RS Module 4 Notes

Module 4.1: Knowledge-Based Recommendation

Knowledge Based Recommendations

Knowledge-based recommendation systems (KBS) rely on explicit knowledge about items, users, and their relationships to generate personalized recommendations. Unlike collaborative or content-based methods, which rely on historical interactions, KBS are designed to incorporate domain-specific knowledge and user requirements, often leveraging logical reasoning or constraint satisfaction.

Types of KBS

Two basic types of knowledge-based recommender systems are **constraint-based** & **case-based** systems. Both approaches are similar in terms of the recommendation process: the user must specify the requirements, and the system tries to identify a solution. If no solution can be found, the user must change the requirements. The system may also provide explanations for the recommended items.

These recommenders, however, differ in the way they use the provided knowledge: case-based recommenders focus on the retrieval of similar items on the basis of different types of similarity measures, whereas constraint-based recommenders rely on an explicitly defined set of recommendation rules. In constraint-based systems, the set of recommended items is determined by, for instance, searching for a set of items that fulfill the recommendation rules. Case-based systems, on the other hand, use similarity metrics to retrieve items that are similar (within a predefined threshold) to the specified customer requirements.

Overview of Knowledge-Based Recommendation Systems

A KBS works by:

1. **Understanding the domain knowledge** (e.g., properties of items, relationships, and constraints).
2. **Matching user preferences or requirements** with item properties.
3. **Generating recommendations** based on explicit reasoning or matching algorithms.

Examples:

- A travel booking system that recommends destinations based on preferences like budget, weather, and activities.
- A medical recommender suggesting treatments based on patient symptoms and medical history.

Key Characteristics of KBS

1. **No dependency on user-item interaction data:** Unlike collaborative filtering, KBS does not rely on historical user interactions.
2. **Handles complex requirements:** Capable of addressing scenarios where user preferences are highly specific (e.g., "find a laptop under $1000 with at least 16GB RAM").
3. **Incorporates domain expertise:** Uses explicit knowledge representation (e.g., rules, ontologies).
4. **Constraint satisfaction:** Ensures recommendations satisfy all specified constraints or preferences.

Knowledge Representation & Reasoning

In general, knowledge-based systems rely on detailed knowledge about item characteristics.

Constraints

A constraint-based recommendation problem can, in general, be represented as a **constraint satisfaction problem** that can be solved by a **constraint solver** or in the form of a **conjunctive query** that is executed and solved by a database engine.

A classical **constraint satisfaction problem** $(CSP)^1$ can be described by a-tuple (V, D, C) where:

1. V is a set of variables
2. D is a set of finite domains for these variables
3. C is a set of constraints that describes the combinations of values the variables can simultaneously take
   A solution to a CSP corresponds to an assignment of a value to each variable in V in a way that all constraints are satisfied.

Constraint-based recommender systems can build on this formalism and exploit a *recommender knowledge base* that typically includes two different sets of variables $(V = V_C \cup V_{PROD})$, one describing potential customer requirements and the other describing product properties. Three different sets of constraints $(C = C_R \cup C_F \cup C_{PROD})$ define which items should be recommended to a customer in which situation.

1. $Customer\ properties(V_C)$: describe the possible customer requirements. The customer property max-price denotes the maximum price acceptable for the customer, the property usage denotes the planned usage of photos (print versus digital organization), and photography denotes the predominant type of photos to be taken; categories are, for example, sports or portrait photos.
2. $Product\ properties(V_{PROD})$: describe the properties of products in an assortment; for example, mpix denotes possible resolutions of a digital camera.

3. $Compatibility\ constraints(C_R)$: define allowed instantiations of customer properties – for example, if large-size photoprints are required, the maximal accepted price must be higher than 200.
4. $Filter\ conditions(C_F)$: define under which conditions which products should be selected – in other words, filter conditions define the relationships between customer properties and product properties. An example filter condition is large-size photoprints require resolutions greater than 5 mpix.
5. $Product\ constraints(C_{PROD})$: define the currently available product assortment. Each conjunction in this constraint completely defines a product (item) – all product properties have a defined value. The task of identifying a set of products matching a customer's wishes and needs is denoted as a recommendation task. The customer requirements REQ can be encoded as unary constraints over the variables in V_C and V_{PROD} – for example, max-price = 300.

