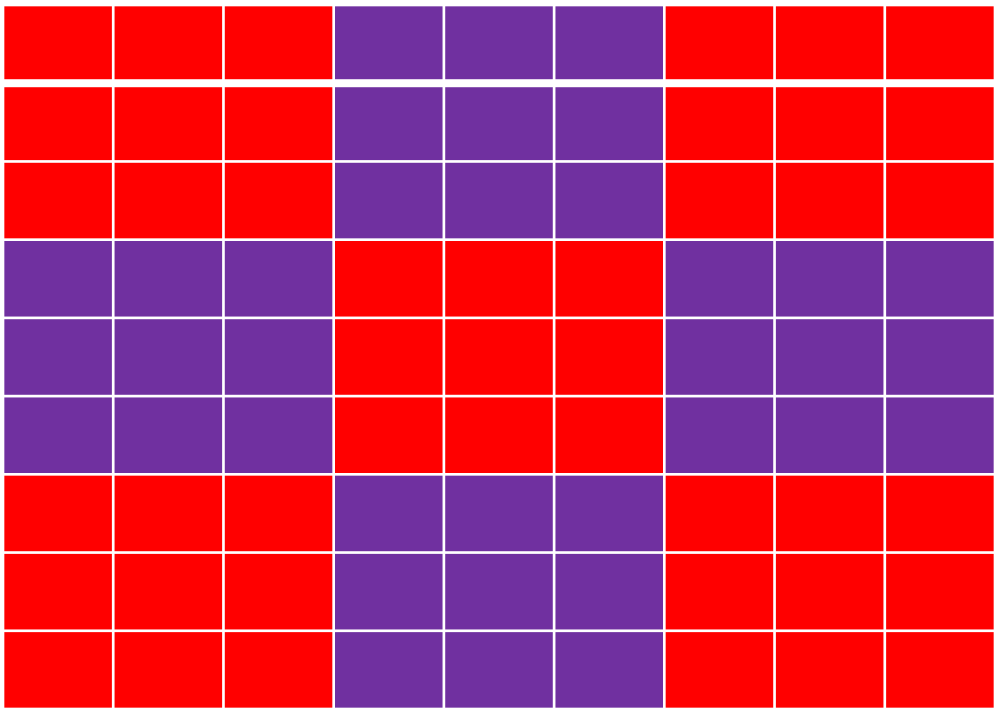
X1 XOR X2

	Input	Output
a)	0 0	0
b)	0 1	1
c)	1 0	1
d)	1 1	0



If sum inputs > 0, then output is 1, else 0

Genetic Algorithm For Sudoku

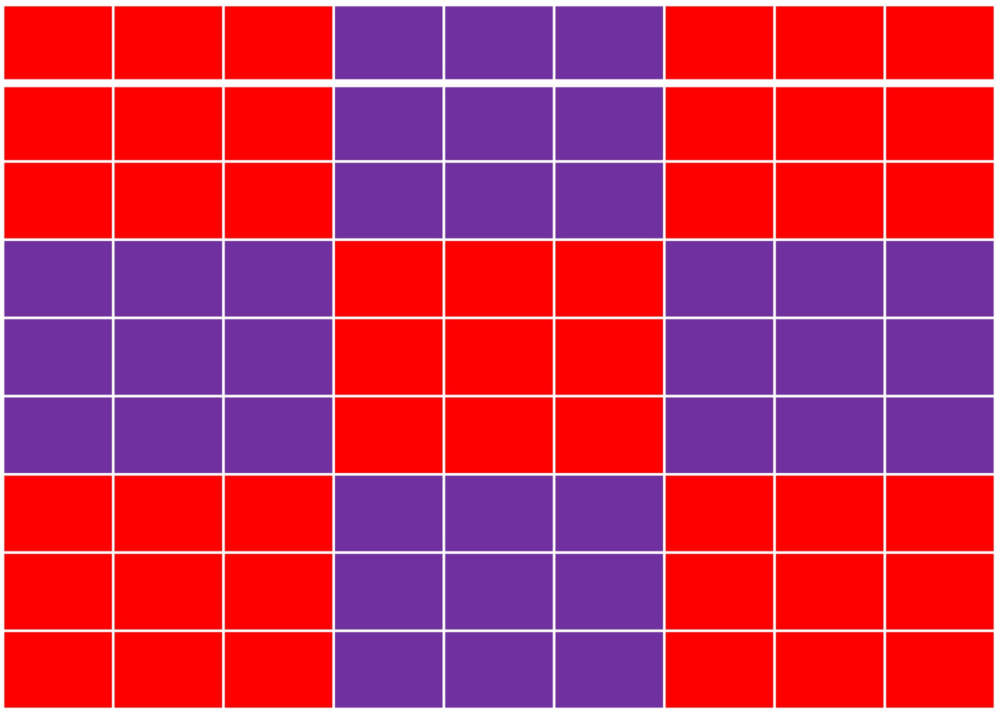


Goal: Generate Grid

Constraints:
1) Can't change givens
2) 1-9 in each 3x3 subgrid
3) 1-9 in each row
4) 1-9 in each column

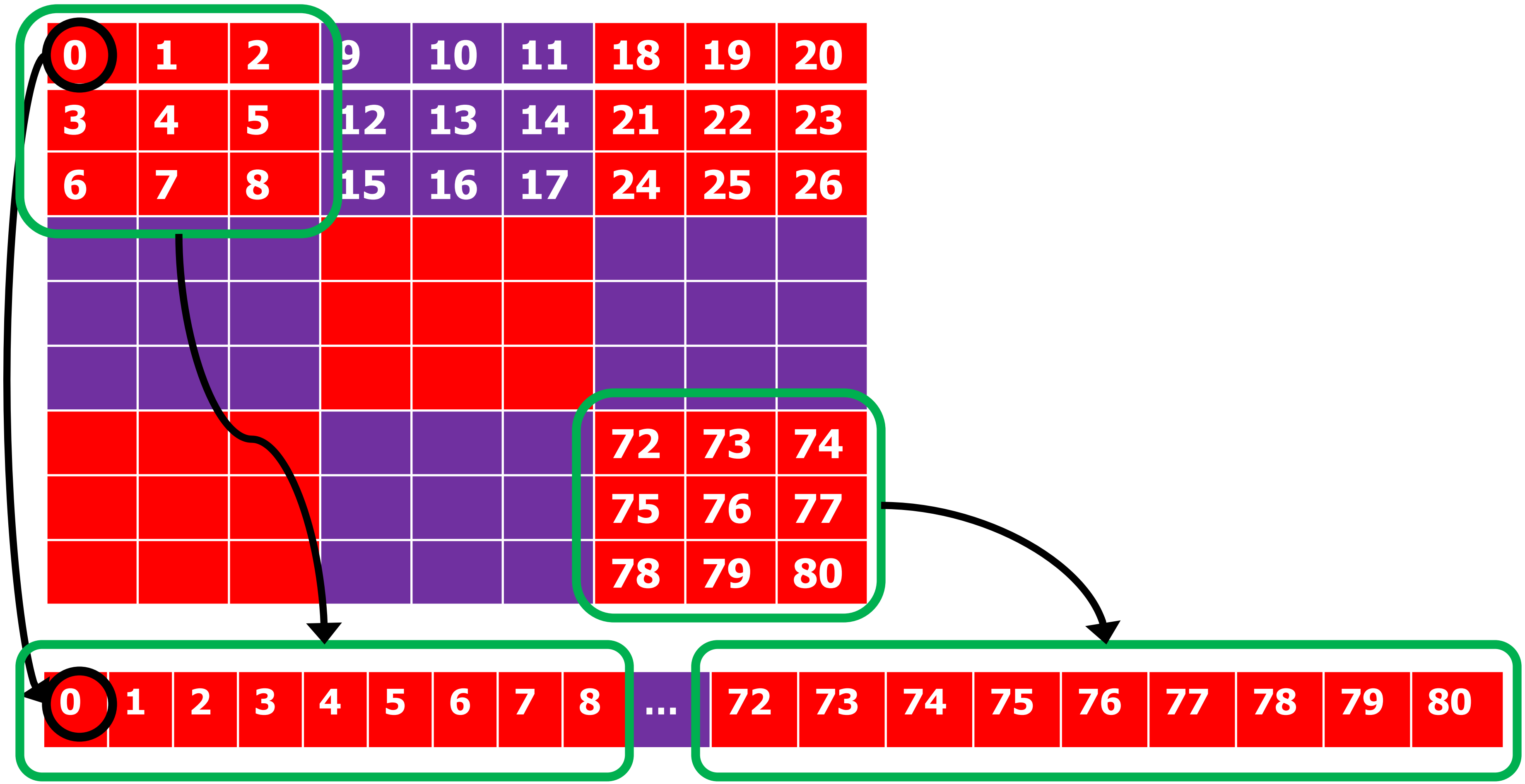
Solution components:
1) Initialiazation
2) Representation
3) Crossovers
4) Mutations
5) Fitness function

