5
Hybrid recommendation approaches

The three most prominent recommendation approaches discussed in the previous chapters exploit different sources of information and follow different paradigms to make recommendations. Although they produce results that are considered to be personalized based on the assumed interests of their recipients, they perform with varying degrees of success in different application domains. Collaborative filtering exploits a specific type of information (i.e., item ratings) from a user model together with community data to derive recommendations, whereas content-based approaches rely on product features and textual descriptions. Knowledge-based algorithms, on the other hand, reason on explicit knowledge models from the domain. Each of these basic approaches has its pros and cons – for instance, the ability to handle data sparsity and cold-start problems or considerable ramp-up efforts for knowledge acquisition and engineering. These have been discussed in the previous chapters. 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. From a linguistic point of view, the term *hybrid* derives from the Latin noun *hybrida* (of mixed origin) and denotes an object made by combining two different elements. Analogously, hybrid recommender systems are technical approaches that combine several algorithm implementations or recommendation components. The following section introduces opportunities for hybridizing algorithm variants and illustrates them with several examples.

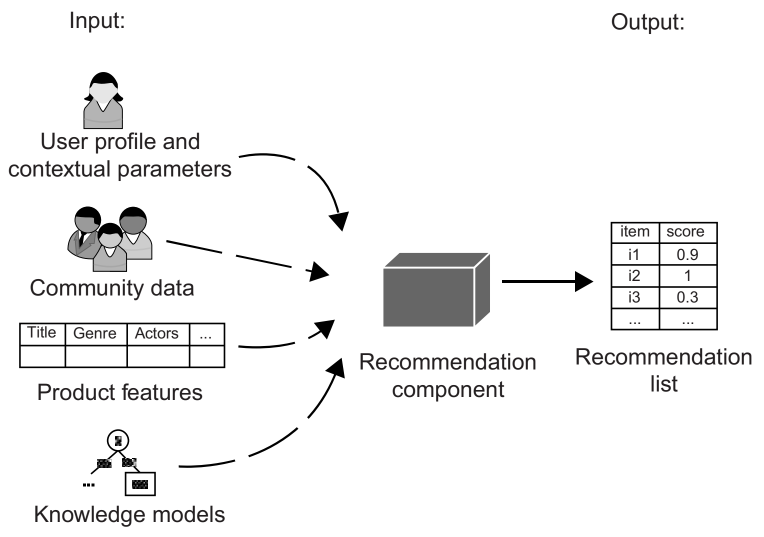

Figure 5.1. Recommender system as a black box.

5.1 Opportunities for hybridization

Although many recommender applications are actually hybrids, little theoretical work has focused on how to hybridize algorithms and in which situations one can expect to benefit from hybridization. An excellent example for combining different recommendation algorithm variants is the Netflix Prize competition[1], in which hundreds of students and researchers teamed up to improve a collaborative movie recommender by hybridizing hundreds of different collaborative filtering techniques and approaches to improve the overall accuracy. Robin Burke's article, "Hybrid Recommender Systems: Survey and Experiments" (2002b) is a well-known survey of the design space of different hybrid recommendation algorithms. It proposes a taxonomy of different classes of recommendation algorithms. Collaborative filtering and content-based and knowledge-based recommender systems are the three base approaches covered in this chapter. Furthermore, Burke (2002b) investigates utility-based recommendation that can be considered as a specific subset of knowledge-based recommender systems, because a utility scheme can be seen as another specific encoding of explicit personalization knowledge. Demographic recommender systems make collaborative propositions based on demographic user profiles.

[1] See http://www.netflixprize.com for reference.

As mentioned in Chapter 2, demographic information can be seen as just an additional piece of user knowledge that can be exploited to determine similar peers on the web and is therefore a variant of a collaborative approach. For instance, in the case that few user ratings are available, demographic data can be used to bootstrap a recommender system as demonstrated by Pazzani (1999b). Thus, the basic recommendation paradigm is one dimension of the problem space and will be discussed further in the following subsection.

The second characterizing dimension is the system's hybridization design – the method used to combine two or more algorithms. Recommendation components can work in parallel before combining their results, or two or more single recommender systems may be connected in a pipelining architecture in which the output of one recommender serves as input for the next one.