The task of identifying a set of products matching a customer's wishes and needs is denoted as a *recommendation task*. The customer requirements REQ can be encoded as unary constraints over the variables in V_C and V_{PROD} for example, max-price = 300. Formally, each solution to the CSP $(V = V_C \cup V_{PROD}, D, C = C_R \cup C_F \cup C_{PROD} \cup REQ)$ corresponds to a consistent recommendation. In many practical settings, the variables in V_C do not have to be instantiated, as the relevant variables are already bound to values through the constraints in REQ. The task of finding such valid instantiations for a given constraint problem can be accomplished by every standard constraint solver.

Conjunctive queries: A slightly different way of constraint-based item retrieval for a given catalog, is to view the item selection problem as a data filtering task. The main task in such an approach, therefore, is not to find valid variable instantiations for a CSP but rather to construct a conjunctive database query that is executed against the item catalog. A conjunctive query is a database query with a set of selection criteria that are connected conjunctively.

Cases & Similarities

Case-based recommendation systems mostly exploit similarity metrics for the retrieval of items from a catalog. In case-based recommendation approaches, items are retrieved using similarity measures that describe to which extent item properties match some given user's requirements. The so-called distance similarity of an item p to the requirements $r \in REQ$ is often defined as shown in Formula 4.1. In this context, $sim(p, r)$ expresses for each item attribute value $\varphi_r(p)$ its distance to the customer requirement $r \in REQ$ – for example, $\varphi_{mpix}(p1) = 8.0$ Furthermore, w_r is the importance weight for requirement r.

$$similarity(p, REQ) = \frac{\sum_{r \in REQ} w_r * sim(p, r)}{\sum_{r \in REQ} w_r}$$

Where, MIB / LIB:

$$sim(p,r) = \frac{\varphi_r(p) - min(r)}{max(r) - min(r)} = \frac{max(r) - \varphi_r(p)}{max(r) - min(r)}$$

Constraint-Based Recommenders

The general interaction flow of a knowledge-based, conversational recommender can be summarized as follows:

1. The user specifies his or her initial preferences – for example, by using a web-based form. Such forms can be identical for all users or personalized to the specific situation of the current user. Some systems use a question/answer preference elicitation process, in which the questions can be asked either all at once or incrementally in a wizard-style, interactive dialog.
2. When enough information about the user's requirements and preferences has been collected, the user is presented with a set of matching items. Optionally, the user can ask for an explanation as to why a certain item was recommended.
3. The user might revise his or her requirements, for instance, to see alternative solutions or narrow down the number of matching items.
   More on CBS can be found in RS Module 5 Notes

Interacting with Constraint-based Recommenders

In this section we analyze in detail different techniques to support users in the interaction with constraint-based recommender applications. These techniques help improve the usability of these applications and achieve higher user acceptance in dimensions such as trust or satisfaction with the recommendation process and the output quality.

1. **Proposing default values**: Defaults are an important means to support customers in the requirements specification process, especially in situations in which they are unsure about which option to select or simply do not know technical details. Defaults can support customers in choosing a reasonable alternative. For example, if a customer is interested in printing large-format pictures from digital images, the camera should support a resolution of more than 5.0 megapixels (default). The negative side of the coin is that defaults can also be abused to manipulate consumers to choose certain options. For example, users can be stimulated to buy a park distance control functionality in a car by presenting the corresponding default value. Defaults can be specified in various ways:
   1. **Static defaults**: In this case, one default is specified per customer property – for example, default(usage)=large-print, because typically users want to generate posters from high-quality pictures.
   2. **Dependent defaults**: In this case a default is defined on different combinations of potential customer requirements – for example, default(usage=small-print, max-price) = 300.
   3. **Derived defaults**: When the first two default types are strictly based on a declarative approach, this third type exploits existing interaction logs for the

automated derivation of default values. Derived defaults can be determined based on various schemes; basic approaches to the determination of suitable default values are, *1-nearest neighbor* & *weighted majority voter*.