Sudoku: Initialization

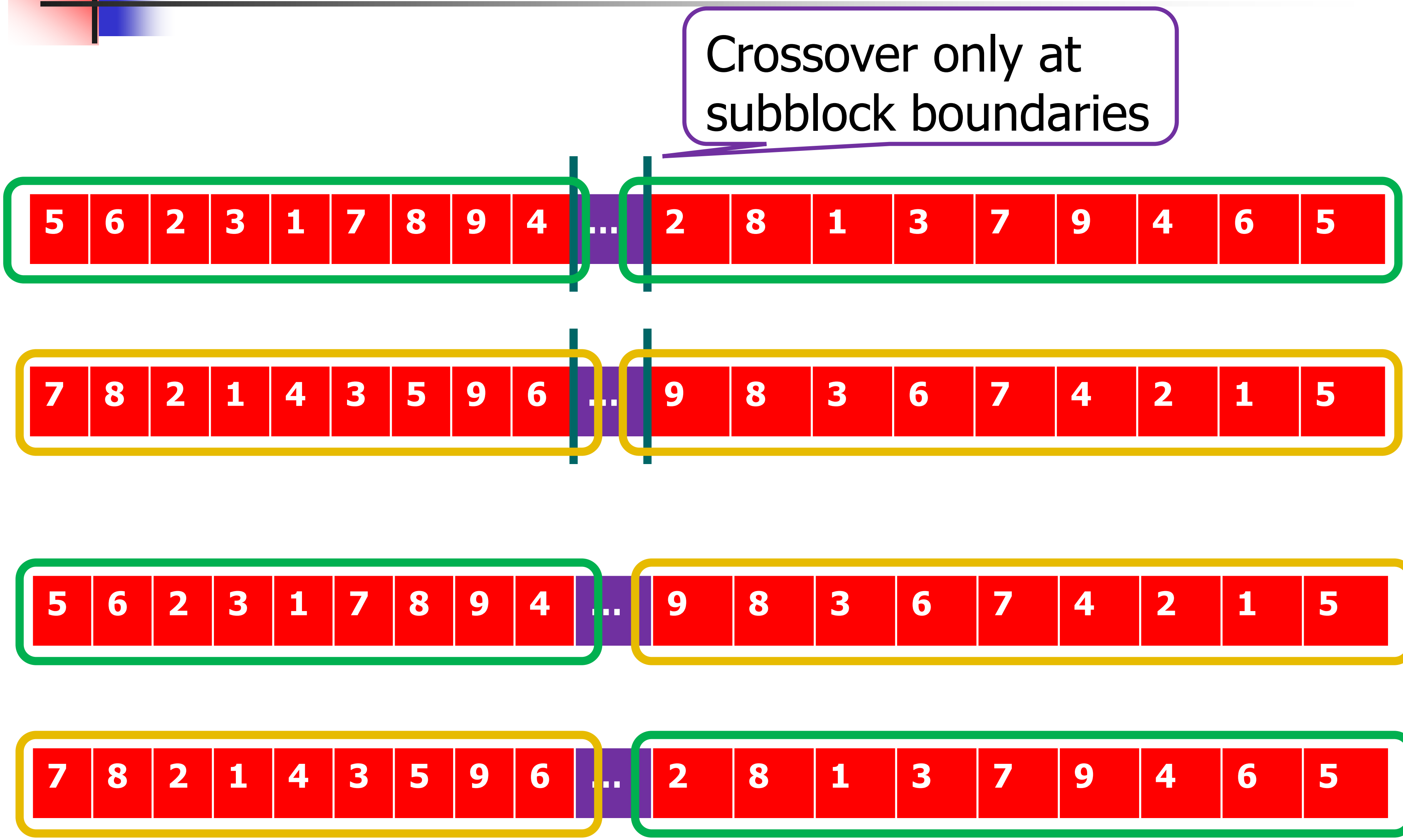


Ensure that each 3x3 subgrid has 1—9 **appearing exactly once!**

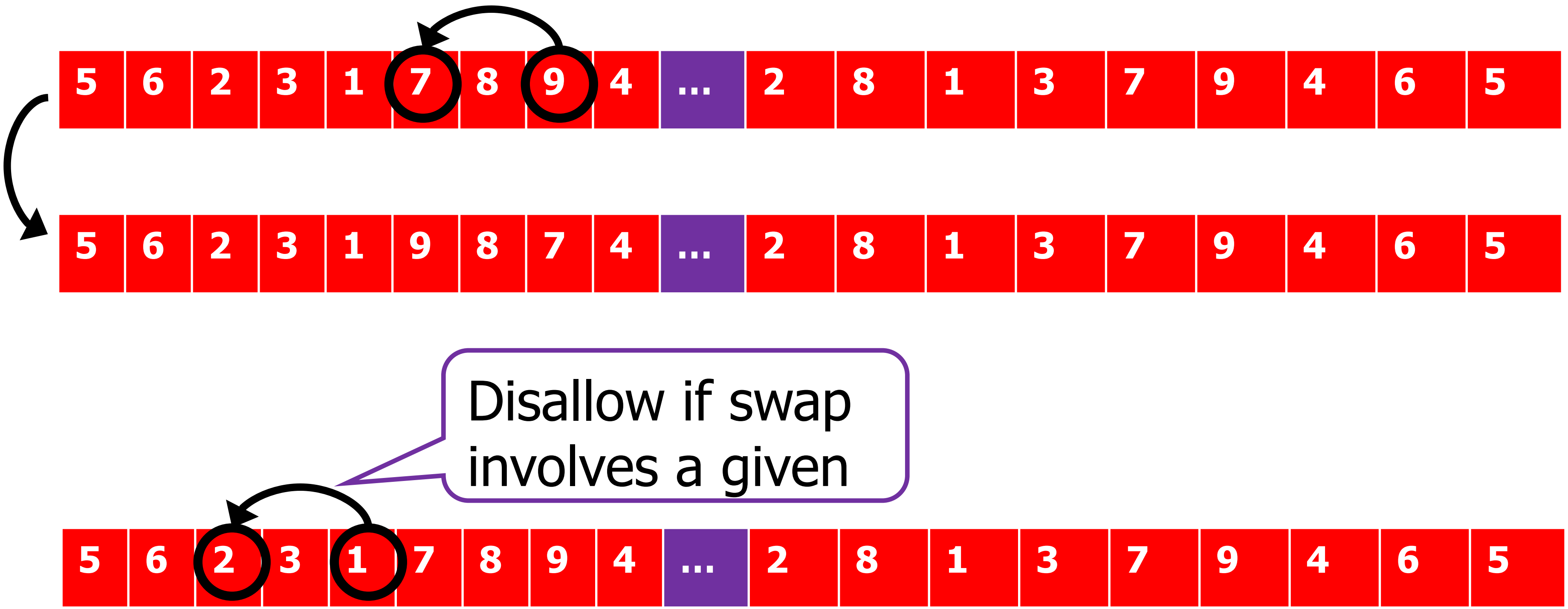
Sudoku: Representation



Sudoku: Crossovers



Sudoku: Mutations



Sudoku: Fitness Function

- Representation and operators enforce these constraints:
  - Givens are not moved around
  - Each sub-block has 1--9 appearing exactly once
- Ignore these constraints:
  - Each column has 1--9 appearing exactly once
  - Each row has 1--9 appearing exactly once
- Fitness function: Penalize these states
  - Fewer violated constraints, the fitter the solution
  - Could penalize based on “how far off” solution is, i.e., row of all 9’s is worse than row with two 9’s

Outline

- Homework 3 review
- Association rule mining
  - Introduction and definitions
  - Naïve algorithm
  - Apriori
  - PCY
  - Limiting disk I/O
  - Presenting results, other metrics
- Take away messages from class

Association Rule Mining

Given: Set of transactions
Find: Rules that predict the occurrence of an item based on other items in the transaction

TID	Items
1	Bread, Milk
2	Bread, Milk, Diaper, Beer, Eggs
3	Milk, Diaper, Beer, Coke
4	Bread, Milk, Diaper, Beer
5	Bread, Milk, Diaper, Coke

Association Rules

{Diaper} $\rightarrow$ {Beer},
{Milk, Bread} $\rightarrow$ {Eggs,Coke}
{Beer, Bread} $\rightarrow$ {Milk}

Implication means co-occurrence, not causality!

Why Association Rule Mining

- Motivation: Finding regularities in data
  - What products were often purchased together?
  - What kinds of DNA are sensitive to this new drug?
- Foundation for many data mining tasks
  - Association
  - Correlation
  - Causality
- Algorithms do not require labeled data or for a user to specify a predefined target concept

Market-Basket Model

- A large set of *items*, e.g., things sold in a supermarket
- A large set of *baskets (transactions)*, each of which is a small set of the items, e.g., the things one customer buys on one day

TID	Items
1	Bread, Milk
2	Bread, Milk, Diaper, Beer, Eggs
3	Milk, Diaper, Beer, Coke
4	Bread, Milk, Diaper, Beer
5	Bread, Milk, Diaper, Coke

Market-Baskets – (2)

- Really a general many-many mapping (association) between two kinds of things
- We ask about connections among "items," not among "baskets"
- The technology focuses on common events, not rare events ("long tail")

Definition: Item Set

- **Itemset:** A collection of one or more items
  - Example: {Bread, Milk}
- **k-itemset:** An itemset that contains k items
  - 3-itemset: {Bread, Milk, Diaper}

TID	Items
1	Bread, Milk
2	Bread, Milk, Diaper, Beer, Eggs
3	Milk, Diaper, Beer, Coke
4	Bread, Milk, Diaper, Beer
5	Bread, Milk, Diaper, Coke

Definition: Support and Frequent Itemsets

- Simplest question: find sets of items that appear "frequently" in the baskets
- *Support count* for itemset I = the number of baskets containing all items in I
- **Support** Fraction of transactions that contain an itemset
- Given a *support threshold* s, sets of items that appear in at least s baskets are called *frequent itemsets*

Example Support

Items
Bread, Milk
Bread, Milk, Diaper, Beer, Eggs
Milk, Diaper, Beer, Coke
Bread, Milk, Diaper, Beer
Bread, Milk, Diaper, Coke

Itemset	Freq
{Br,M}	4
{Br,D}	3

Support({Br,M}) = 4/5 = 0.8

Support({Br,D}) = 3/5 = 0.6

Example: Frequent Itemsets

- Items={milk, coke, pepsi, beer, juice}.
- Support = 3 baskets.

  $B_1 = \{m, c, b\}$ $\quad$ $B_2 = \{m, p, j\}$

  $B_3 = \{m, b\}$ $\quad$ $B_4 = \{c, j\}$

  $B_5 = \{m, p, b\}$ $\quad$ $B_6 = \{m, c, b, j\}$

  $B_7 = \{c, b, j\}$ $\quad$ $B_8 = \{b, c\}$
- Frequent itemsets: {m}, {c}, {b}, {j},

Example: Frequent Itemsets

- Items={milk, coke, pepsi, beer, juice}.
- Support = 3 baskets.

B_1 = {m, c, b} B_2 = {m, p, j}

B_3 = {m, b} B_4 = {c, j}

B_5 = {m, p, b} B_6 = {m, c, b, j}

B_7 = {c, b, j} B_8 = {b, c}

- Frequent itemsets: {m}, {c}, {b}, {j},

{m,b}

Example: Frequent Itemsets

- Items={milk, coke, pepsi, beer, juice}.
- Support = 3 baskets.

$B_1 = \{m, c, b\}$	$B_2 = \{m, p, j\}$
$B_3 = \{m, b\}$	$B_4 = \{c, j\}$
$B_5 = \{m, p, b\}$	$B_6 = \{m, c, b, j\}$
$B_7 = \{c, b, j\}$	$B_8 = \{b, c\}$

- Frequent itemsets: {m}, {c}, {b}, {j}, {m,b}

Example: Frequent Itemsets

- Items={milk, coke, pepsi, beer, juice}.
- Support = 3 baskets.

