## MODULE 3

### Hybrid recommendation approaches

Collaborative filtering systems rely on community ratings, content-based methods rely on textual descriptions and the target user's own ratings, and knowledge-based systems rely on interactions with the user in the context of knowledge bases. Similarly, demographic systems use the demographic profiles of the users to make recommendations. It is noteworthy that these different systems use different types of input, and have different strengths and weaknesses. Some recommender systems, such as knowledge-based systems, are more effective in cold-start settings where a significant amount of data is not available.

Other recommender systems, such as collaborative methods, are more effective when a lot of data is available. In many cases where a wider variety of inputs is available, one has the flexibility of using different types of recommender systems for the same task. In such cases, many opportunities exist for hybridization, where the various aspects from different types of systems are combined to achieve the best of all worlds. Hybrid recommender systems are closely related to the field of ensemble analysis, in which the power of multiple types of machine learning algorithms is combined to create a more robust model. Ensemble-based recommender systems are able to combine not only the power of multiple data sources, but they are also able to improve the effectiveness of a particular class of recommender systems (e.g., collaborative systems) by combining multiple models of the same type. This scenario is not very different from that of ensemble analysis in the field of data classification

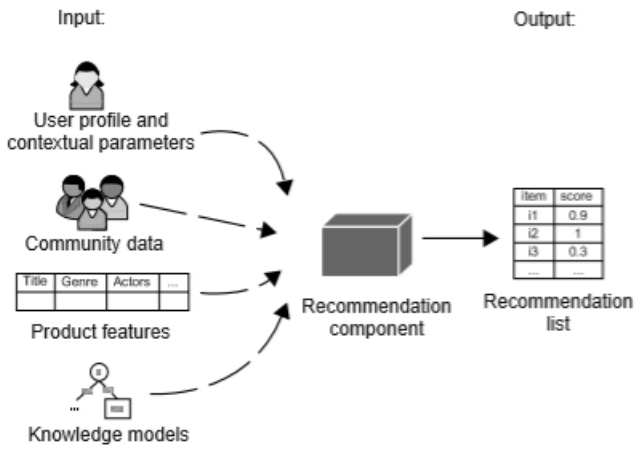


Figure 5.1. Recommender system as a black box.

Figure 5.1 sketches a recommendation system as a black box that transforms input data into a ranked list of items as output. User models and contextual information, community and product data, and knowledge models constitute the potential types of recommendation input. However, none of the basic approaches is able to fully exploit all of these. Consequently, building hybrid systems that combine the strengths of different algorithms and models to overcome some of the aforementioned shortcomings and problems has become the target of recent research.

Dimensions of hybrid recommendation system

- Recommendation paradigm
- Hybridization design- method used to combine two or more algorithms.

Recommendation paradigms

In general, a recommendation problem can be treated as a utility function rec that predicts the usefulness of an item i in a set of items I for a specific user u from the universe of all users U. The prediction task of a recommender algorithm is to presume this utility score for a given user and item. The selection task of any recommender system RS is to identify those n items from a catalog I that achieve the highest utility scores for a given user u:

$$RS(u, n) = \{i_1, \ldots, i_k, \ldots, i_n\}, \text{ where}$$

$$i_1, \ldots, i_n \in I \text{ and}$$

$$\forall k \; rec(u, i_k) > 0 \wedge rec(u, i_k) > rec(u, i_{k+1})$$

Thus, a recommendation system RS will output a ranked list of the top n items that are presumably of highest utility for the given user.

We focus on the three base recommendation paradigms: collaborative, content-based and knowledge-based.

Hybridization designs

1. Monolithic hybridization design
2. Parallelised
3. Pipelined

Monolithic denotes a hybridization design that incorporates aspects of several recommendation strategies in one algorithm implementation. As depicted in Figure 5.2, several recommenders contribute virtually because the hybrid uses additional input data that are specific to another recommendation algorithm, or the input data are augmented by one technique and factually exploited by the other. For instance, a content-based recommender that also exploits community data to determine item similarities falls into this category.