2. **Selecting the next question**: Besides using defaults to support the user in the requirements specification process, the interaction log and the default mechanism can also be applied for identifying properties that may be interesting for the user within the scope of a recommendation session. For example, if a user has already specified requirements regarding the properties price and opt-zoom, defaults could propose properties that the user could be interested to specify next. The precondition for such approaches is the availability of user interaction logs. One basic approach to the determination of defaults for the presentation of selectable customer properties, in which question recommendation is based on the principle of frequent usage (popularity). Such a popularity value can be calculated using Formula below, in which the recommendation of a question depends strictly on the number of previous selections of other users.

$$popularity(attribute, pos) = \frac{\#selections(attribute, pos)}{\#sessions}$$

3. **Ranking the items/utility-based recommendation**: It is important to rank recommended items according to their utility for the customer. Because of primacy effects that induce customers to preferably look at and select items at the beginning of a list, such rankings can significantly increase the trust in the recommender application as well as the willingness to buy. In knowledge-based conversational recommenders, the ranking of items can be based on the multi-attribute utility theory (MAUT), which evaluates each item with regard to its utility for the customer. Each item is evaluated according to a predefined set of dimensions that provide an aggregated view on the basic item properties. For example, quality and economy are dimensions in the domain of digital cameras; availability, risk, and profit are such dimensions in the financial services domain. The formula for calculating the overall utility is given below:

$$utility(p) = \sum_{j=1}^{\#(dimensions)} interest(j) * contribution(p, j)$$

Case-Based Recommenders

Similar to constraint-based recommenders, earlier versions of case-based rec- ommenders followed a pure query-based approach, in which users had to specify (and often respecify) their requirements until a target item (an item that fits the user's wishes and needs) has been identified. Especially for nonexperts in the product domain, this type of requirement elicitation process can lead to tedious recommendation sessions, as the interdependent properties of items require a substantial domain knowledge to perform well. This drawback of pure query-based approaches motivated the development of browsing-based approaches

to item retrieval, in which users – maybe not knowing what they are seeking – are navigating in the item space with the goal to find useful alternatives.

Interacting with Case-based Recommenders

Critiquing is an effective way to support such navigations and, in the meantime, it is one of the key concepts of case-based recommendation; this concept will be discussed in detail in the following subsections:

1. **Critiquing**: The idea of critiquing is that users specify their change requests in the form of goals that are not satisfied by the item currently under consideration (entry item or recommended item). If, for example, the price of the currently displayed digital camera is too high, a critique cheaper can be activated; if the user wants to have a camera with a higher resolution (mpix), a corresponding critique more mpix can be selected. On one hand, critiquing supports an effective navigation in the item space; on the other hand, similarity-based case retrieval supports the identification of the most similar items – that is, items similar to those currently under consideration. Critiquing-based recommender systems allow users to easily articulate preferences without being forced to specify concrete values for item properties (see the previous example). The goal of critiquing is to achieve time savings in the item selection process and, at the same time, achieve at least the same recommendation quality as standard query-based approaches. The

major steps of a critiquing-based recommender application are the following:

Algorithm 4.3 SIMPLECRITIQUING(q, CI)

Input: Initial user query q; Candidate items CI
procedure SIMPLECRITIQUING(q, CI)
 repeat
  $r \leftarrow$ ITEMRECOMMEND(q, CI);
  $q \leftarrow$ USERREVIEW(r, CI);
 until empty(q)
end procedure

procedure ITEMRECOMMEND(q, CI)
 $CI \leftarrow \{ci \in CI$: satisfies($ci$, q)$\}$;
 $r \leftarrow$ mostsimilar(CI, q);
 return r;
end procedure

procedure USERREVIEW(r, CI)
 $q \leftarrow$ critique(r);
 $CI \leftarrow CI - r$;
 return q;
end procedure