B_1 = {m, c, b} B_2 = {m, p, j}
B_3 = {m, b} B_4 = {c, j}
B_5 = {m, p, b} B_6 = {m, c, b, j}
B_7 = {c, b, j} B_8 = {b, c}

- Frequent itemsets: {m}, {c}, {b}, {j},

{m,b}, {b,c}

Example: Frequent Itemsets

- Items={milk, coke, pepsi, beer, juice}.
- Support = 3 baskets.

B_1 = {m, c, b} B_2 = {m, p, j}
B_3 = {m, b} B_4 = {c, j}
B_5 = {m, p, b} B_6 = {m, c, b, j}
B_7 = {c, b, j} B_8 = {b, c}

- Frequent itemsets: {m}, {c}, {b}, {j},

{m,b}, {b,c}

Example: Frequent Itemsets

- Items={milk, coke, pepsi, beer, juice}.
- Support = 3 baskets.

B_1 = {m, c, b}　　B_2 = {m, p, j}
B_3 = {m, b}　　B_4 = {c, j}
B_5 = {m, p, b}　　B_6 = {m, c, b, j}
B_7 = {c, b, j}　　B_8 = {b, c}

- Frequent itemsets: {m}, {c}, {b}, {j}, {m,b}, {b,c} , {c,j}

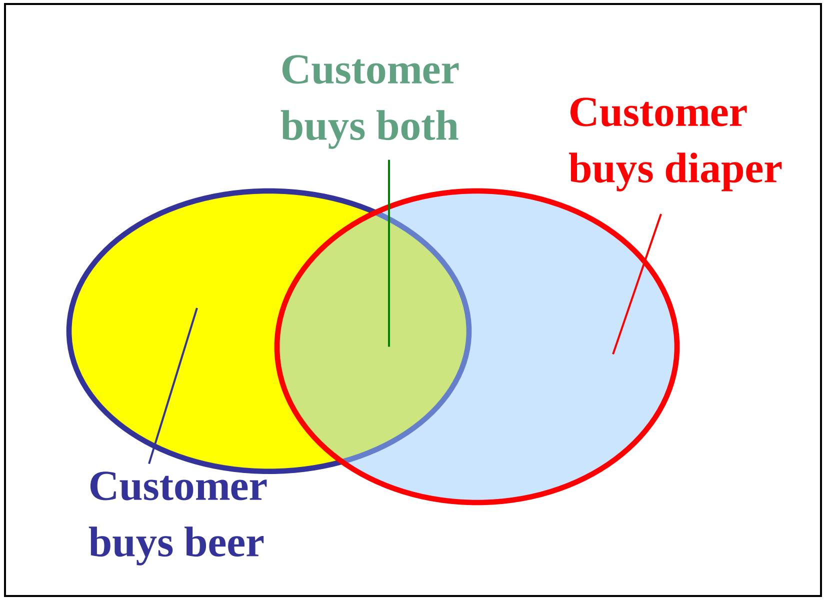
Definition: Association Rules

- If-then rules about the contents of baskets
- Given:
  - Set of *items*: $I = \{i_1, i_2, \ldots, i_m\}$
  - Set of *transactions*: $D = \{d_1, d_2, \ldots, d_n\}$
- An *association rule*: $A \Rightarrow B$, where
  - $A \subset I$
  - $B \subset I$
  - $A \cap B = \varnothing$
- $\{i_1, i_2, \ldots, i_k\} \rightarrow j$ means: "if a basket contains all of $i_1, \ldots, i_k$ then it is *likely* to contain j."

Definition: Confidence

- *Confidence* of this association rule is the conditional probability of j given $i_1,\ldots,i_k$.
  - This gives a measure of how accurate the rule is.
  - confidence($A \Rightarrow B$) = $P(B|A)$ = sup({A,B}) /sup(A)

TID	Items
1	Bread, Milk
2	Bread, Milk, Diaper, Beer, Eggs
3	Milk, Diaper, Beer, Coke
4	Bread, Milk, Diaper, Beer
5	Bread, Milk, Diaper, Coke



Example: Confidence

$+ B_1 = \{m, c, b\}$ $B_2 = \{m, p, j\}$

$- B_3 = \{m, b\}$ $B_4 = \{c, j\}$

$- B_5 = \{m, p, b\}$ $+ B_6 = \{m, c, b, j\}$

$B_7 = \{c, b, j\}$ $B_8 = \{b, c\}$

- An association rule: $\{m, b\} \rightarrow c$.
  - Confidence = 2/4 = 50%.

Applications – (1)

- Items = products; baskets = sets of products someone bought in one trip to the store.
- Example application: given that many people buy beer and diapers together:
  - Run a sale on diapers; raise price of beer.
- Only useful if many buy diapers & beer.

Applications – (2)

- Baskets = sentences; items = documents containing those sentences.
- Items that appear together too often could represent plagiarism.
- Notice items do not have to be “in” baskets.

Applications – (3)

- Baskets = Web pages; items = words.
- Unusual words appearing together in a large number of documents, e.g., “Brad” and “Angelina,” may indicate an interesting relationship.

Outline

- Homework 3 review
- Association rule mining
  - Introduction and definitions
  - Naïve algorithm
  - Apriori
  - PCY
  - Limiting disk I/O
  - Presenting results, other metrics
- Take away messages from class

Scale of the Problem

- WalMart sells 100,000 items and can store billions of baskets
- The Web has billions of words and many billions of pages
- We have access to lots and lots of data…

Association Rule Mining Goal

- Question: "find all association rules with support $\geq s$ and confidence $\geq c$."
  - Note: "support" of an association rule is the support of the set of items on the left
- Hard part: finding the frequent itemsets
  - Note: if $\{i_1, i_2, \ldots, i_k\} \rightarrow j$ has high support and confidence, then both $\{i_1, i_2, \ldots, i_k\}$ and $\{i_1, i_2, \ldots, i_k, j\}$ will be "frequent"

Creating Associating Rules

- Given: Support s, confidence c
- Step 1: Find all itemsets with support s
- Step 2: For each frequent itemset L
  - For each non-empty subset s of L
    - Output the rule s $\rightarrow$ {l-s} if its condifence $\geq$ c

Example: Association Rule

Transaction-id	Items bought
10	A, B, C
20	A, C
30	A, D
40	B, E, F

Min. support 50%
Min. confidence 50%

Frequent pattern	Support
{A}	75%
{B}	50%
{C}	50%
{A, C}	50%

For rule $A \Rightarrow C$:

support = support($\{A\} \cup \{C\}$) = 50%

confidence = support($\{A\} \cup \{C\}$)/support($\{A\}$) = 66.6%

Example: Itemset to Association Rule

Items
Bread, Milk
Bread, Milk, Diaper, Beer, Eggs
Milk, Diaper, Beer, Coke
Bread, Milk, Diaper, Beer
Bread, Milk, Diaper, Coke

Itemset	Freq
{Br,M}	4
{Br,D}	3
{M,Be}	3
{M,D}	3
{Br,M,D}	3
{M,D,Be}	3

$\{Br\} \rightarrow \{M\}$, s = 0.8, c = 1.0
$\{M\} \rightarrow \{Br\}$, s = 1.0, c = 0.8
...
$\{Br,M\} \rightarrow \{D\}$, s = 0.8, c = 0.75
$\{Be\} \rightarrow \{M,D\}$, s = 0.6, c = 1.0

Computation Model

- Typically, data is kept in flat files rather than in a database system
  - Stored on disk
  - Stored basket-by-basket
  - Expand baskets into pairs, triples, etc. as you read baskets
    - Use *k* nested loops to generate all sets of size *k*.

Computation Model – (2)

- The true cost of mining disk-resident data is usually the number of disk I/O’s
- In practice, association-rule algorithms read the data in *passes* – all baskets read in turn
- Thus, we measure the cost by the number of passes an algorithm takes

Main-Memory Bottleneck

- For many frequent-itemset algorithms, main memory is the critical resource
  - As we read baskets, we need to count something, e.g., occurrences of pairs
  - The number of different things we can count is limited by main memory
  - Swapping counts in/out is a disaster (why?)

Finding Frequent Pairs

- The hardest problem often turns out to be finding the frequent pairs
  - Often frequent pairs are common, frequent triples are rare
  - Probability of being frequent drops exponentially with size
  - number of sets grows more slowly with size
- We'll concentrate on pairs, then extend to larger sets

Naïve Algorithm

- Read file once, counting in main memory the occurrences of each pair
  - From each basket of n items, generate its $n(n-1)/2$ pairs by two nested loops
- Fails if $(\#items)^2$ exceeds main memory
  - Remember: #items can be 100K (Wal-Mart) or 10B (Web pages)

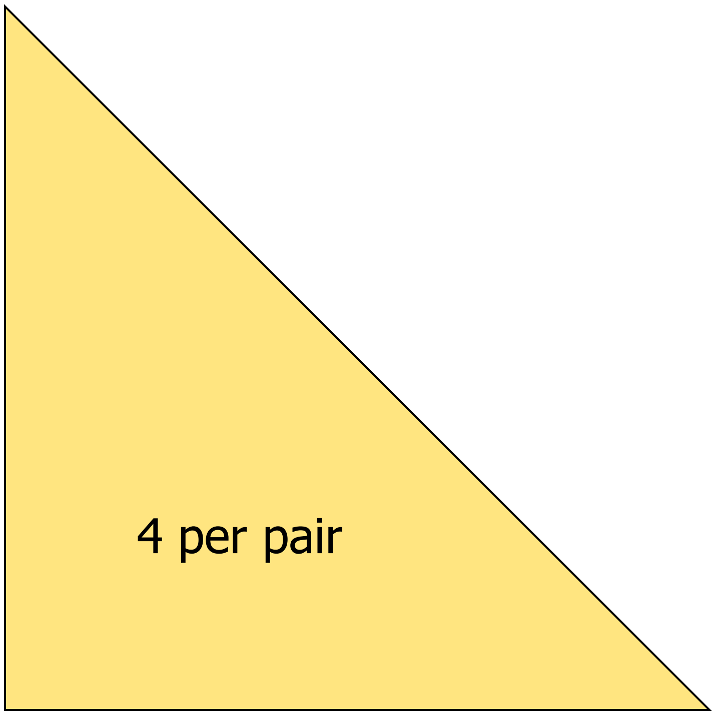
Example: Counting Pairs

- Suppose 10^5 items
- Suppose counts are 4-byte integers
- Number of pairs of items: $10^5(10^5-1)/2 = 5*10^9$ (approximately)
- Therefore, $2*10^{10}$ (20 gigabytes) of main memory needed