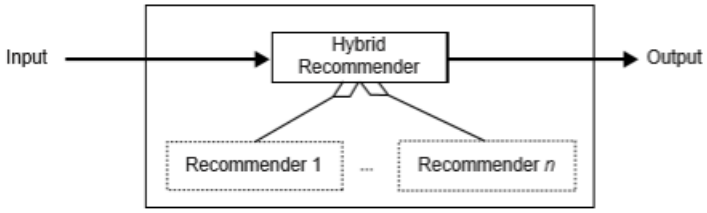


Figure 5.2. Monolithic hybridization design.

The two remaining hybridization designs require at least two separate recommender implementations, which are consequently combined. Based on their input, parallelized hybrid recommender systems operate independently of one another and produce separate recommendation lists, as sketched in Figure 5.3. In a subsequent hybridization step, their output is combined into a final set of recommendations. The weighted, mixed, and switching strategies require recommendation components to work in parallel.

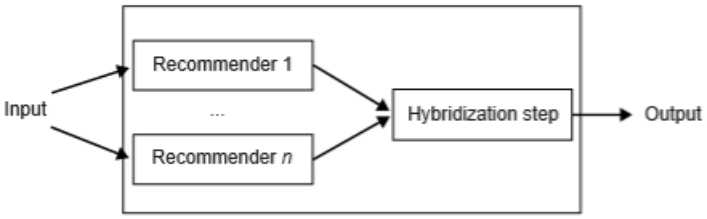


Figure 5.3. Parallelized hybridization design.

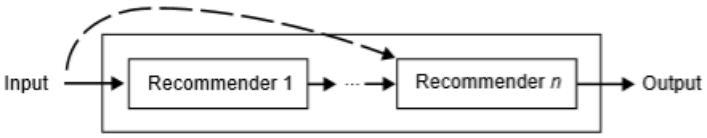


Figure 5.4. Pipelined hybridization design.

When several recommender systems are joined together in a pipeline architecture, as depicted in Figure 5.4, the output of one recommender becomes part of the input of the subsequent oneThe Cascade and meta-level hybrids are examples of such pipeline architectures.

Monolithic hybridization design

Whereas the other two designs for hybrid recommender systems consist of two or more components whose results are combined, monolithic hybrids consist of a single recommender component that integrates multiple approaches by preprocessing and combining several knowledge sources. Hybridization is thus achieved by a built-in modification of the algorithm behavior to exploit different types of input data. Both feature combination and feature augmentation strategies can be assigned to this category.

a. Feature combination hybrids

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. The following example illustrates this technique in the book domain.

Table 5.2. *Community and product knowledge.*

| User | Item1 | Item2 | Item3 | Item4 | Item5 |
|---|---|---|---|---|---|
| Alice | | 1 | | 1 | |
| User1 | | 1 | 1 | | 1 |
| User2 | 1 | 1 | | | 1 |
| User3 | 1 | | 1 | | |
| User4 | | | | | 1 |

| Item | Genre |
|---|---|
| Item1 | romance |
| Item2 | mystery |
| Item3 | mystery |
| Item4 | mystery |
| Item5 | fiction |

Table 5.3. *Hybrid input features.*

| Feature | Alice | User1 | User2 | User3 | User4 |
|---|---|---|---|---|---|
| User likes many *mystery* books | true | true | | | |
| User likes some *mystery* books | | | true | true | |
| User likes many *romance* books | | | | | |
| User likes some *romance* books | | | true | true | |
| User likes many *fiction* books | | | | | |
| User likes some *fiction* books | | true | true | | true |

Table 5.2 presents several users' unary ratings of a product catalog. Ratings constitute implicitly observed user feedback, such as purchases. Product knowledge is restricted to an item's genre. Obviously, in a pure collaborative approach, without considering any product features, both User1 and User2 would be judged as being equally similar to Alice. However, the feature-combination approach identifies new hybrid features based on community and product data.