2. **Compound Critiquing**: Allowing the specification of critiques that operate over multiple properties can significantly improve the efficiency of recommendation dialogs, for example, in terms of a reduced number of critiquing cycles. Such critiques are denoted as compound critiques. An important advantage of compound critiques is that they allow a faster progression through the item space. However, compound critiques still have disadvantages as long as they are formulated statically, as all critique alternatives are available for every item displayed.
3. **Dynamic critiquing**: exploits patterns, which are generic descriptions of dif- ferences between the recommended (entry) item and the candidate items – these patterns are used for the derivation of compound critiques. Critiques are denoted as dynamic because they are derived on the fly in each critiquing cycle. Dynamic critiques are calculated using the concept of association rule mining. Such a rule can be, for example, “42.9% of the remaining digital cameras have a higher zoom and a lower price”. The critique that corresponds to this property combination is “more zoom and

lower price". A dynamic critiquing cycle consists of the following basic steps:

Algorithm 4.4 DYNAMICCRITIQUING(q, CI)

Input: Initial user query q; Candidate items CI;
number of compound critiques per cycle k;
minimum support for identified association rules σ_{min}

procedure DYNAMICCRITIQUING(q, CI, k, σ_{min})
repeat
$r \leftarrow$ ITEMRECOMMEND(q, CI);
$CC \leftarrow$ COMPOUNDCRITIQUES(r, CI, k, σ_{min});
$q \leftarrow$ USERREVIEW(r, CI, CC);
until empty(q)
end procedure

procedure ITEMRECOMMEND(q, CI)
$CI \leftarrow \{ci \in CI$: satisfies($ci, q$)$\}$;
$r \leftarrow$ mostsimilar(CI, q);
return r;
end procedure

procedure USERREVIEW(r, CI, CC)
$q \leftarrow$ critique(r, CC);
$CI \leftarrow CI - r$;
return q;
end procedure

procedure COMPOUNDCRITIQUES(r, CI, k, σ_{min})
$CP \leftarrow$ CRITIQUEPATTERNS(r, CI);
$CC \leftarrow$ APRIORI(CP, σ_{min});
$SC \leftarrow$ SELECTCRITIQUES(CC, k);
return SC;
end procedure

4. **Advanced item recommendation**: After a critique has been selected by the user, the next item must be proposed (recommended item for the next critiquing cycle). An approach to doing this, besides the application of simple similarity measures is described, where a compatibility score is introduced that represents the percentage of compound critiques $cc_i \in CC_U$ that already have been selected by the user and are consistent with the candidate item ci. This compatibility-based approach to item

selection is implemented in Formula:

$$compatibility(ci, CC_U) = \frac{|cc_i \in CC_U : satisfies(cc_i, ci)|}{|CC_U|}$$

Module 4.2: Hybrid Approaches

Hybrid Recommendation

Hybrid recommendation systems (HRS) combine two or more recommendation techniques to leverage the strengths of individual approaches while minimizing their weaknesses. These systems aim to improve the accuracy, diversity, and robustness of recommendations compared to standalone methods like collaborative filtering or content-based filtering.

Opportunities for Hybridization

Hybrid recommendation systems are designed to address limitations of single-method approaches:

- **Cold-Start Problem**: Combining techniques helps handle new users (cold-start users) or items.
- **Sparsity Problem**: Hybrid systems mitigate sparse user-item interaction data.
- **Accuracy and Diversity**: By integrating multiple techniques, hybrid systems improve recommendation relevance and diversity.
- **Adaptability**: Hybridization allows systems to adapt to various user scenarios and contexts.

Why Use Hybridization?