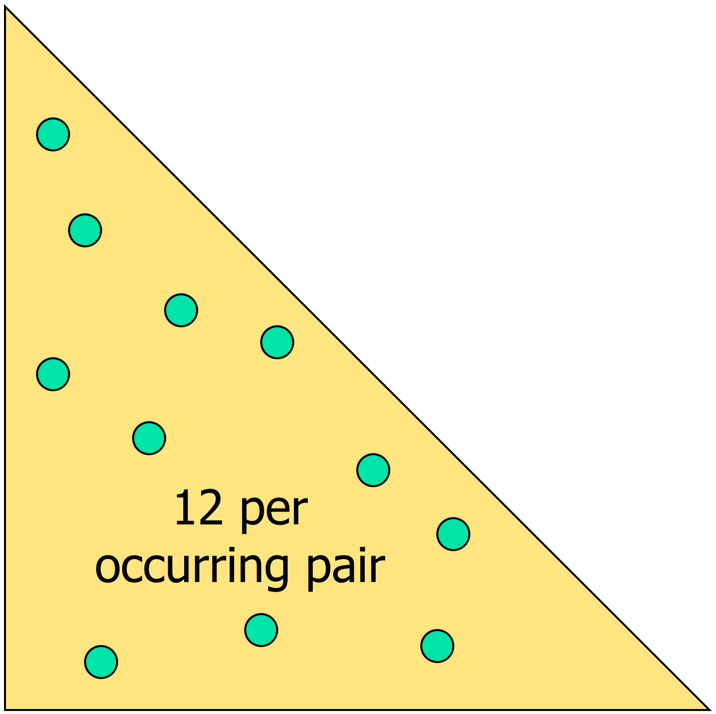
Details of Main-Memory Counting

- Two approaches:
  1. Count all pairs, using a triangular matrix.
  2. Keep a table of triples [*i*, *j*, *c*] = "the count of the pair of items {*i*, *j* } is *c*."
- 1. requires only 4 bytes/pair
- 2. requires 12 bytes, but only for those pairs with count > 0

Approaches Pictorially



Method (1)



Method (2)

Approach 1

- Assign each item a number
- Count $\{i, j\}$ only if $i < j$
- Keep pairs in the order
  - {1,2}
  - …
  - $\{1,n\}$
  - {2,3}
  - …
  - $\{n-1,n\}$
- Pair $\{i, j\}$ at the position: $(i-1)(n-i/2) + j - i$

Approach 2

- Total bytes used is about 12*p*, where *p* is the number of pairs that actually occur
- Beats triangular matrix if at most 1/3 of possible pairs actually occur
- Require extra space for retrieval structure

Outline

- Homework 3 review
- Association rule mining
  - Introduction and definitions
  - Naïve algorithm
  - Apriori
  - PCY
  - Limiting disk I/O
  - Presenting results, other metrics
- Take away messages from class

Apriori Algorithm

- Generate and test approach for discovering frequent itemsets
- Iterative approach
  - Find all frequent itemsets of size k before finding frequent itemsets of size k+1
  - One pass through the data for each frequent itemset size

Apriori's Key Idea

- *Aproiri Principle (monotonicity)*: if an itemset appears at least *s* times, so do all its subsets
  - Contrapositive for pairs: if item *i* does not appear in *s* baskets, then no pair including *i* can appear in *s* baskets
- Apriori principle holds due to the following property of the support measure:

$$\forall X, Y : (X \subseteq Y) \Rightarrow s(X) \geq s(Y)$$

A-Priori Algorithm: Frequent Pairs

- Pass 1: Read baskets and count in main memory the occurrences of each item
  - Requires memory proportional to #items
  - *Frequent items*: those that appear s times

- Pass 2: Read baskets again and count in main memory only those pairs both of which were found in Pass 1 to be frequent
  - Requires memory proportional to square of *frequent* items, plus a list of the frequent items
  - *Frequent itemsets*: those that appear s times

The Apriori Algorithm

- Join Step: C_k is generated by joining $L_{k-1}$with itself
- Prune Step: Any (k-1)-itemset that is not frequent cannot be a subset of a frequent k-itemset
- Pseudo-code:

  C_k: Candidate itemset of size k
  L_k : frequent itemset of size k

  L_1 = {frequent items}
  for (k = 1; L_k !=∅; k++) **do begin**
  C_{k+1} = candidates generated from L_k
  for each transaction t in database do
  increment the count of all candidates in C_{k+1} that are contained in t
  L_{k+1} = candidates in C_{k+1} with min_support
  end
  return $\cup_k L_k$;

Apriori: Pass 1

Given: Min support is 2

Database D

TID	Items
1	1,3,4
2	2,3,5
3	1,2,3,5
4	2,5

Scan D $\longrightarrow$

C_1

Itemset	Sup
{1}	2
{2}	3
{3}	3
{4}	1
{5}	3

Apriori: Pass 1

Given: Min support is 2

Database D

TID	Items
1	1,3,4
2	2,3,5
3	1,2,3,5
4	2,5

Scan D

C_1

Itemset	Sup
{1}	2
{2}	3
{3}	3
{4}	1
{5}	3

Prune

L_1

Itemset	Sup
{1}	2
{2}	3
{3}	3
{5}	3

Apriori: Pass 2

Given: Min support is 2

Database D

TID	Items
1	1,3,4
2	2,3,5
3	1,2,3,5
4	2,5

L_1

Itemset	Sup
{1}	2
{2}	3
{3}	3
{5}	3

C_2

Itemset
{1,2}
{1,3}
{1,5}
{2,3}
{2,5}
{3,5}

Scan D →

L_2

Itemset	Sup
{1,2}	1
{1,3}	2
{1,5}	1
{2,3}	2
{2,5}	3
{3,5}	2

Apriori: Pass 2

Given: Min support is 2

Database D

TID	Items
1	1,3,4
2	2,3,5
3	1,2,3,5
4	2,5

L_1

Itemset	Sup
{1}	2
{2}	3
{3}	3
{5}	3

C_2

Itemset
{1,2}
{1,3}
{1,5}
{2,3}
{2,5}
{3,5}

Scan D →

Prune

L_2

Itemset	Sup
{1,2}	1
{1,3}	2
{1,5}	1
{2,3}	2
{2,5}	3
{3,5}	2

Apriori: Pass 2

Given: Min support is 2

Database D

TID	Items
1	1,3,4
2	2,3,5
3	1,2,3,5
4	2,5

L_1

Itemset	Sup
{1}	2
{2}	3
{3}	3
{5}	3

C_2

Itemset
{1,2}
{1,3}
{1,5}
{2,3}
{2,5}
{3,5}

$\xrightarrow{\text{Scan D}}$

L_2

Itemset	Sup
{1,3}	2
{2,3}	2
{2,5}	3
{3,5}	2

Apriori: Pass 3

Given: Min support is 2

Database D

TID	Items
1	1,3,4
2	2,3,5
3	1,2,3,5
4	2,5

L_2

Itemset	Sup
{1,3}	2
{2,3}	2
{2,5}	3
{3,5}	2

C_3

Itemset
{2,3,5}

Scan D →

L_3

Itemset	Sup
{2,3,5}	2

Apriori: Join Step

- Suppose the items in L_{k-1} are listed in an order
- Join each element in L_{k-1} with itself
- If l1, l2 $\in L_{k-1}$, the are joinable if:
  - The first k-2 items in l1 and l2 are the same
  - l1[1] = l2[1] AND
    l1[2] = l2[2] AND
    … AND
    l1[k-2] = l2[k-2]

Apriori: Prune Step

- For each candidate itemsets C_k
  - Look at each subset of size k-1 [i.e., drop one item from the candidate]
    - If ANY one of these subsets isn't frequent, discard this candidate
    - Application of the Apriori principle

Example: Candidate Generation

- L_3={*abc, abd, acd, ace, bcd*}
- Self-joining: L_3*L_3
  - *abcd* from *abc* and *abd*
  - *acde* from *acd* and *ace*
  - Note: other joins (i.e., abc and acd, abc and ace, etc. are illegal)
- Pruning:
  - *acde* is removed because *ade* is not in L_3
- C_4={*abcd*}

Outline

- Homework 3 review
- Association rule mining
  - Introduction and definitions
  - Naïve algorithm
  - Apriori
  - PCY
  - Limiting disk I/O
  - Presenting results, other metrics
- Take away messages from class