5.1.1 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. Adomavicius and Tuzhilin (2005) thus formalized *rec* as a function $U \times I \mapsto R$. In most applications, R is the interval $[0 \ldots 1]$ representing the possible utility scores of a recommended item. An item's utility must always be seen in the microeconomic context of a specific user and the sales situation he or she is currently in. Utility therefore denotes an item's capability to fulfill an abstract goal, such as best satisfying the assumed needs of the user or maximizing the retailer's conversion rate. Consequently, the prediction task of a recommender algorithm is to presume this utility score for a given user and item. Utility-based or knowledge-based recommendation systems, for instance, derive the score values directly from a priori known utility schemes; collaborative filtering methods estimate them from community ratings. In contrast, 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:

$$RS(u, n) = \{i_1, \ldots, i_k, \ldots, i_n\}, \quad \text{where} \tag{5.1}$$

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

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

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 will come back to this formalization in the following sections to specify the different hybridization designs.

As already mentioned, we focus on the three base recommendation paradigms: collaborative, content-based and knowledge-based. The collaborative principle assumes that there are clusters of users who behave in a similar way and have comparable needs and preferences. The task of a collaborative recommender is thus to determine similar peers and derive recommendations from the set of their favorite items. Whereas the content-based paradigm follows a "more of the same" approach by recommending items that are similar to those the user liked in the past, knowledge-based recommendation assumes an additional source of information: explicit personalization knowledge. As described in Chapter 4, this knowledge can, for instance, take the form of logical constraints that map the user's requirements onto item properties. Multiattribute utility schemes and specific similarity measures are alternate knowledge representation mechanisms. These knowledge models can be acquired from a third party, such as domain experts; be learned from past transaction data; or use a combination of both. Depending on the representation of the personalization knowledge, some form of reasoning must take place to identify the items to recommend.

Consequently, the choice of the recommendation paradigm determines the type of input data that is required. As outlined in Figure 5.1, four different types exist. The user model and contextual parameters represent the user and the specific situation he or she is currently in. For instance, the items the user has rated so far; the answers the user has given in a requirements elicitation dialogue; demographic background information such as address, age, or education; and contextual parameters such as the season of the year, the people who will accompany the user when he or she buys or consumes the item (e.g., watching a movie), or the current location of the user. The latter contextual parameters are of particular interest and are therefore extensively studied in mobile or more generally pervasive application domains; in Chapter 12 we will specifically focus on personalization strategies applied in physical environments.

It can be assumed that all these user- and situation-specific data are stored in the user's profile. Consequently, all recommendation paradigms require access to this user model to personalize the recommendations. However, depending on the application domain and the usage scenario, only limited parts of the user model may be available. Therefore, not all hybridization variants are possible or advisable in every field of application.

As depicted in Table 5.1, recommendation paradigms selectively require community data, product features, or knowledge models. Collaborative filtering, for example, works solely on community data and the current user profile.

Table 5.1. *Input data requirements of recommendation algorithms.*

Paradigm	User profile and contextual parameters	Community data	Product features	Knowledge models
Collaborative	Yes	Yes	No	No
Content-based	Yes	No	Yes	No
Knowledge-based	Yes	No	Yes	Yes

5.1.2 Hybridization designs

The second dimension that characterizes hybrid algorithms is their design. Burke's taxonomy (2002b) distinguishes among seven different hybridization strategies that we will refer to in this book. Seen from a more general perspective, however, the seven variants can be abstracted into only three base designs: monolithic, parallelized, and pipelined hybrids. *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.

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. Following Burke's taxonomy, the *weighted*, *mixed*, and *switching* strategies require recommendation components to work in parallel.

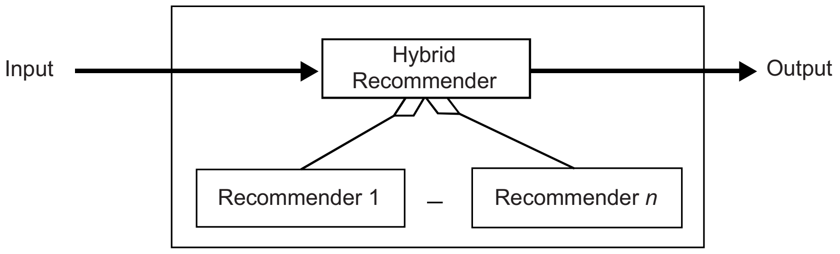

Figure 5.2. Monolithic hybridization design.