1. **Overcome Limitations**:
   - Collaborative filtering struggles with cold start or sparsity.
   - Content-based filtering may suffer from overspecialization (recommendations are too similar to past behavior).
2. **Improve Personalization**:
   - Combining methods allows better understanding of user preferences.
3. **Leverage Multiple Data Sources**:
   - Incorporates user ratings, item metadata, context, and knowledge.

Monolithic Hybridization Design

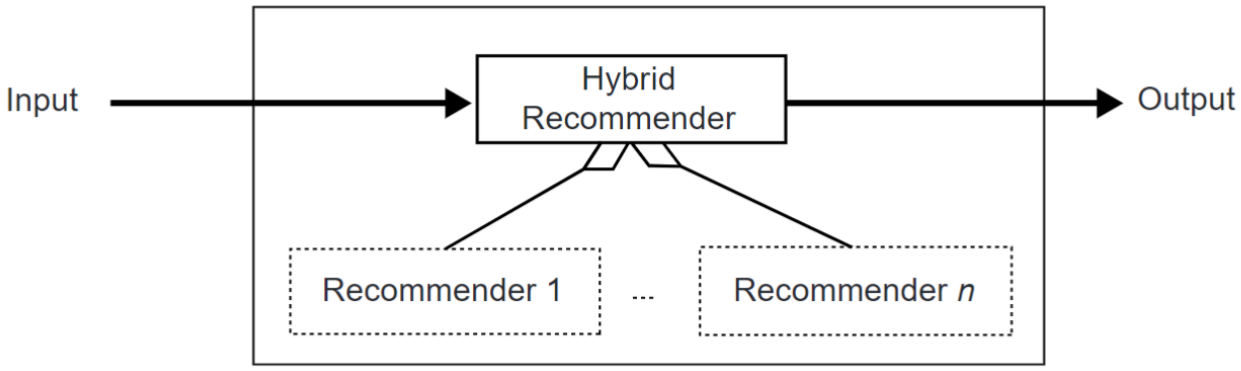

Figure 5.2. Monolithic hybridization design.

In monolithic designs, the recommendation techniques are tightly integrated into a single model. Hybridization is thus achieved by a built-in modification of the algorithm behavior to exploit different types of input data. Typically, data-specific preprocessing steps are used to transform the input data into a representation that can be exploited by a specific algorithm paradigm.

Techniques

1. **Feature Combination**:
    - A feature combination hybrid is a monolithic recommendation component that uses a diverse range of input data. For instance, a feature combination hybrid that combines collaborative features, such as a user's likes and dislikes, with content features of catalog items. This method allows the recommendation engine to utilize various types of information to improve prediction accuracy.
    - For example, in a movie recommendation system, user interaction data like ratings or watch history can be combined with item metadata such as genres, cast, directors, and release year. These combined features can then be processed using a supervised learning model like regression, decision trees, or deep learning to predict the likelihood of a user liking a movie. Feature combination allows the recommendation system to capture nuanced interactions between user behaviors and item characteristics. This approach is particularly effective in handling sparse datasets, as it leverages diverse data points from multiple sources to enhance predictive accuracy.
2. **Feature Augmentation**:
    - Feature augmentation is another monolithic hybridization design that may be used to integrate several recommendation algorithms. In contrast with feature combination, this hybrid does not simply combine and preprocess several types of input, but rather applies more complex transformation steps. In fact, the output of a contributing recommender system augments the feature space of the actual recommender by preprocessing its knowledge sources. However, this must not be mistaken for a pipelined design, because the implementation of the contributing

recommender is strongly interwoven with the main component for reasons of performance and functionality.

- It involves enriching the feature set of one recommendation technique with features derived from another technique. Here, one method (e.g., collaborative filtering) is used to generate additional insights or predictions that are added as features to the input of another method (e.g., content-based filtering).
- Feature augmentation is particularly useful when one recommendation method alone is insufficient. For example, when cold-start problems arise for new items, metadata from content-based filtering can be augmented with collaborative filtering insights derived from similar items. This augmentation helps mitigate the sparsity of user-item interactions while preserving the strengths of collaborative filtering.