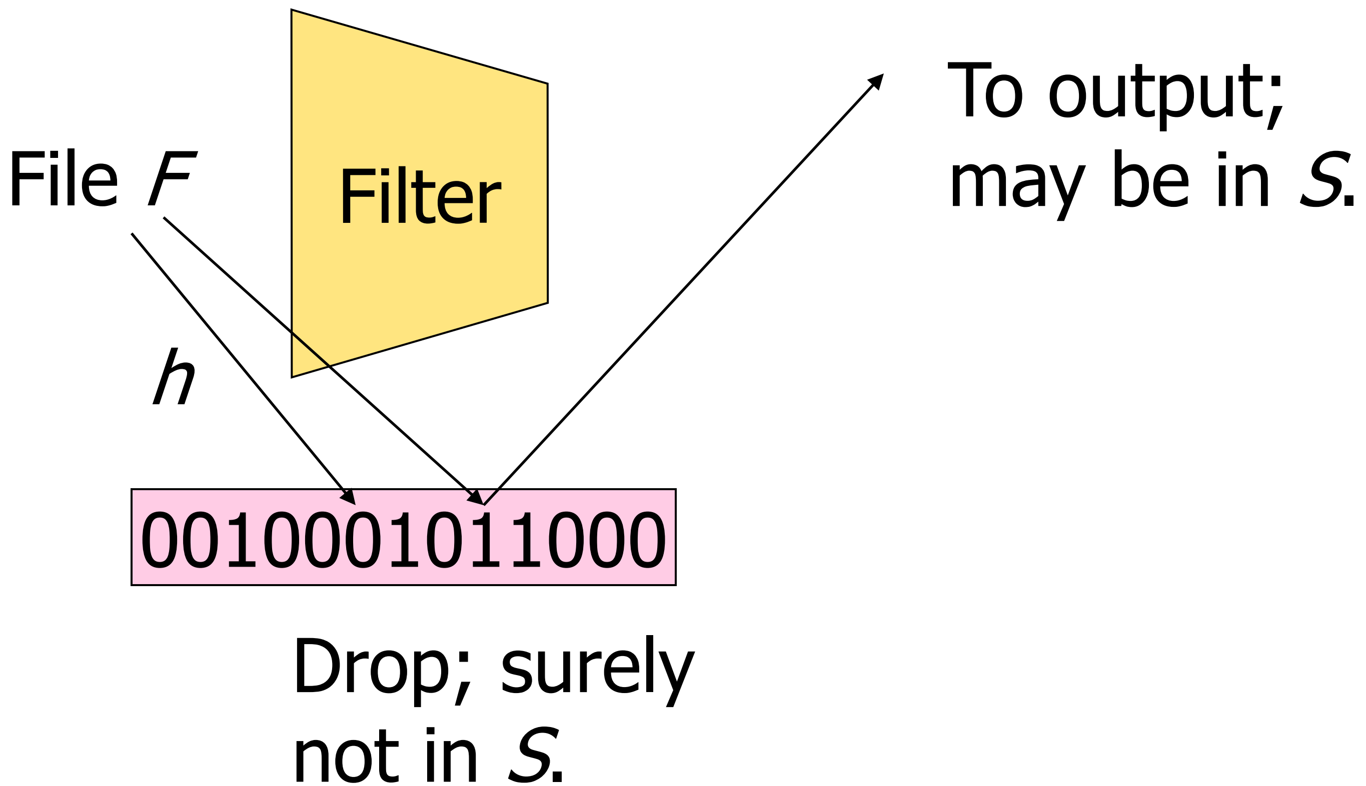
Aside: Hash-Based Filtering

- Simple problem: I have a set *S* of one billion strings of length 10.
- I want to scan a larger file *F* of strings and output those that are in *S*.
- I have 1GB of main memory.
  - So I can't afford to store *S* in memory.

Solution – (1)

- Create a bit array of 8 billion bits, initially all 0's.
- Choose a hash function *h* with range $[0, 8*10^9]$, and hash each member of *S* to one of the bits, which is then set to 1.
- Filter the file *F* by hashing each string and outputting only those that hash to a 1.

Solution – (2)


File F
Filter
h
To output;
may be in S.
0010001011000
Drop; surely
not in S.

PCY Algorithm

- During Pass 1 of A-priori, most memory is idle.
- Idea: Use tmemory for a hash table
  - Hash pairs of items that appear in a transaction – we need to generate these
  - Just the count, not the pairs themselves
  - Interested in the presence of a pair AND whether it is present at least *s* (support) times