Table 5.3 thus provides a hybrid encoding of the information contained in Table 5.2. These features are derived by the following rules. If a user bought mainly books of genre X (i.e., two-thirds of the total purchases and at least two books) we set the user characteristic User likes many X books to true. Analogously, we also set User likes some X books to true, requiring one-third of the user's purchases, or at least one item in absolute numbers. Although the transformation seems to be quite trivial at first glance, it nevertheless reflects that some knowledge, such as an item's genre, can lead to considerable improvements. Initially Alice seems to possess similar interests to User1 and User2, but the picture changes after transforming the user/item matrix. It actually turns out that User2 behaves in a very indeterminate manner by buying some books of all three genres. In contrast, User1 seems to focus specifically on mystery books, as Alice does.

Another approach for feature combination was is to exploit different types of rating feedback based on their predictive accuracy and availability.

Table 5.4. *Different types of user feedback.*

| User | $R_{nav}$ | $R_{view}$ | $R_{ctx}$ | $R_{buy}$ |
|---|---|---|---|---|
| Alice | $n_3, n_4$ | $i_5$ | $k_5$ | $\emptyset$ |
| User1 | $n_1, n_5$ | $i_3, i_5$ | $k_5$ | $i_1$ |
| User2 | $n_3, n_4$ | $i_3, i_5, i_7$ | $\emptyset$ | $i_3$ |
| User3 | $n_2, n_3, n_4$ | $i_2, i_4, i_5$ | $k_2, k_4$ | $i_4$ |

Table 5.4 depicts a scenario in which several types of implicitly or explicitly collected user feedback are available as unary ratings, such as navigation actions Rnav, click-throughs on items' detail pages Rview, contextual user requirements Rctx, or actual purchases Rbuy.These categories differentiate themselves by their availability and their aptness for making predictions. For instance, navigation actions and page views of online users occur frequently, whereas purchases are less common. In addition, information about the user's

context, such as the keywords used for searching or input to a conversational requirements elicitation dialog, is typically a very useful source for computing recommendations. In contrast, navigation actions typically contain more noise, particularly when users randomly surf and explore a shop. Therefore, rating categories may be prioritized – for example (Rbuy, Rctx) ≺ Rview ≺Rnav, where priority decreases from left to right. Thus, when interpreting all rating data in Table 5.4 in a uniform manner, User2 and User3 seem to have the most in common with Alice, as both share at least three similar ratings. However, the algorithm for neighbourhood determination uses the given precedence rules and thus initially uses Rbuy and Rctx as rating inputs to find similar peers. In this highly simplified example, User1 would be identified as being most similar to Alice.

**Feature Augmentation**

**Feature augmentation** is a **hybridization method** in recommender systems where one algorithm's output enhances the features of another. Unlike simple **feature combination**, it involves **transforming and augmenting the input data** using the results of other recommendation algorithms. Here's a detailed explanation:

---

**Key Points**

1. **Definition**:
   - The **output of one recommender system** becomes part of the feature space (input) for another recommender system.
   - It doesn't just merge inputs but **preprocesses and transforms the data** to enhance recommendations.
2. **How It Works**:
   - A contributing algorithm provides additional insights, which are **fed into the primary algorithm** to refine predictions.
   - Example: In **content-boosted collaborative filtering**, content-based predictions are used to fill missing user ratings. These filled-in ratings enhance collaborative filtering by adding more data to work with.
3. **Output**:
   - The final output is a **refined prediction**, with the augmented features providing a richer context for decision-making.

---

**Example: Content-Boosted Collaborative Filtering**

- **Problem**: Collaborative filtering struggles when there is a lack of user ratings (cold-start problem).
- **Solution**: Use content-based predictions to fill missing ratings and augment the collaborative filtering process.

**Steps:**

1. **Content-Based Predictions (cu,i)**:
   - For missing user ratings, predict them using content features like item attributes.
   - For instance, if Alice hasn't rated a movie, we estimate her rating using the genre, director, and similar movies she likes.
2. **Augmented Rating Matrix**:

- The matrix now includes actual ratings ru,ir_{u,i}ru,i and predicted ratings cu,ic_{u,i}cu,i, ensuring no gaps.

