ADB Lecture 5

- Data Mining
  - ⇒ Discover Patterns or rules from vast amounts of data. (new info)
  - ⇒ Discover interesting structure in data
  - ⇒ Automatically analyze & extract knowledge from data

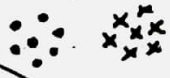
- Knowledge Discovery
  - Data Selection
    - Select Subset
  - Data Cleaning
    - Remove noise & outliers, unify units, new fields for analysis, ...
  - Data Mining
  - Evaluation
    - by Presenting Patterns to end users (visually)

- Goals of data mining & Knowledge Discovery

P I C O

- Prediction (Future behavior)
- Identification (e.g. intruders)
- Classification (Partition data)
- Optimization (max $ under..)

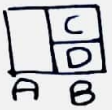
Types of Discovered Knowledge



Clustering
- Partition data into sets grouping similar items

Association Rules
- Correlate items with values or other items

(e.g. buy milk → buy eggs)



Classification Hierarchies
- goal is to create a hierarchy of classes

Time Series
- Detect similarities between items over time

Sequential Patterns
- Predict based on sequence of events (e.g. symptoms)

1. Association Rules

⇒ Can help businesses much
- They have big datasets of transactions (transaction id + date + list of items)
- Use it to make associations such as
  98% customers who buy X, also buy Y
  → helpful for marketing strategies, store layout, add-on sales, ...

Notation
→ $I = \{ i_1, i_2, \dots, i_m \}$ Set of all possible items
→ $D = \{ t_1, t_2, \dots, t_n \}$ dataset of n transactions
- each transaction has a subset of the items in I ($t_j \subseteq I$) & has an ID (TID)

- An association rule conveys correlation between two item sets X, Y (i.e. $X, Y \subset I$)
  * Denoted by $X \rightarrow Y$ where X, Y are disjoint: $X \cap Y = \emptyset$

e.g. {milk, bread} → e.g. {eggs, yoghurt}

- The strength of the association is measured by
  - Confidence

$$C = \frac{\#\text{Tr. with } X \& Y}{\#\text{Tr. with } X}$$

$P(Y_{int} | X_{int})$

  - Support

$$S = \frac{\#\text{Tr. with } X \& Y}{n}$$

$P(X, Y_{int})$

- n is #transactions
- also applies for one set (eg. SUPP(X))

Given
- → Set of transactions D
- → minimum support (minsup)
- → minimum confidence (minconf)

Wanted
- → All association rules with support ≥ minsup and confidence ≥ minconf

- This can be broken down into two subproblems

1. Find "large" itemsets i.e., those with support exceeding minsup
   # Apriori Algo. used for this.

2. Find Association Rules $X \to Y$
   - $X \cup Y$ must have high support (beyond minSup)
   - $X \to Y$ must have high confidence (beyond minConf)

* Association rules have nothing to do with functional dependencies
  → Functional dependency $X \to Y$ for a relation R is a constraint that two tuples with same value of X, must have same value for Y
  - implies
    $XZ \to Y$ (also holds for each value of XZ) (concat.)
  - also
    $X \to Y$, $Y \to Z$ implies $X \to Z$

  ⇒ Neither of these (concatenation or association) hold for association rules
  - e.g. If $X \to Y$ has minSup, $XA \to Y$ may not necessarily do as well (e.g. there's only one transaction {AX}, if it also has {Y} then Supp is $\frac{1}{n}$)
  - If $X \to A$, $A \to Z$ have minConf., $X \to Z$ may not (e.g. only have {XA}, {AZ} then Conf for XZ is zero)

- APriori Algorithm

⇒ Basic idea: {A,B}

If ItemSet AB doesn't surpass minSUP then so will not ABC, ABCD,... (APriori Principle)

- A K-ItemSet means an ItemSet of length K
- A large ItemSet means an itemset with SUPP exceeding minSUPP

- The algorithm uses two (main) data structures

L_K	C_K
has all ItemSets of length K exceeding minimum support (i.e. large)	has all ItemSets of length K that may be large

- We assume each row in dataset is <tid, i1, i2, ...>
  (tid: transaction; i1, i2, ...: items in it)
  → is okay if relation is normalized <tid, item>

- Example

Tid	items
10	A, C, D
20	B, C, E
30	A, B, C, E
40	B, E

, $MinSup_{Count} = 2$ (2/4)

C_1 =

ItemSet	SupCount
A	2
B	3
C	3
D	1
E	3

- Will ignore this step later (directly L_1)

|| no. of transactions with B

- C_1 has all possible 1-items
- Violation $K_{min\,sup} = 2$

L_1 =

A	2
B	3
C	3
E	3

- L_K is C_K but keeps only large item sets → hence, D discarded

C_2 =

A,B	1
A,C	2
A,E	1
B,C	2
B,E	3
C,E	2

- To get candidate 2-item sets, do $L_1 \bowtie L_1$ (keeping only 2-item sets uniquely)

L_2 =

A,C	2
B,C	2
B,E	3
C,E	2

- Discarded small items {A,B}, {A,E}

- Now you think we should do $L_2 \bowtie L_2$ to get C_3

→ Not exactly. Need to carefully define $\bowtie$ first ($K > 2$)

- $L_{K-1} \bowtie L_{K-1} \Rightarrow C_K$

length = K-1

→ Do the join by unioning any two itemsets ① that differ in only one item (same K-2 items)

e.g. A, B, C → A, B, C, D
A, B, D

→ justification: if they differ in two items or more we don't get a K-itemset (A,B,C / A,D,E → A,B,C,D,E ; length ≠ 4)

& if they don't differ at all, we get a K-1 itemset

- hence, this will give us the itemsets of length K exactly (what we need)

→ It turns out to be sufficient to ②
- union two K-1 itemsets only if all elements are the same & they differ in the last element only.