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
    }
}
```

Observation About Buckets

- A bucket that a frequent pair hashes to meets minimum support threshold
  - Cannot eliminate any member of this bucket
- Even without any frequent pair, a bucket can be frequent
  - Cannot eliminate any member of this bucket
- Best case: Count for a bucket is less than minimum support
  - Eliminate all pairs hashed to this bucket even if the pair consists of two frequent items

PCY: Pass 1

Given: Min support is 2

Database D

TID	Items
1	1,3,4
2	2,3,5
3	1,2,3,5
4	2,5

Scan D

{1,3}, {1,4}, {3,4}

{2,3}, {2,5}, {3,5}

{1,2}, {1,3}, {1,5}, {2,3}, {2,5}, {3,5}

{2,5}

C_1

Itemset	Sup
{1}	2
{2}	3
{3}	3
{4}	1
{5}	3

Bucket	1	2	3	4	5
Count	3	2	4	1	3

PCY: Between Passes

Given: Min support is 2

Database D

TID	Items
1	1,3,4
2	2,3,5
3	1,2,3,5
4	2,5

C_1

Itemset	Sup
{1}	2
{2}	3
{3}	3
{5}	3

C_2

Itemset
{1,2}
{1,3}
{1,5}
{2,3}
{2,5}
{3,5}

C_2

Itemset
{1,3}
{1,5}
{2,3}
{2,5}
{3,5}

Bucket	1	2	3	4	5
Count	3	2	4	1	3

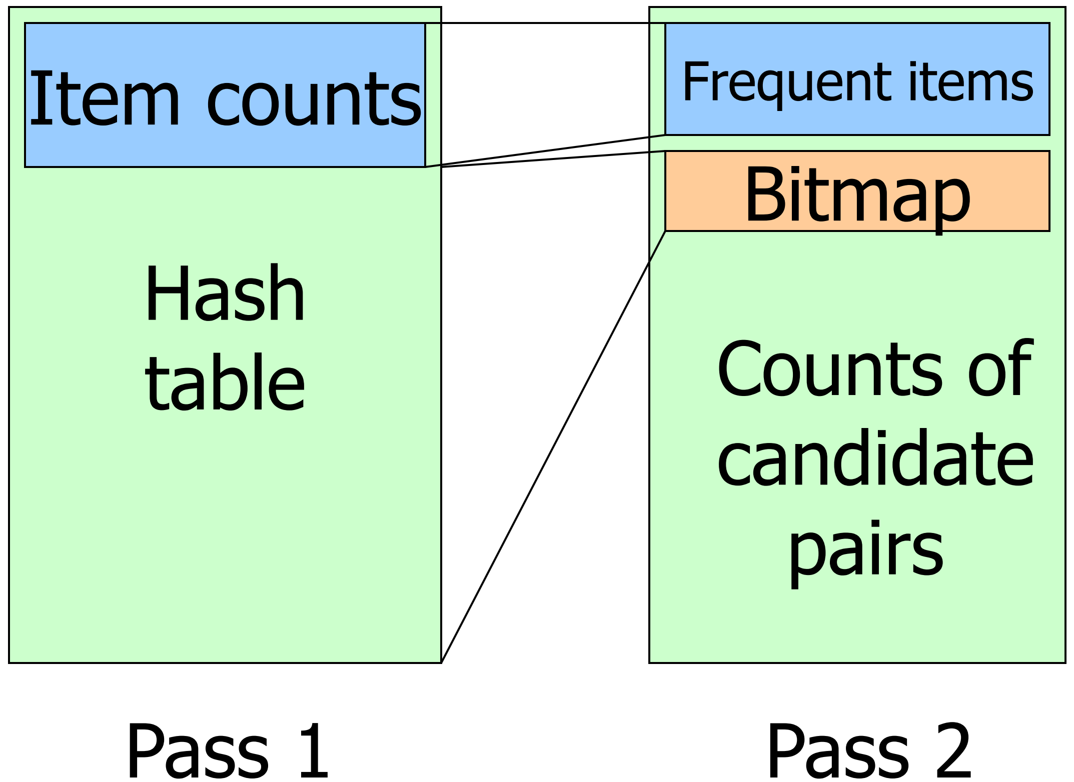
Between Passes

- Replace the buckets by a bit-vector:
  - 1 means the bucket is frequent
  - 0 means it is not frequent

- 4-byte integers are replaced by bits, so the bit-vector requires 1/32 of memory

- Also, decide which items are frequent and list them for the second pass

Picture of PCY



PCY Algorithm: Pass 2

- Count all pairs {*i*, *j* } that meet the conditions for being a candidate pair:
  1. Both *i* and *j* are frequent items.
  2. The pair {*i*, *j* }, hashes to a bucket number whose bit in the bit vector is 1.
- Notice all these conditions are necessary for the pair to have a chance of being frequent.

Outline

- Homework 3 review
- Association rule mining
  - Introduction and definitions
  - Naïve algorithm
  - Apriori
  - PCY
  - Limiting disk I/O
  - Presenting results, other metrics
- Take away messages from class

All (Or Most) Frequent Itemsets in $\leq$ 2 Passes

- A-Priori, PCY, etc., take *k* passes to find frequent itemsets of size *k*
- Other techniques use 2 or fewer passes for all sizes:
  - Simple algorithm
  - SON (Savasere, Omiecinski, and Navathe)
  - Toivonen

Simple Algorithm

- Take a random sample of the market baskets that fits in main memory
- Run a-priori or one of its improvements in main memory, so you don't pay for disk I/O each time you increase the size of itemsets
  - Be sure you leave enough space for counts

Copy of sample baskets	Space for counts

Algorithm Details

- Scale back support threshold a suitable number
  - E.g., if sample is 1/100 of the baskets, use $s/100$ as your support threshold instead of s
- Optional: Verify that your guesses are truly frequent in the entire data set by a second pass

- Miss sets frequent in whole but not in sample
  - Smaller threshold, e.g., $s/125$, helps limit misses, but requires more space

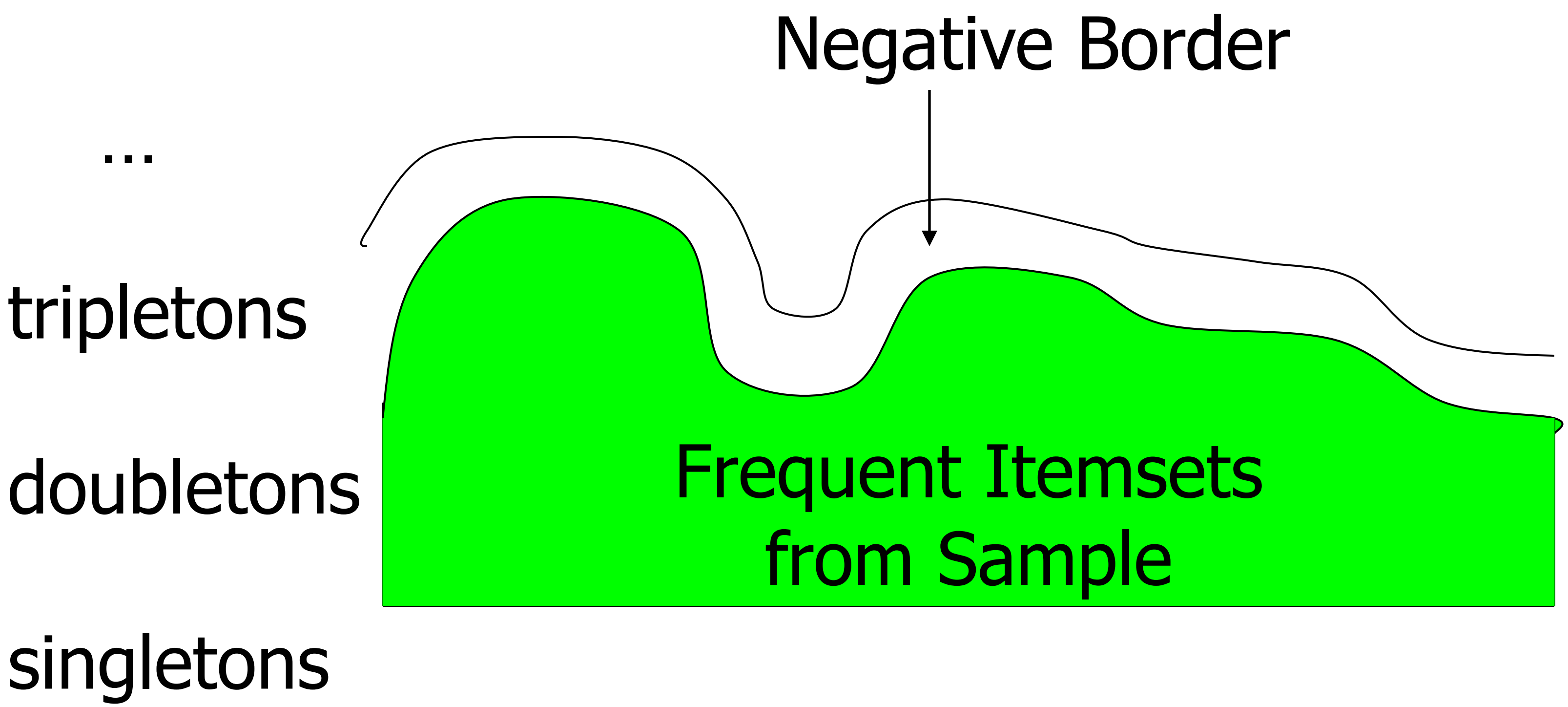
Toivonen's Algorithm

- Use simple algorithm, but lower the threshold s for the sample
  - Example: if the sample is 1% of the baskets, use *s* /125 vs. *s* /100.
  - Goal: Avoid missing truly frequent itemsets
- Add to the itemsets that are frequent in the sample the *negative border* of these itemsets.
- An itemset is in the negative border if it is not deemed frequent in the sample, but *all* its immediate subsets are

Example: Negative Border

- *ABCD* is in the negative border if and only if:
  1. It is not frequent in the sample, but
  2. All of *ABC*, *BCD*, *ACD*, and *ABD* are.
- *A* is in the negative border if and only if it is not frequent in the sample.
  - Because the empty set is always frequent.
    - Unless there are fewer baskets than the support threshold (silly case).

Picture of Negative Border



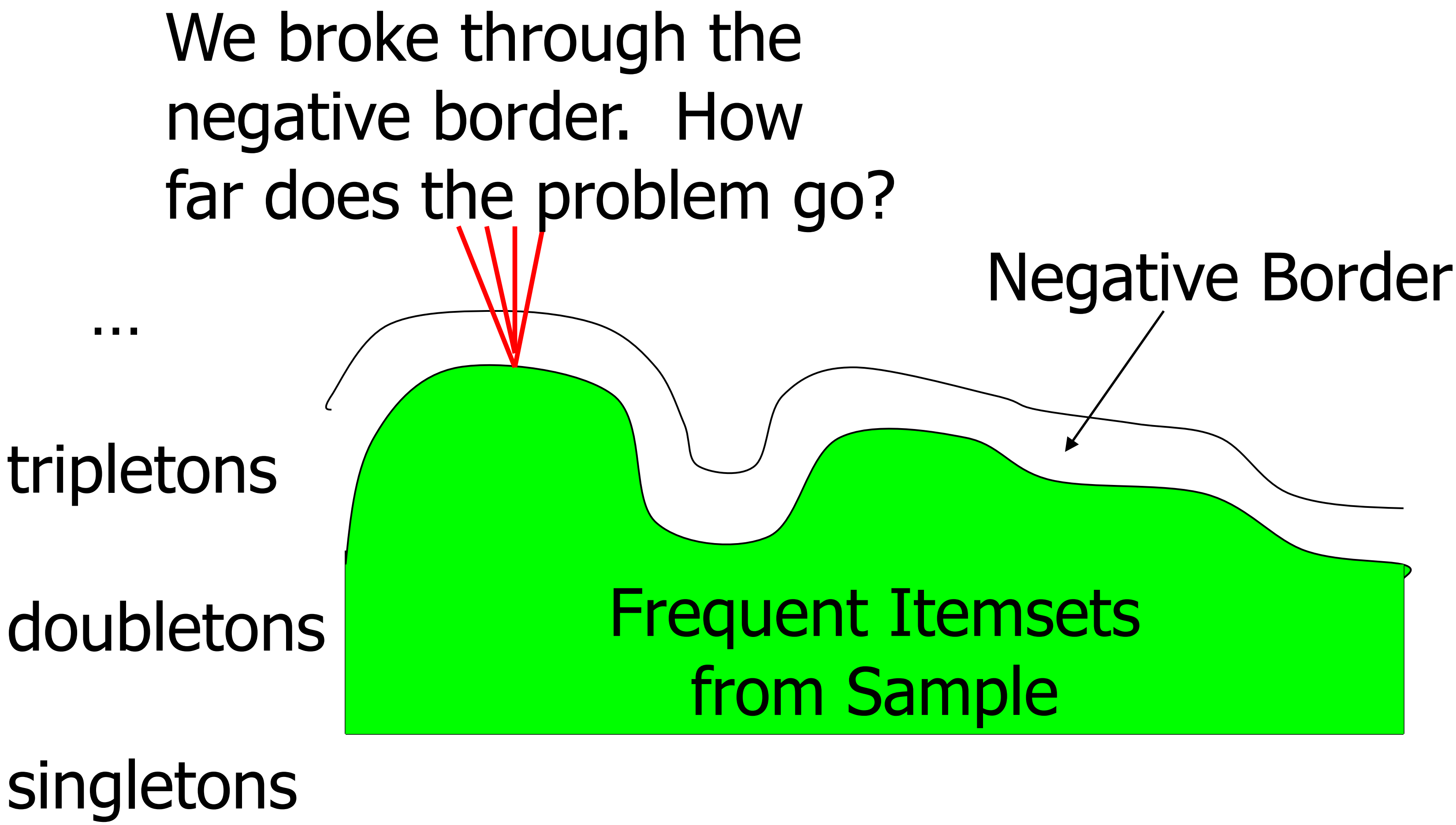
Toivonen's Algorithm Continued