3. **Collaborative Filtering with Augmented Features**:
   - Apply collaborative filtering to the updated matrix. The system now works with both user data and augmented predictions.

---

**Advantages of Feature Augmentation**

1. **Handles Sparse Data**: By filling gaps in the user/item matrix, it resolves the cold-start problem.
2. **Improved Accuracy**: The hybridization leverages multiple data sources to refine predictions.
3. **Dynamic and Flexible**: Easily adaptable to various recommendation scenarios.

Parallelized hybridization design

Parallelized hybridization designs employ several recommenders side by side and employ a specific hybridization mechanism to aggregate their outputs. Burke (2002b) 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.

Mixed hybrids

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(score,k) for each of its n constituting recommenders reck

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

The top-scoring items for each recommender are then displayed to the user next to each other.

Weighted hybrids

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 βk:

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

Obviously, this technique is quite straightforward and is thus a popular strategy for combining the predictive power of different recommendation techniques in a weighted manner. Consider an example in which two recommender systems are used to suggest one of five items for a user Alice. As can be easily seen from Table 5.6, these recommendation lists are hybridized by using a uniform weighting scheme with β1 = β2 = 0.5. The weighted hybrid recommender recw thus produces a new ranking by combining the scores from rec1 and rec2. Items that are recommended by only one of the two users, such as Item2, may still be ranked highly after the hybridization step.

Table 5.6. *Recommendations of weighted hybrid.*

| item | $rec_1$ score | $rec_1$ rank | $rec_2$ score | $rec_2$ rank | $rec_w$ score | $rec_w$ rank |
|---|---|---|---|---|---|---|
| Item1 | 0.5 | 1 | 0.8 | 2 | 0.65 | 1 |
| Item2 | 0 | | 0.9 | 1 | 0.45 | 2 |
| Item3 | 0.3 | 2 | 0.4 | 3 | 0.35 | 3 |
| Item4 | 0.1 | 3 | 0 | | 0.05 | |
| Item5 | 0 | | 0 | | 0 | |

Switching hybrids

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. Furthermore, a switching strategy can be applied to optimize results in a similar fashion to the NewsDude system

Pipelined hybridization design

Pipelined hybrids implement a staged process in which several techniques sequentiallybuildoneachotherbeforethefinaloneproducesrecommendations 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.

Cascade hybrids

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. Formally, assume a sequence of n techniques, where rec1 represents the recommendation function of the first technique and recn the last one. Consequently, the final recommendation score for an item is computed by the nth technique. However, an item will be suggested by the kth technique only if the (k−1)th technique also assigned a nonzero score to it. This applies to all $k \geq 2$ by induction as defined in Formula.

$$rec_{cascade}(u, i) = rec_n(u, i)$$

where $\forall k \geq 2$ must hold:

$$rec_k(u, i) = \begin{cases} rec_k(u, i) & : rec_{k-1}(u, i) \neq 0 \\ 0 & : \text{else} \end{cases}$$

Thus in a cascade hybrid all techniques, except the first one, can only change the ordering of the list of recommended items from their predecessor or exclude an item by setting its utility to 0. However, they may not introduce new items – items that have already been excluded by one of the higher-priority techniques – to the recommendation list. Thus cascading strategies do have the unfavorable property of potentially reducing the size of the recommendation set as each additional technique is applied. As a consequence, situations can arise in which cascade algorithms do not deliver the required number of propositions, thus decreasing the system's usefulness. Therefore cascade hybrids may be combined with a switching strategy to handle the case in which the cascade strategy does not produce enough recommendations

Meta-level hybrids

In a meta-level hybridization design, one recommender builds a model that is exploited by the principal recommender to make recommendations. Formula (5.8) 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. For instance, the Fab system exploits a collaborative approach that builds on user models that have been built by a content-based recommender. The application domain of Fab is online news. Fab employs a content-based recommender that builds user models based on a vector of term categories and the users' degrees of interest in them. The recommendation step, however, does not propose items that are similar to the user model, but employs a collaborative technique. The latter

determines the user's nearest neighbors based on content models and recommends items that similar peers have liked.

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