RS Module 5 Notes

Module 5.1: Context-Aware

Overview of Context-Aware RS

The majority of existing approaches to recommender systems focus on recommend- ing the most relevant items to individual users and do not take into consideration any contextual information, such as time, place and the company of other people (e.g., for watching movies or dining out). In other words, traditionally recommender sys- tems deal with applications having only two types of entities, users and items, and do not put them into a context when providing recommendations.

Context-aware recommender systems (CARS) extend traditional recommendation systems by incorporating contextual information to provide more personalized and accurate recommendations. Context refers to any information beyond the traditional user-item interaction, such as time, location, mood, device type, or even social environment. Incorporating context improves the relevance of recommendations, especially in dynamic and diverse environments.

Key Concepts in Context-Aware Recommendation Systems

Definition of Context:

Context refers to any additional information that can influence user preferences or item relevance. Examples include:

1. **Temporal Context**: Time of the day, season, or event.
2. **Spatial Context**: User's location (e.g., at home, work, or traveling).
3. **Social Context**: Presence of friends or family during consumption.
4. **Device Context**: Desktop, mobile, or tablet device being used.

Why Use Context in Recommendations?

1. To increase personalization: Context helps in adapting recommendations based on real-time factors.
2. To improve accuracy: Context-aware models outperform traditional models in dynamic scenarios.
3. To solve cold-start problems: Context can act as a substitute when user-item interaction data is sparse.

Obtaining Contextual Information

1. **Explicitly**: by directly approaching relevant people and other sources of contextual information and explicitly gathering this information either by ask- ing direct questions or eliciting this information through other means. For ex- ample, a website may obtain contextual information by asking a person to fill out a web form or to answer some specific questions before providing access to certain web pages.
2. **Implicitly**: from the data or the environment, such as a change in location of the user detected by a mobile telephone company. Alternatively, temporal contex- tual information can be implicitly obtained from the timestamp of a transaction. Nothing needs to be done in these cases in terms of interacting with the user or other sources of contextual information – the source of the implicit contextual information is accessed directly and the data is extracted from it.
3. **Inferring**: the context using statistical or data mining methods. For example, the household identity of a person flipping the TV channels (husband, wife, son, daughter, etc.) may not be explicitly known to a cable TV company; but it can be inferred with reasonable accuracy by observing the TV programs watched 7 Context-Aware Recommender Systems 229 and the channels visited using various data mining methods. In order to in- fer this contextual information, it is necessary to build a predictive model (i.e., a classifier) and train it on the appropriate data. The success of inferring this contextual information depends very significantly on the quality of such classifier, and it also varies considerably across different applications. For example, it was demonstrated in [54] that various types of contextual information can be inferred with a reasonably high degree of accuracy in certain applications and using certain data mining methods, such as Na¨ıve Bayes classifiers and Bayesian Networks.
4. Finally, the contextual information can be “hidden” in the data in some latent form, and we can use it implicitly to better estimate the unknown ratings without explicitly knowing this contextual information.

Approaches to Incorporating Context

Context-aware recommendation process that is based on contextual user preference elicitation and estimation can take one of the three forms, based on which of the three components the context is used in, as shown below:

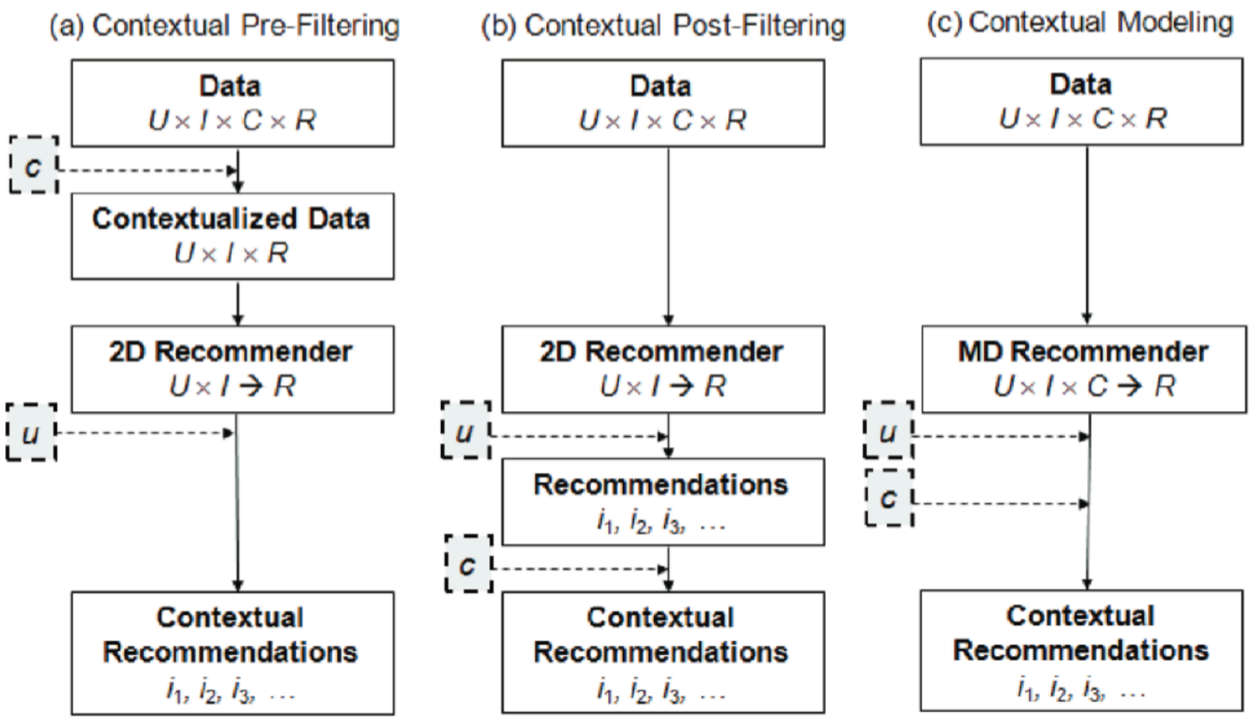
(a) Contextual Pre-Filtering
Data
$U \times I \times C \times R$
c
Contextualized Data
$U \times I \times R$
2D Recommender
$U \times I \rightarrow R$
u
Contextual Recommendations
$i_1, i_2, i_3, \ldots$
(b) Contextual Post-Filtering
Data
$U \times I \times C \times R$
2D Recommender
$U \times I \rightarrow R$
u
Recommendations
$i_1, i_2, i_3, \ldots$
c
Contextual Recommendations
$i_1, i_2, i_3, \ldots$
(c) Contextual Modeling
Data
$U \times I \times C \times R$
MD Recommender
$U \times I \times C \rightarrow R$
u
c
Contextual Recommendations
$i_1, i_2, i_3, \ldots$

Fig. 7.4: Paradigms for incorporating context in recommender systems.

1. **Contextual Pre-Filtering (contextualization of recommendation input)**
    - Context is used to filter out irrelevant data before applying a recommendation algorithm.
    - In this recommendation paradigm, contextual information drives data selection or data construction for that specific context. In other words, information about the current context c is used for selecting or constructing the relevant set of data records (i.e., ratings). Then, ratings can be predicted using any traditional 2D recommender system on the selected data.
    - Example: In a restaurant recommendation system, only restaurants near the user's current location are considered.
2. **Contextual Post-Filtering (contextualization of recommendation output)**
    - A traditional RS model generates recommendations first, and context is applied later to rank or filter these recommendations.
    - In this recommendation paradigm, contextual information is initially ignored, and the ratings are predicted using any traditional 2D recommender system on the entire data. Then, the resulting set of recommendations is adjusted (contextualized) for each user using the contextual information.
    - Example: A general movie recommendation is re-ranked based on the user's current mood (e.g., "feel-good" movies for a happy mood).
3. **Contextual Modeling (contextualization of recommendation function)**
    - Context is directly integrated into the recommendation model as an additional dimension or feature.

- In this recommendation paradigm, contextual information is used directly in the modeling technique as part of rating estimation.
- Example: Collaborative filtering is extended by adding context as a third dimension in a tensor model (user × item × context).

Techniques for Context-Aware RS

1. **Contextual Multi-Armed Bandits**
   - In multi-armed bandit algorithms, context is included as features to decide the best recommendation option dynamically.
   - Example: A music app recommending playlists adjusts suggestions based on the time of day (e.g., energetic music in the morning).
2. **Reinforcement Learning (RL)**
   - RL algorithms, such as Q-Learning, use context as part of the state to learn optimal policies for making recommendations.
   - Example: An e-commerce site learns to recommend seasonal products (e.g., umbrellas in monsoon) using temporal context.
3. **Learning to Rank**
   - Context is treated as a feature in ranking models where algorithms predict the relevance score of items for a given user-context pair.
   - Example: A travel site ranks holiday packages based on user preferences, location, and travel season.
4. **Latent Factor Models with Context**
   - Context is incorporated into matrix factorization or tensor factorization models.
   - Example: Netflix could extend its matrix factorization model by adding the “watching time” as a contextual feature.