- In a second pass, count all candidate frequent itemsets from the first pass, and also count their negative border
- If no itemset from the negative border turns out to be frequent, then the candidates found to be frequent in the whole data are *exactly* the frequent itemsets

Toivonen’s Algorithm Continued

- What if we find that something in the negative border is actually frequent?
- We must start over again!
- Try to choose the support threshold so the probability of failure is low, while the number of itemsets checked on the second pass fits in main-memory.

If Something in the Negative Border is Frequent . . .



Theorem:

- If there is an itemset that is frequent in the whole, but not frequent in the sample, then there is a member of the negative border for the sample that is frequent in the whole.

Proof

- Suppose not; i.e.;
  1. There is an itemset S frequent in the whole but not frequent in the sample, and
  2. Nothing in negative border is frequent in the whole
- Let T be a smallest subset of S that is not frequent in the sample
- T is frequent in the whole (S is frequent + monotonicity)
- T is in the negative border (else not "smallest")

Outline

- Homework 3 review
- Association rule mining
  - Introduction and definitions
  - Naïve algorithm
  - Apriori
  - PCY
  - Limiting disk I/O
  - Presenting results, other metrics
- Take away messages from class

Compacting the Output

1. *Maximal Frequent itemsets* : no immediate superset is frequent

2. *Closed itemsets* : no immediate superset has the same count (> 0).
   - Stores not only frequent information, but exact counts

Example: Maximal/Closed

	Count	Maximal (s=3)	Closed
A	4	No	No
B	5	No	Yes
C	3	No	No
AB	4	Yes	Yes
AC	2	No	No
BC	3	Yes	Yes
ABC	2	No	Yes

Frequent, but superset BC also frequent

Frequent, and its only superset, ABC, not freq

Superset BC has same count

Its only super-set, ABC, has smaller count

Interestingness Measurements

- Two popular objective measurements:
  - *support*
  - *confidence*

- Subjective measures: A rule (pattern) is interesting if it is:
  - *Unexpected* (surprising to the user)
  - *Actionable* (the user can do something with it)

Criticism of Support and Confidence

- Example: 5000 students
  - 3000 play basketball
  - 3750 eat cereal
  - 2000 both play basket ball and eat cereal
- *play basketball* $\Rightarrow$ *eat cereal* [40%, 66.7%]
  - misleading as the overall percentage of students eating cereal is 75% which is higher than 66.7%
- *play basketball* $\Rightarrow$ *not eat cereal* [20%, 33.3%]
  - More accurate, but lower support and confidence

	basketball	not basketball	sum(row)
cereal	2000	1750	3750
not cereal	1000	250	1250
sum(col.)	3000	2000	5000

Statistical Measures

- $P(S \wedge B) = P(S) \times P(B)$ => Statistical independence
- $P(S \wedge B) > P(S) \times P(B)$ => Positively correlated
- $P(S \wedge B) < P(S) \times P(B)$ => Negatively correlated
- $\text{Lift}(A \Rightarrow B) = \dfrac{P(B \mid A)}{P(B)}$

Example: Lift

	Coffee	$\overline{\text{Coffee}}$	
Tea	15	5	20
$\overline{\text{Tea}}$	75	5	80
	90	10	100

Association Rule: Tea $\rightarrow$ Coffee

Confidence= P(Coffee|Tea) = 0.75

but P(Coffee) = 0.9

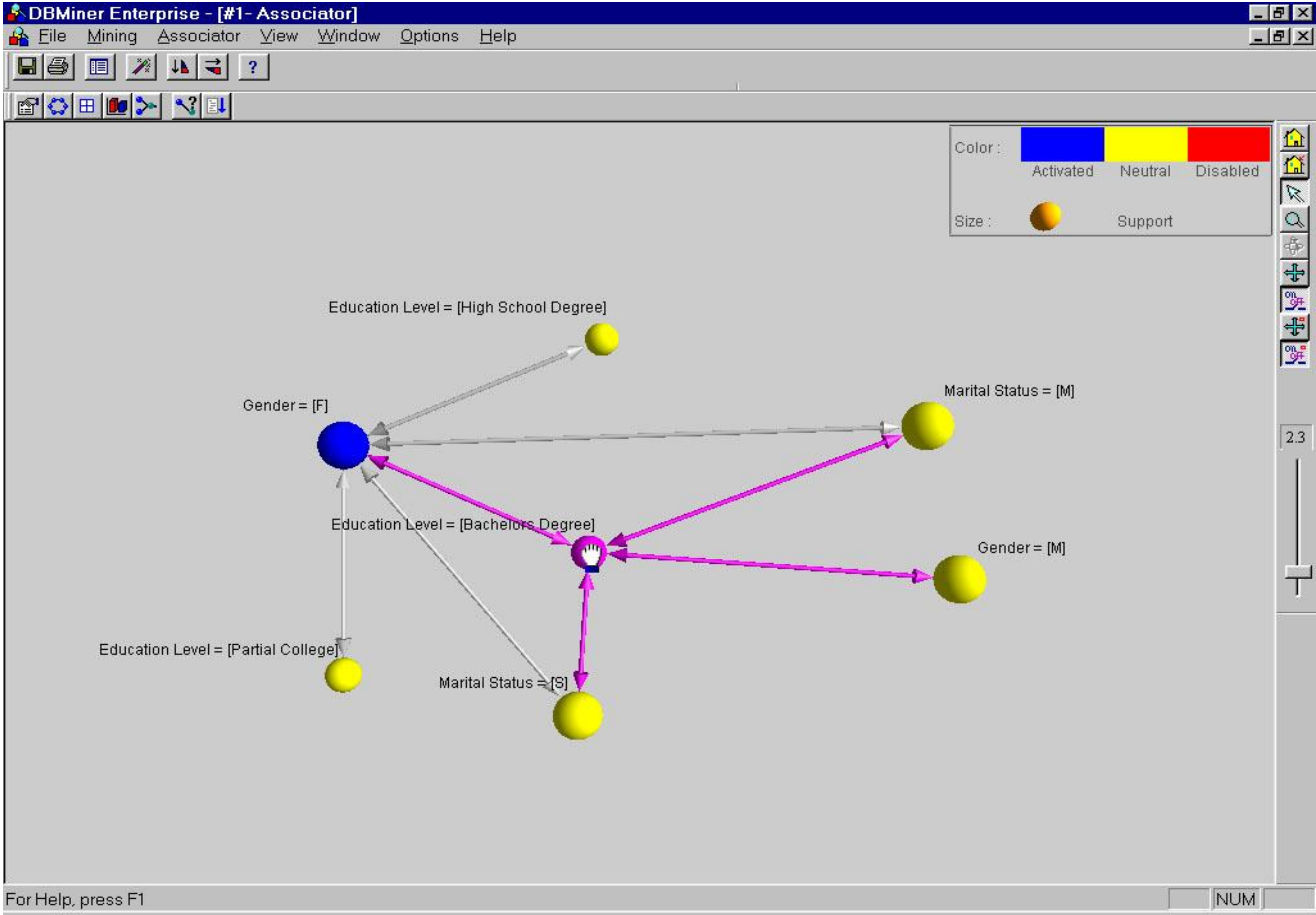
$\Rightarrow$ Lift = 0.75/0.9= 0.8333 (< 1, therefore is negatively associated)

Presentation of Association Rules (Table Form)

	Body	Implies	Head	Supp (%)	Conf (%)	F	G	H	I
1	cost(x) = '0.00~1000.00'	==>	revenue(x) = '0.00~500.00'	28.45	40.4				
2	cost(x) = '0.00~1000.00'	==>	revenue(x) = '500.00~1000.00'	20.46	29.05				
3	cost(x) = '0.00~1000.00'	==>	order_qty(x) = '0.00~100.00'	59.17	84.04				
4	cost(x) = '0.00~1000.00'	==>	revenue(x) = '1000.00~1500.00'	10.45	14.84				
5	cost(x) = '0.00~1000.00'	==>	region(x) = 'United States'	22.56	32.04				