Comparison Table: Feature Combination vs. Feature Augmentation

Aspect	Feature Combination	Feature Augmentation
Definition	Merges features from multiple data sources into a single model.	Enhances the feature set of one method by incorporating insights or predictions from another.
Usage	Features from collaborative, content-based, and demographic data are equally combined.	Output or predictions from one technique are used as input features for another.
Focus	Treats all feature sources equally.	Focuses on augmenting the feature set of a primary technique.
Cold-Start Problem	Handles cold-start problems effectively by combining diverse data sources.	Mitigates cold-start issues by enriching metadata with predictions from another method.
Implementation	Uses combined features directly in a supervised model like regression or neural networks.	Uses the output of one technique (e.g., collaborative filtering) as input features for another (e.g., content-based).
Example	Combining user ratings with item genres, keywords, and demographic information.	Augmenting collaborative filtering predictions with metadata such as genres, cast, or release year.
Strengths	Captures multi-source dependencies and enhances predictive power.	Improves performance by enriching the feature set without altering the original algorithm structure.

Aspect	Feature Combination	Feature Augmentation
Real-World Example	Amazon: Combines user purchase history and product descriptions for personalized recommendations.	Spotify: Augments collaborative filtering predictions with audio features like tempo and pitch.

Parallelized Hybridization Design

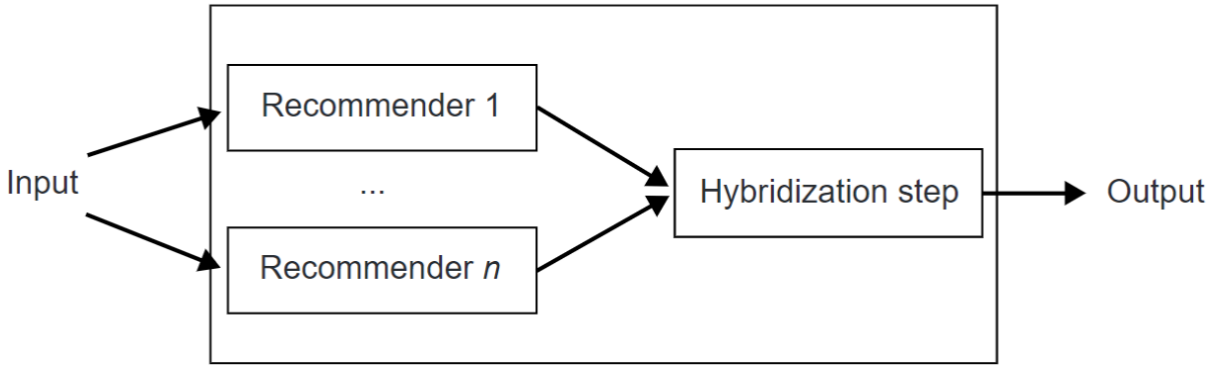

Figure 5.3. Parallelized hybridization design.

In parallelized designs, multiple recommendation techniques work independently, and their outputs are combined. Parallelized hybridization designs employ several recommenders side by side and employ a specific hybridization mechanism to aggregate their outputs. Burke elaborates on the mixed, weighted, and switching strategies. However, additional combination strategies for multiple recommendation lists, such as majority voting schemes, may also be applicable.

Techniques:

1. **Weighted Hybridization**:
    - Assign weights to the results of different methods and combine them.
    - A weighted hybridization strategy combines the recommendations of two or more recommendation systems by computing weighted sums of their scores. Thus, given n different recommendation functions reck with associated relative weights β_k:

$$rec_{weighted}(u,i) = \sum_{k=1}^{n} \beta_k \times rec_k(u,i)$$

    - Obviously, this technique is quite straightforward and is thus a popular strategy for combining the predictive power of different recommen- dation techniques in a weighted manner.
2. **Switching Hybridization**:
    - Dynamically switch between recommendation techniques based on the context or user behavior.