Challenges in Context-Aware RS

1. **Context Modeling**: Determining which context features are relevant and how to encode them in the model.
2. **Data Sparsity**: The addition of contextual dimensions increases the sparsity of the data matrix, making learning more challenging.
3. **Real-Time Adaptation**: Adapting recommendations in real-time as contextual factors change requires high computational efficiency.
4. **Cold-Start for Context**: When context data is missing or incomplete, the system may struggle to provide meaningful recommendations.

Detailed Examples of Context-Aware Techniques

Pre-Filtering Example

- A restaurant recommender system filters restaurants by considering only those within a 5 km radius (spatial context).
- The remaining restaurants are ranked using a collaborative filtering model.

Post-Filtering Example

- A streaming platform generates a list of movies using content-based filtering.
- Movies with genres matching the user's mood (context) are then promoted in the final list.

Reinforcement Learning Example

- An e-commerce system using Q-learning learns that users are more likely to purchase winter clothes during the cold season.
- The system prioritizes recommending jackets and boots when the temperature drops.

Tensor Factorization Example

- Netflix extends its user-item interaction matrix to a tensor by adding the "time of day" context.
- The tensor captures patterns like specific users watching comedies during the weekend.

Module 5.2: Constraint-Based

Constraint-Based Recommenders

Constraint-Based Recommender Systems (CBRS) use explicitly defined constraints to guide the recommendation process. Unlike collaborative filtering or content-based approaches, CBRS relies on a set of rules or conditions defined by domain experts or inferred from user input. These systems are particularly useful in domains where recommendations must adhere to strict requirements or limitations, such as travel planning, e-commerce, or real estate. More on CBR can be found in RS Module 4 Notes

Development of Recommender Knowledge Bases

The success of constraint-based recommender systems (CBRS) in commercial applications depends on efficient tools that support knowledge engineers and domain experts in creating and maintaining recommender knowledge bases. A key challenge arises due to the limited technical skills of domain experts, who provide the knowledge but rely on knowledge engineers for formalizing it into an executable system. Tools like *CWAdvisor* aim to address this knowledge acquisition bottleneck by empowering domain experts to autonomously develop and maintain knowledge bases through accessible, graphical interfaces.

Key principles implemented in CWAdvisor include:

1. **Rapid Prototyping**: Changes in properties, explanation texts, recommendation rules, or processes are immediately reflected in a prototype system. Templates are used to graphically define and convert properties into an executable application.
2. **Graphical Editing**: Domain experts can modify knowledge bases and recommendation logic graphically, eliminating the need for programming expertise. This functionality is very important to make knowledge acquisition environments more accessible to domain experts without a well-grounded technical education. This ensures a clear separation between application logic and implementation.
3. **Integrated Testing and Debugging**: Errors in recommendation rules and processes are automatically detected and reported during development. This reduces deployment risks, fosters trust in the system, and ensures reliable recommendations. Thus, knowledge bases are maintained in a structured way and not deployed in a productive environment until all test cases specified for the knowledge base are fulfilled.

User Guidance in Recommendation Processes

As constraint-based recommender systems operate on the basis of explicit statements about the current customer’s needs and wishes, the knowledge about these user requirements has to be made available to the system before recommendations can be made. The general options for such a requirements elicitation process in increasing order of implementation complexity include the following:

Techniques for User Guidance

1. **Session-independent customer profiles**: users enter their preferences and interests in their user profile by, for example, specifying their general areas of interest. This is a common approach in web portals or social networking platforms.
2. **Static fill-out forms per session**: customers fill out a static web-based form every time they use the recommender system. Such interfaces are easy to implement and web users are well-acquainted with such interfaces, which are often used on web shops search for items.
3. **Conversational recommendation dialogs**: the recommender system incrementally acquires the user’s preferences in an interactive dialog, based on, for example, “critiquing”, “wizard-like“ and form-based preference elicitation dialogs, natural-language interaction or a combination of these techniques.
4. **Critiquing**: A popular interaction style for knowledge-based recommender systems, which was first proposed in the context of Case-Based Reasoning (CBR) approaches to conversational recommendation. The idea is to present individual items (instances), for example, digital cameras or financial products, to the user who can then interactively give feedback in terms of critiques on individual features. A user might, for instance, ask for a financial product with a “shorter investment period” or a “lower risk”. This recommend-review-revise cycle is repeated until the desired item is found. Note that although this method was developed for CBR recommendation approaches, it can also be applied to constraint-based recommendation, as the critiques can be directly

translated into additional constraints that reflect the user's directional preferences on some feature. More can be found in RS Module 4 Notes.