② or ①

hence,

A, C
B, C
B, E
C, E
L_2 ⋈ L_2

→ C_3 =

itemset	sup
B, C, E	2

- to ignore duplicates (red arrow), can add condition that last element in 1st operand must be smaller than last element in 2nd operand (only black arrow lives)

- Clear as well that $L_3 =$

ItemSet	Sup
B, C, E	2

→ Nothing satisfies join condition for C_4 (i.e. $C_4 = \phi$) and hence algorithm terminates

→ large item sets are: 4 item sets from L1, 4 item sets from L2, 1 itemset of L3

- In this example, we never had to do the Pruning Step (why)
  → And what's that?

→ Suppose $L_2 =$

A, B	4
A, C	4
A, E	2
B, C	4
B, D	2
B, E	2

then $C_3 =$

A, B, C
A, B, E
A, C, E
B, C, D
B, C, E
B, D, E

} Do we have to count the support of each or Can we do better

→ $C_3 = Prune(C_3)$ ← use the APriori Principle

- Since {E, C} wasn't in L2 → it didn't have minSup → {B, C, E} and {A, C, E} won't have minSup

- Likewise {E,D} didn't have minsup → {B,D,E} won't

  {C,D} didn't have minsup → {B,C,D} won't

$C_3 =$ A,B,C
A,B,E
} We only need to count minsup for these (then filter in L3)

→ In general, the Prune Step takes result from join and keeps only itemsets where all (K-1)-itemset subsets are in L_{K-1}

e.g. A,B,C → A,C (A,B, B,C) ; A,B,E → A,E (A,B, B,E) } all in L2

- Thus, our algorithm is

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

For ($K=2$; $L_{K-1} \neq \Phi$; K++) // until L_{K-1} is empty

C_K = aPriori_gen(L_{K-1}) // generate K-itemsets $L_{K-1} \rightarrow C_K$

For (t in transactions) // For each candid. itemset in t (i.e.)
For (C in Subset(C_K, t))
C.Count++;

$L_K = \{c \in C_K \mid c.count \geq minsup\}$ // include only those ≥ minsup

return $L_1 \cup L_2 \cup \ldots \cup L_{K-2}$

- APriGen

1. Join Step ($L_{k-1} \bowtie L_{k-1}$)

* go up 3 Pages (things should make sense)

Insert into C_k
Select $p.i_1, p.i_2, \ldots, p.i_{k-1}, q.i_{k-1}$ } ABC, ABD → ABCD
From L_{k-1} as p, L_{k-1} as q
where $p.i_1 = q.i_1, \ldots, p.i_{k-2} = q.i_{k-2}, p.i_{k-1} < q.i_{k-1}$

↳ respect alph. order as well (<)

2. Prune Step

* go up 2 Pages

For each c in C_k
  For each K-1 subset of c
    If subset not in L_{k-1}
      delete c from C_k

return C_k

- APriori TID

→ just like APriori but encodes itemsets
→ later passes, smaller # itemsets, smaller encodings & faster reads
- only need database to count supp in 1st pass ①

- involves other modifications to make ① hold ($\bar{C}_k$)

- Problems with APriori (why AprioriTid or Hybrid)

⇒ APriori goes over entire dataSet every Pass ← calc. support (L_k)

→ #Transactions is so big (millions added Per hour)

→ Scaning db is expensive

- A Possible improvement is to only check transactions that can Possibly Contain the K-itemSet

* Now we've Solved the 1st Part of the SubProb.
  → generating all large item sets

