*** Advanced Machine Learning ***
Association Rule Learning

M. Sc. Daniel Wehner

SAP SE / DHBW Mannheim

Summer term 2020

DHBW
Duale Hochschule
Baden-Württemberg

Agenda for this Unit

Introduction

What is Association Rule Mining?

- Association rule mining belongs to the category of unsupervised learning.
- Association rules describe frequent co-occurrences in the data (**not necessarily causality!**)
- Examples:
  - Market basket analysis (*Which products are frequently bought together? E. g. Amazon*)
  - Course schedule planning (*Which courses are often attended together?*)
  - Other use cases: Marketing promotions, inventory management, customer relationship management (CRM)
- The general form of a rule is given by:

$$\overbrace{\{a_1, a_2, \ldots, a_n\}}^{\textbf{Antecedent}} \rightarrow \overbrace{\{b_1, b_2, \ldots, b_m\}}^{\textbf{Consequent}} \tag{1}$$

- Example: $\{bread, cheese\} \rightarrow \{wine\}$

Figure 1: Famous example from Amazon

Important Terminology

- Suppose $\mathcal{I}$ is a set of unique items which we have in our portfolio $\mathcal{T} = \{t_1, t_2, \ldots, t_n\}$ is a list of transactions (what customers bought).
- Each transaction $t_i \in \mathcal{T}$ is an element of $\mathfrak{P}(\mathcal{I})$, the power set of $\mathcal{I}$. (**What is a power set?**)
- Example:

Id	Transactions
1	$\{beer, chips, wine\}$
2	$\{beer, chips\}$
3	$\{pizza, wine\}$
4	$\{chips, pizza\}$

Id	*beer*	*chips*	*pizza*	*wine*
1	1	1	0	1
2	1	1	0	0
3	0	0	1	1
4	0	1	1	0

Figure 2: Left: List of transactions (raw), right: List of transactions in binary form

Simplification: We ignore quantities and prices of the items sold.

Item sets

- A collection of k items is called k-item set.
- Example: $\{pizza, wine\}$ is a 2-item set.
- The number of items contained in a transaction t_i is sometimes referred to as the **transaction width** $w(t_i) = |t_i|$.
- An important property of an item set X is the **support count $\boldsymbol{\sigma}$**:

$$\sigma(X) = |\{t_i | X \subseteq t_i \wedge t_i \in \mathcal{T}\}| \tag{2}$$

- **What does the support count tell us?** $\sigma(X)$ refers to the number of transactions X occurs in.

Quality Measures

- *Question:* How to measure the quality of an association rule?
- **Support:**
  - Proportion of examples for which head and body are true.
  - Example $A \to B$: How many customers bought A and B together?

$$\text{support}(A \to B) = \text{support}(A \cup B) = \frac{\sigma(A \cup B)}{n} \tag{3}$$

- **Confidence:**
  - Proportion of examples for which the head is true among those for which the body is true.
  - Example: If customers bought A, how likely are they to also buy B?

$$\text{confidence}(A \to B) = \frac{\text{support}(A \cup B)}{\text{support}(A)} = \frac{\sigma(A \cup B)}{\sigma(A)} \tag{4}$$

- Support: There is a huge number of possible rules, but not all of them are interesting.
  $\Rightarrow$ **Prune (remove) rules with low support.**

- Confidence: The higher the confidence the more reliable is the rule.

- Example:
  - $R = \{bread, cheese\} \rightarrow \{wine\}$
  - $\text{support}(R) = 0.01$ and $\text{confidence}(R) = 0.8$
  - 80 % of all customers who bought bread and cheese also bought red wine.
  - However, only 1 % of the customers bought all three items together.

Apriori

Learning Problem

- The **Apriori algorithm** can be used to find association rules.
- The learning problem can be summarized as follows:
  Given a set of transactions $\mathcal{T}$*, find all rules having support* $\geqslant s_{min}$ *and confidence* $\geqslant c_{min}$*, where* s_{min} *and* c_{min} *are thresholds.*
- Obviously, mining all possible rules is super expensive.

$$|\text{rules}| = 3^d - 2^{d+1} + 1 \qquad \text{where} \qquad d \equiv |\mathcal{T}| \tag{5}$$

- Also, rules can be spurious (i. e. patterns may occur by chance and are not systematic).