6	cost(x) = '1000.00~2000.00'	==>	order_qty(x) = '0.00~100.00'	12.91	69.34				
7	order_qty(x) = '0.00~100.00'	==>	revenue(x) = '0.00~500.00'	28.45	34.54				
8	order_qty(x) = '0.00~100.00'	==>	cost(x) = '1000.00~2000.00'	12.91	15.67				
9	order_qty(x) = '0.00~100.00'	==>	region(x) = 'United States'	25.9	31.45				
10	order_qty(x) = '0.00~100.00'	==>	cost(x) = '0.00~1000.00'	59.17	71.86				
11	order_qty(x) = '0.00~100.00'	==>	product_line(x) = 'Tents'	13.52	16.42				
12	order_qty(x) = '0.00~100.00'	==>	revenue(x) = '500.00~1000.00'	19.67	23.88				
13	product_line(x) = 'Tents'	==>	order_qty(x) = '0.00~100.00'	13.52	98.72				
14	region(x) = 'United States'	==>	order_qty(x) = '0.00~100.00'	25.9	81.94				
15	region(x) = 'United States'	==>	cost(x) = '0.00~1000.00'	22.56	71.39				
16	revenue(x) = '0.00~500.00'	==>	cost(x) = '0.00~1000.00'	28.45	100				
17	revenue(x) = '0.00~500.00'	==>	order_qty(x) = '0.00~100.00'	28.45	100				
18	revenue(x) = '1000.00~1500.00'	==>	cost(x) = '0.00~1000.00'	10.45	96.75				
19	revenue(x) = '500.00~1000.00'	==>	cost(x) = '0.00~1000.00'	20.46	100				
20	revenue(x) = '500.00~1000.00'	==>	order_qty(x) = '0.00~100.00'	19.67	96.14				
21									
22									
23	cost(x) = '0.00~1000.00'	==>	revenue(x) = '0.00~500.00' AND order_qty(x) = '0.00~100.00'	28.45	40.4				
24	cost(x) = '0.00~1000.00'	==>	revenue(x) = '0.00~500.00' AND order_qty(x) = '0.00~100.00'	28.45	40.4				
25	cost(x) = '0.00~1000.00'	==>	revenue(x) = '500.00~1000.00' AND order_qty(x) = '0.00~100.00'	19.67	27.93				
26	cost(x) = '0.00~1000.00'	==>	revenue(x) = '500.00~1000.00' AND order_qty(x) = '0.00~100.00'	19.67	27.93				
27	cost(x) = '0.00~1000.00' AND order_qty(x) = '0.00~100.00'	==>	revenue(x) = '500.00~1000.00'	19.67	33.23				

Sheet1

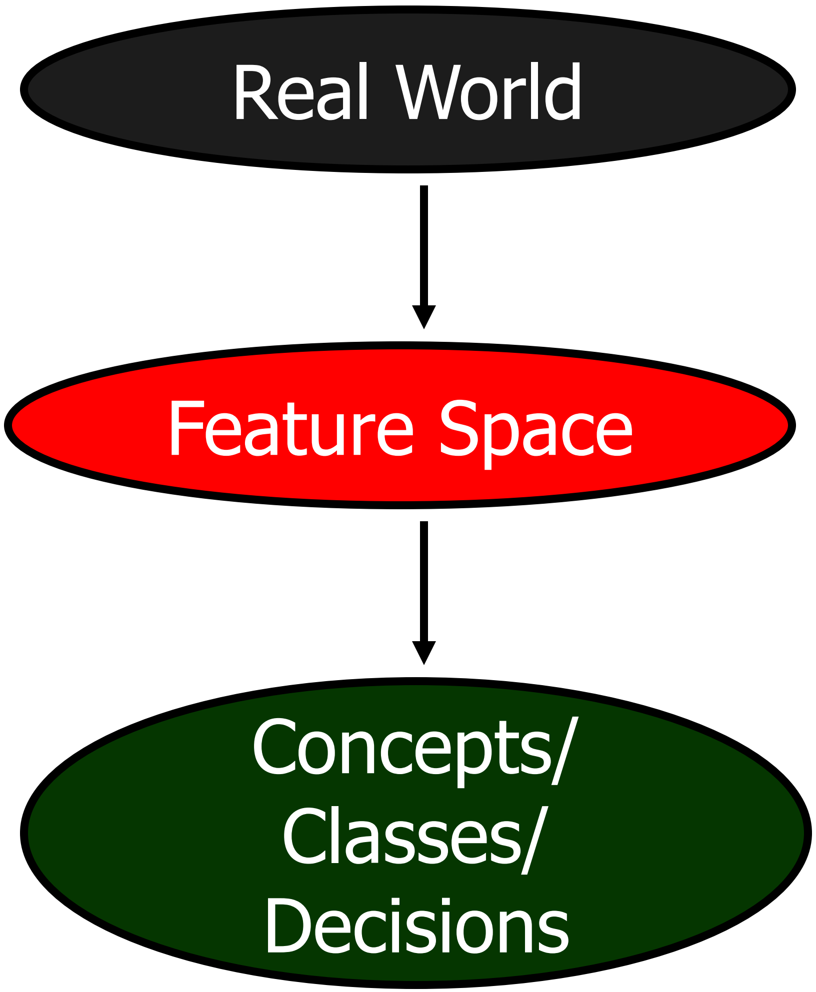
Visualization of Association Rule Using Rule Graph



Outline

- Homework 3 review
- Association rule mining
- Take away messages from class

Take Away: Feature Construction

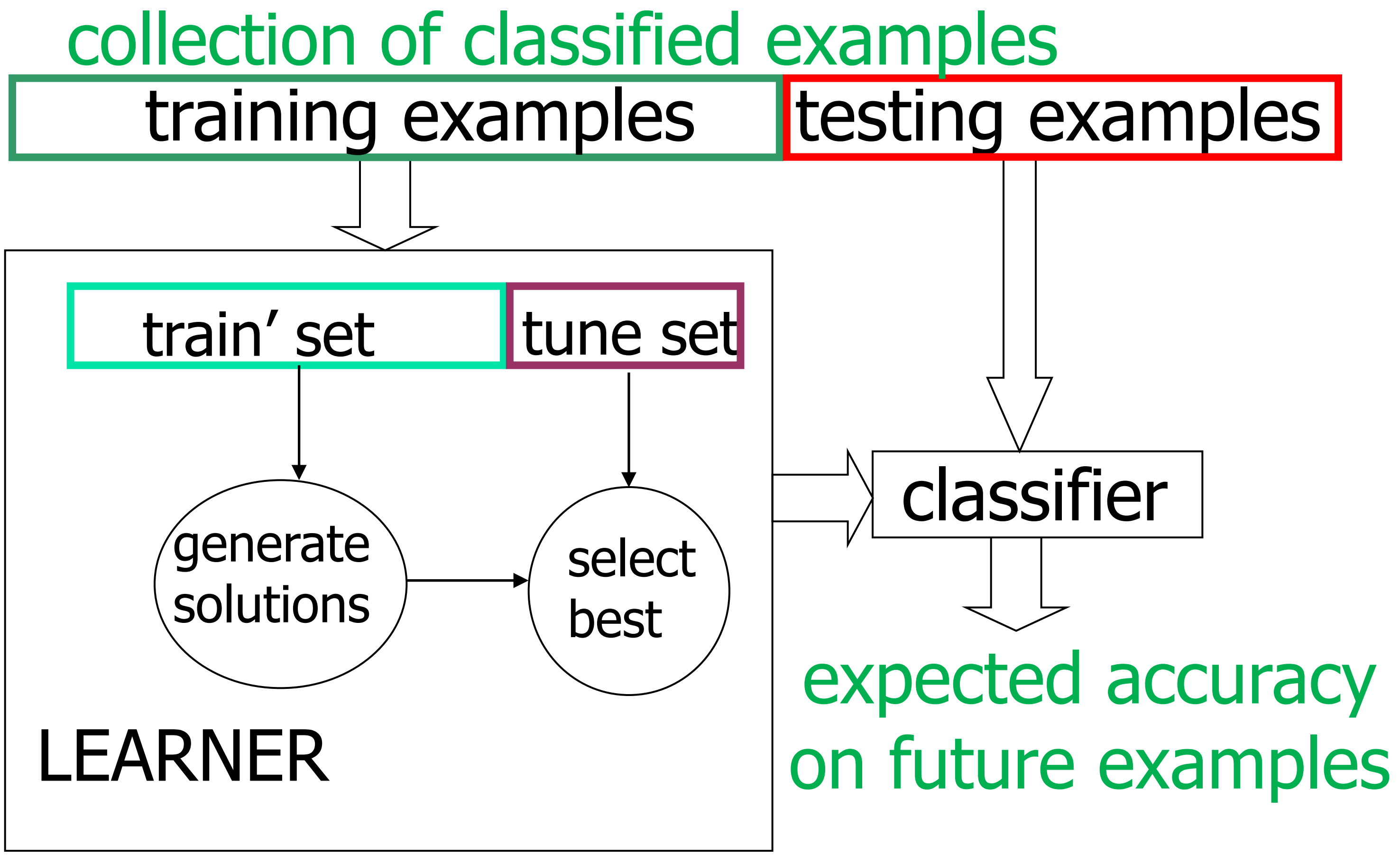


Feature construction is crucial!!!

Worth spending time on

Take Away: Empirical Evaluation

Use statistical techniques such as 10-fold cross validation to get meaningful results



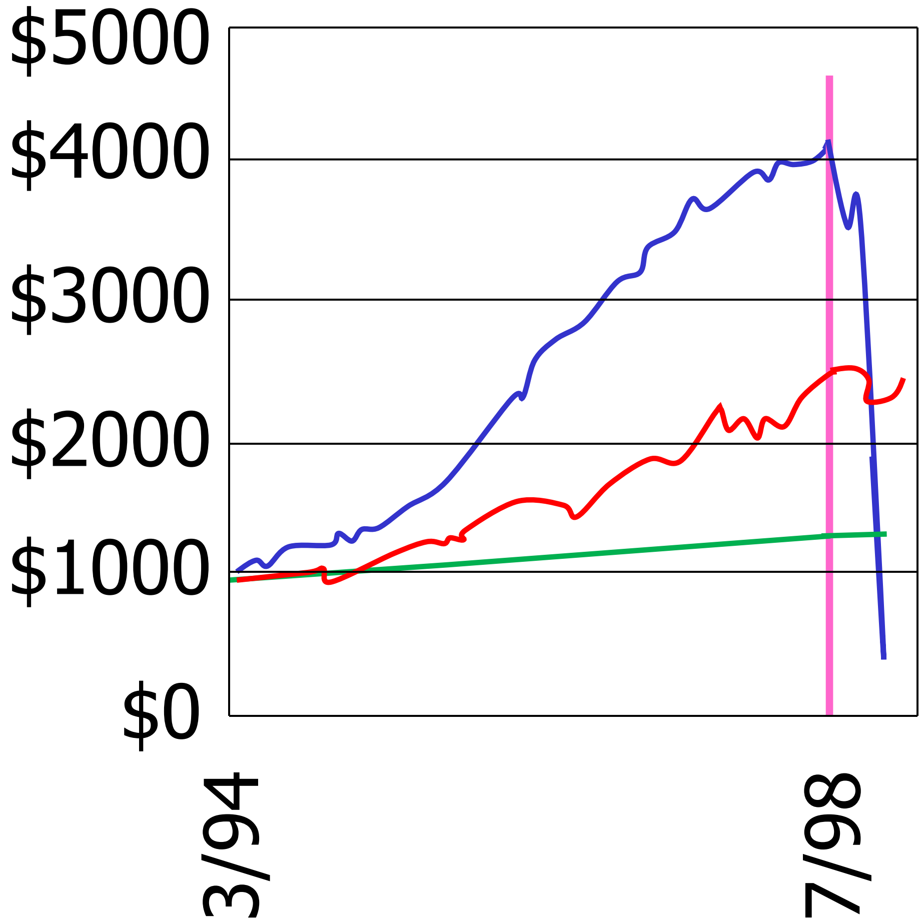
Take Away: Empirical Evaluation

- Often, an ML system has to choose when to stop learning, select among alternative answers, etc.
- One wants the model that produces the highest accuracy on **future** examples ("overfitting avoidance")
- It is a **"cheat"** to look at the **test** set while still learning
- Better method
  - Set aside part of the training set
  - Measure performance on this "tuning" data to estimate future performance for a given set of parameters
  - Use best parameter settings, train with **all** training data (except **test** set) to estimate future performance on **new** examples

Take Away: Empirical Evaluation

- Accuracy only can be misleading
- Look at alternative measures
  - True positive rate/recall
  - False positive rate
  - Precision
  - Area under the curve

Take Away: Be Wary of Assumptions



LTCM
DJ
30 T-Bill

Simplification:
Assumed investments were independent

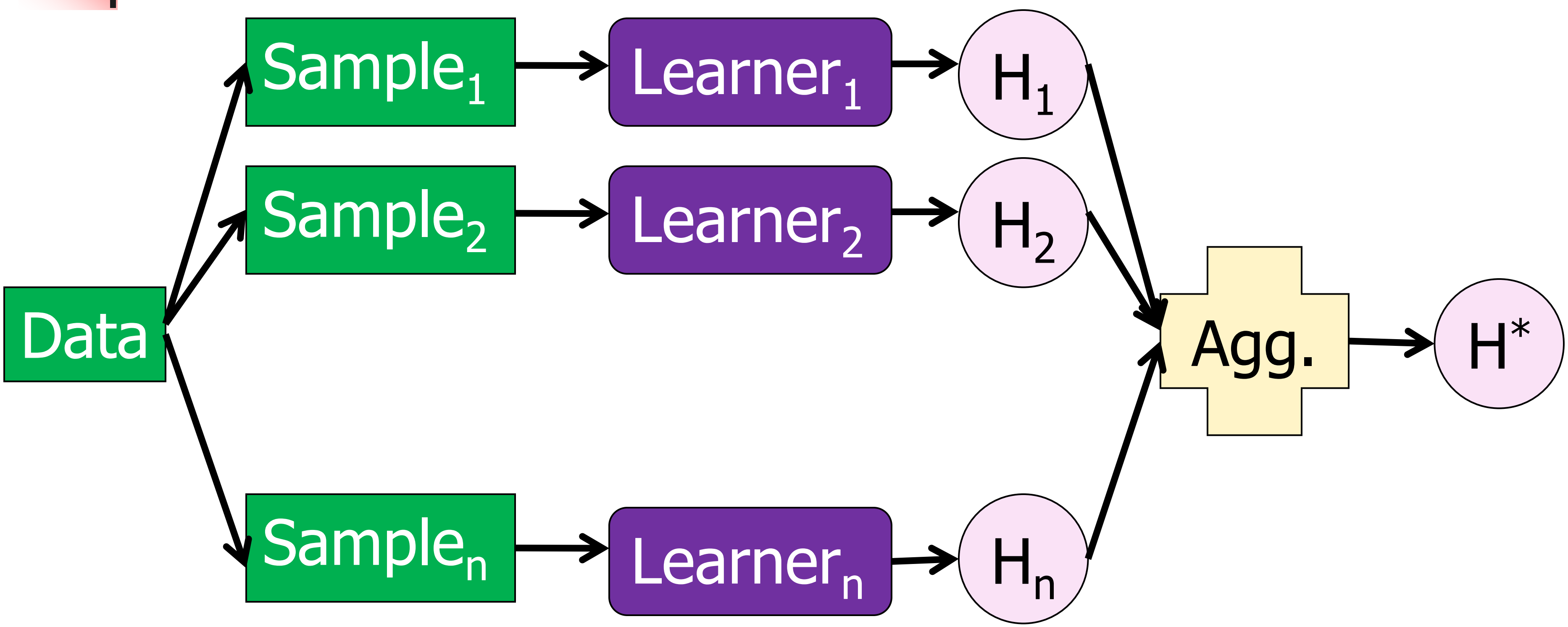
Reality:
All similar type of bet

Take Away: Simple Methods

- Simple approaches often work reasonable well in practice
  - 1-nn
  - Naïve Bayes
  - Perceptron

- Often worth trying tfirst

Take Away: Ensembles



1) Many classifiers often better than single classifier
2) Bagging/boosting are simple and very effective
3) Worth trying!

Summary

- Association rules: Efficient way to mine interesting information very large databases
  - Get probabilities
  - Don't require user guidance for interesting patterns
- Apriori algorithm and it's extensions allow the user to gather a good deal of information without too many passes through data

Questions?