5. **Personalized preference elicitation dialogs** Another form of acquiring the user's wishes and needs for a constraint-based recommender system is to rely on explicitly modeled and adaptive preference elicitation dialogs. Such dialog models can for instance be expressed using a dialog grammar [2] or by using finite-state automaton. In the later system, the end user is guided by a "virtual advisor" through a series of questions about the particular needs and requirements before a recommendation is displayed. In contrast to static fill-out forms, the set of questions is personalized, i.e., depending on the current situation and previous user answers, a different set of questions (probably also using a different technical or non-technical language) will be asked by the system.

Calculating Recommendations

Following our characterization of a recommendation task (see Definition 1), we will now discuss corresponding problem solving approaches which are listed below:

1. **Constraint Satisfaction**: Solutions for constraint satisfaction problems are calculated on the basis of search algorithms that use different combinations of backtracking and constraint propagation - the basic principle of both concepts will be explained in the following:
   - **Backtracking**: In each step, backtracking chooses a variable and assigns all the possible values to this variable. It checks the consistency of the assignment with the already existing assignments and defined set of constraints. If all the possible values of the current variable are inconsistent with the existing assignments and the constraints, the constraint solver backtracks which means that the previously instantiated variable is selected again. In the case that a consistent assignment has been identified, a recursive activation of the backtracking algorithm is performed and the next variable is selected.
   - **Constraint Propagation**: The major disadvantage of backtracking is "trashing" where parts of the search space are revisited although no solution exists in these parts. In order to make constraint solving more efficient, constraint propagation techniques have been introduced. These techniques try to modify an existing constraint satisfaction problem such that the search space can be reduced significantly. The methods try to create a state of local consistency that guarantees consistent instantiations among groups of variables. The mentioned modification steps turn an existing constraint satisfaction problem into an equivalent one.
2. **Conjunctive Database Queries**: Solutions to conjunctive queries are calculated on the basis of database queries that try to retrieve items which fulfill all of the defined customer requirements.
3. **Ranking Items**: Given a recommendation task, both constraint solvers and database engines try to identify a set of items that fulfill the given customer requirements.

Typically, we have to deal with situations where more than one item is part of a recommendation result. In such situations the items (products) in the result set have to be ranked. In both cases, we can apply the concepts of multi-attribute utility theory (MAUT) that helps to determine a ranking for each of the items in the result set. An alternative to the application of MAUT in combination with conjunctive queries are probabilistic databases which allow a direct specification of ranking criteria within a query.

Key Features of Constraint-Based Recommender Systems

1. **Explicit User Preferences**: CBRS requires users to specify their preferences explicitly, unlike collaborative filtering, which infers preferences from user behavior.
2. **Deterministic Results**: Recommendations are deterministic, meaning the same inputs always yield the same outputs.
3. **Flexibility with Complex Constraints**: CBRS can handle intricate and multi-dimensional constraints, making it suitable for complex domains like travel, e-commerce, or real estate.
4. **Domain Knowledge Integration**: These systems leverage domain knowledge encoded as constraints, ensuring recommendations align with real-world requirements.

Advantages & Disadvantages of CBRS

Advantages	Disadvantages
Recommendations align precisely with user requirements.	Relies heavily on accurate and complete user input.
Can handle complex constraints and multi-criteria scenarios.	Requires domain expertise to define constraints effectively.
Transparent and interpretable recommendations.	Limited scalability in domains with dynamic or large datasets.
Suitable for niche domains where preferences are clear.	Struggles with sparse or implicit preference data.

Summary

Constraint-Based Recommender Systems are a powerful approach for domains where user preferences and constraints play a critical role in decision-making. By leveraging explicit constraints and domain knowledge, these systems can deliver highly accurate and relevant recommendations. However, they require careful design of the knowledge base and inference mechanisms to ensure they meet user needs effectively. Combining CBRS with other approaches, such as hybrid recommenders, can address their limitations and broaden their applicability.