We have to avoid considering all possible rules! $\Rightarrow$ Employ early pruning.

Early Pruning

- The goal is to generate rules which have high support and high confidence.
- Observation: If an item set is infrequent (does not have sufficient support), calculating the confidence can be omitted.
- As a consequence, all rules which can be generated from this item set do not have to be considered anymore.
- Example for the item set $\{beer, diapers, milk\}$:
  - The rules derived from this item set are given by:
  - If we know this item set to be infrequent, we can prune all these rules.
  - There is no need to calculate the confidence for these rules (**decoupling of support and confidence**)

$$\{beer, diapers\} \rightarrow \{milk\} \qquad \{beer, milk\} \rightarrow \{diapers\}$$
$$\{diapers, milk\} \rightarrow \{beer\} \qquad \{beer\} \rightarrow \{diapers, milk\}$$
$$\{milk\} \rightarrow \{beer, diapers\} \qquad \{diapers\} \rightarrow \{beer, milk\}$$

Apriori Algorithm

- The overall algorithm consists of two major steps:

  1. **Frequent item set generation:**
     Find all item sets which have sufficient support (satisfy the support constraint).
  2. **Rule generation:**
     Extract highly confident rules which satisfy the confidence constraint.

- In the following we will have a closer look at these two steps.

Step 1) Frequent item set generation

- It is possible to enumerate all possible item sets with a lattice.
- A brute force approach could calculate the support for each candidate set and rank them by the result.
- **Problem:** The number of candidate sets grows exponentially with $|\mathcal{I}|$: $2^{|\mathcal{I}|} - 1$ (excluding empty set).
- Example: For $\mathcal{I} = \{a, b, c, d, e\}$ we have 31 possible candidates.
- Therefore, the candidate sets should be generated more efficiently.
- We can make use of the **anti-monotonicity** of the support:
  If an item set is frequent, then all of its subsets must be frequent as well. Also, if an item set is infrequent, then all its supersets must be infrequent too.
- Adding a condition can never increase the support of a rule:

$$A \subseteq B \Longrightarrow \text{support}(A) \geqslant \text{support}(B) \tag{6}$$

- **An item set can only be frequent, if all its subsets are frequent and all supersets of an infrequent item set are also infrequent.**

1. $k \leftarrow 1$
2. $C_1 \leftarrow \mathcal{I}$
3. **while** $C_k \neq \emptyset$ **do**
   - $\triangleright$ $S_k \leftarrow C_k \backslash \{\text{all infrequent item sets in } C_k\}$
   - $\triangleright$ $C_{k+1} \leftarrow$ all sets with $k+1$ elements which can be formed by uniting two item sets in S_k
   - $\triangleright$ $C_{k+1} \leftarrow C_{k+1} \backslash \{\text{item sets, where not all subsets of size } k \text{ are in } S_k\}$
   - $\triangleright$ $S \leftarrow S \cup S_k$
   - $\triangleright$ $k \leftarrow k + 1$
4. **return** S

The algorithm leaves it open how the candidate set C_{k+1} is generated. How can this be done efficiently?

- Requirements for efficient candidate generation:
  - We have to avoid producing too many candidates.
  - At the same time we have to ensure that all frequent item sets are found (**completeness**)
  - We don't want to produce duplicates (**efficiency**)
- The Apriori algorithm uses the so-called $S_{k-1} \times S_{k-1}$ method:
  - Merge a pair of $(k-1)$-item sets only if their first $k-2$ items are identical.

$$A = \{a_1, a_2, \dots, a_{k-1}\} \qquad B = \{b_1, b_2, \dots, b_{k-1}\} \tag{7}$$

  - Merge A and B, if $a_j = b_j \ (j = 1, 2, \dots, k-2) \wedge a_{k-1} \neq b_{k-1}$
  - Example:
    - ▷ $A = \{bread, pizza, milk\}$, $B = \{bread, pizza, diapers\}$
    - ▷ A and B are merged into $\{bread, pizza, milk, diapers\}$.
  - This method still requires pruning non-frequent item sets.