- Switching hybrids require an oracle that decides which recommender should be used in a specific situation, depending on the user profile and/or the quality of recommendation results. Such an evaluation could be carried out as follows:

$$\exists_1 k : 1 \ldots n \; rec_{switching}(u, i) = rec_k(u, i)$$

- where k is determined by the switching condition. For instance, to overcome the cold-start problem, a knowledge-based and collaborative switching hybrid could initially make knowledge-based recommendations until enough rating data are available. When the collaborative filtering component can deliver recommendations with sufficient confidence, the recommendation strategy could be switched.
- Example: Use content-based filtering for new users and collaborative filtering for existing users.

3. **Mixed Hybridization**:
   - Merge recommendations from multiple methods into a single list. A mixed hybridization strategy combines the results of different recommender systems at the level of the user interface, in which results from different techniques are presented together. Therefore the recommendation result for user u and item i of a mixed hybrid strategy is the set of -tuples $\langle score, k \rangle$ for each of its n constituting recommenders rec_k:

$$rec_{mixed}(u, i) = \bigcup_{k=1}^{n} \langle rec_k(u, i), k \rangle$$

   - Example: Present a ranked list where recommendations from CF and CBF are combined.

Pipelined Hybridization Design

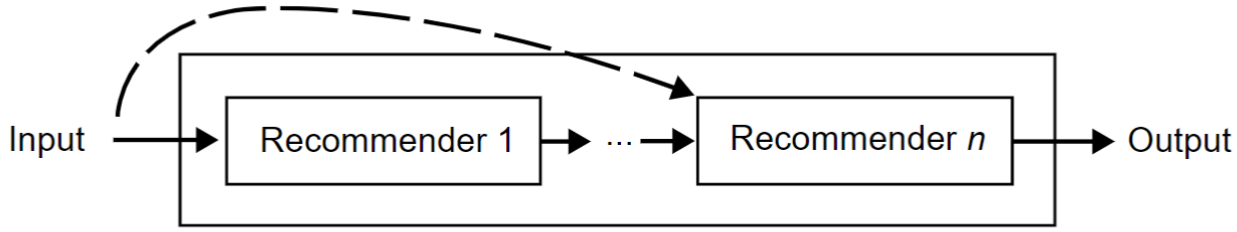

Figure 5.4. Pipelined hybridization design.

In pipelined designs, the output of one method becomes the input for another. Pipelined hybrids implement a staged process in which several techniques sequentially build on each other before the final one produces recommendations for the user. The pipelined hybrid variants differentiate themselves mainly according to the type of output they produce for the next stage. In other words, a preceding component may either preprocess input data to build

a model that is exploited by the subsequent stage or deliver a recommendation list for further refinement.

Techniques:

1. **Cascade Hybridization**:
    - Apply one method first to narrow down the candidate set and refine it using another method. Cascade hybrids are based on a sequenced order of techniques, in which each succeeding recommender only refines the recommendations of its predecessor. The recommendation list of the successor technique is thus restricted to items that were also recommended by the preceding technique.
    - Example:
        - Use collaborative filtering to identify similar users.
        - Use content-based filtering to select the most relevant items from the similar users' preferences.
2. **Meta-Level Hybridization**:
    - Use the model from one method as an input feature for another method.
    - In a meta-level hybridization design, one recommender builds a model Δ that is exploited by the principal recommender to make recommendations. Formula below formalizes this behavior, wherein the nth recommender exploits a model that has been built by its predecessor. However, in all reported systems so far, n has always been 2.

$$rec_{meta-level}(u, i) = rec_n(u, i, \Delta_{rec_n-1})$$

    - Example: Train a collaborative filtering model, then use its embeddings as features for a knowledge-based recommender.

Summary

Hybrid approaches in recommendation systems overcome the limitations of individual methods by combining their strengths. Depending on the use case and data availability, various hybridization strategies can be implemented, such as weighted, switching, mixed, or pipelined designs. Hybrid RS is widely adopted in modern recommendation systems to provide accurate, diverse, and contextually relevant suggestions.