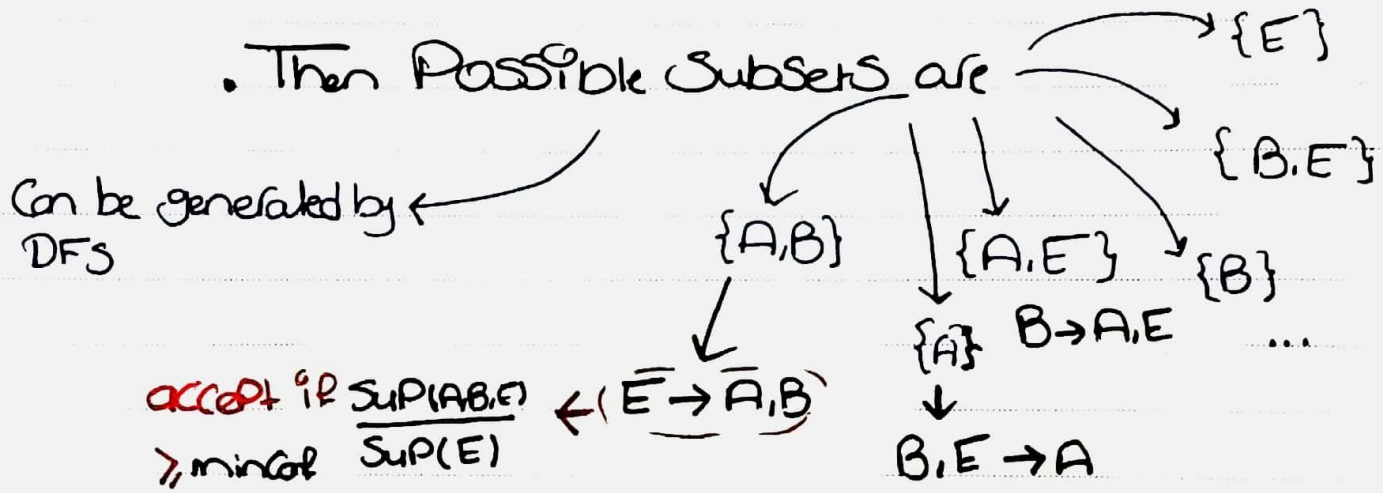
  → hence, time to discover rules (associations)

For every large itemSet P
  For every non-empty subset of P q
    if (SupP(P)/SupP(P-q))

      Form rule P-q ⟶ q

- Example
  → Let item set P be {A, B, E}

- Then Possible Subsets are



→ In general, for K-ItemSet, we have $2^K - 2$ Possible Subsets. (ignoring ∅ & the ItemSet itself)
- Means we'll have to do that much minConf Checks for each ItemSet
  → by the Apriori Principle, have all needed supports (no need for database scanning)

» To Optimize:

Observe that
If $(P - q_{sub}) \rightarrow q_{sub}$ doesn't hold then so will not $(P - q) \rightarrow q$

So we can get the rules smartly by
- Starting with 1-item consequents

Correspond to {A}, {B}, {E} resp.

$B, E \rightarrow A$
$A, E \rightarrow B$
$A, B \rightarrow E$

← If this fails, any rule with Consequent A will also fail (i.e. ignore any q with A in it)
- No need to check $B \rightarrow A, E$ or $E \rightarrow A, B$

Association Rule Complications

- Item sets of high cardinality
  - too many rules
- transactions may depend on variables
  - e.g. season
- can exist on multiple dimensions
  - e.g. time
- hard to find we associations $X \rightarrow Y$
- affected by data quality
  - conflict, missing, ...

2. Classification

- Uses a training set of Pre-classified data (supervised learning)
- Produces a decision tree or set of rules to classify future items

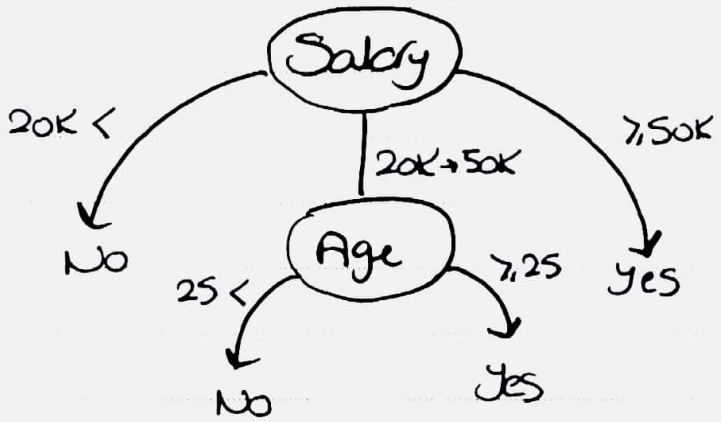
• NN Lec. 7

Sample Training data

RID	Married	Salary	acct_balance	age	loan
1	no	≥ 50K	< 5K	≥ 25	yes

• Can be actual nos

Class to Predict



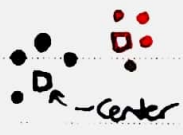
Example decision tree

- involves 4 rules you can extract.

3. Clustering

→ Goal is to place records in groups where the records in a group are highly similar (unsupervised learning)

- K-means
  - → Start with initial cluster centers
    Repeat: // until no change
    Assign each example to closest center
    Recalculate centers as mean of cluster



4. Sequential Pattern Analysis

- Assume transactions ordered by time of purchase
- e.g. {A,B,C}, {B,E}, {A,D,F} ← Sequence of item sets

→ Support of sequence is % of sequences of which it is a subsequence
- Want sequences that exceed minsup
  e.g. S1, S2, S3 exceeding minsup means of customers by S1, they are likely to buy S2 then S3

5. Time Series Analysis

- Time Series = Sequence of events
  → e.g. Closing price of stock occurs every week day (Sun → Thurs)
  - Sequence of values for a stock → time series
  - Can use to predict future trend for stock.

- Also helpful for temporal data management (e.g. find the best quarter for the stock)