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 one. Optionally, the subsequent recommender components may use parts of the original input data, too. The *Cascade* and *meta-level* hybrids, as defined by Burke (2002b), are examples of such pipeline architectures.

The following sections examine each of the three base hybridization designs in more detail.

5.2 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. Typically, data-specific preprocessing steps are used to transform the input data into a representation that can be exploited by a specific algorithm paradigm. Following Burke's taxonomy (2002b), both *feature combination* and *feature augmentation* strategies can be assigned to this category.

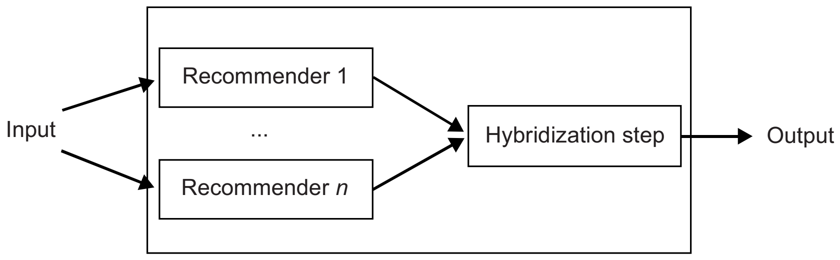

Figure 5.3. Parallelized hybridization design.

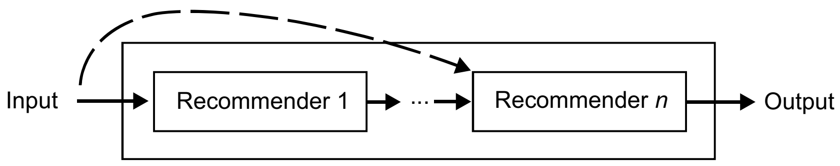

Figure 5.4. Pipelined hybridization design.

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

5.2.1 Feature combination hybrids

A feature combination hybrid is a monolithic recommendation component that uses a diverse range of input data. For instance, Basu et al. (1998) proposed 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 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 of Basu et al. identifies new hybrid features based on community and product data. This reflects "the common human-engineering effort that involves inventing good features to enable successful learning" (Basu et al. 1998).

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.

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 5.4. *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

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. For this example we determine similar peers by simply matching user characteristics, whereas Basu et al. (1998) used set-valued user characteristics and the inductive rule learner Ripper (Cohen 1996) in their original experiments.

Another approach for feature combination was proposed by Zanker and Jessenitschnig (2009b), who exploit different types of rating feedback based on their predictive accuracy and availability.

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 R_{nav}, click-throughs on items' detail pages R_{view}, contextual user requirements R_{ctx}, or actual purchases R_{buy}. 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 (Zanker and Jessenitschnig 2009a). 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 $(R_{buy}, R_{ctx}) \prec R_{view} \prec R_{nav}$, 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 neighborhood determination by Zanker and Jessenitschnig (2009b) uses the given precedence rules and thus initially uses R_{buy} and R_{ctx} as rating inputs to find similar peers. In this highly simplified example, User1 would be identified as being most similar to Alice. In the case that not enough similar peers can be determined with satisfactory confidence, the feature combination algorithm includes additional, lower-ranked rating input. A similar principle for feature combination was also exploited by Pazzani (1999b), who used demographic user characteristics to bootstrap a collaborative recommender system when not enough item ratings were known.

Because of the simplicity of feature combination approaches, many current recommender systems combine collaborative and/or content-based features in one way or another. However, the combination of input features using knowledge-based approaches, such as constraints, with content-based or collaborative knowledge sources has remained largely unexplored. We suggest that critique-based systems that elicit user feedback in the form of constraints on a specific item, such as *price should be less than the price for item a*, could be a suitable starting point for future research.

5.2.2 Feature augmentation hybrids

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, as we will discuss in the following section, because the implementation of the contributing recommender is strongly interwoven with the main component for reasons of performance and functionality.

Content-boosted collaborative filtering is an actual example of this variant (Melville et al. 2002). It predicts a user's assumed rating based on a collaborative mechanism that includes content-based predictions.

Table 5.5 presents an example user/item matrix, together with rating values for Item5 ($v_{User,Item5}$) as well as the Pearson correlation coefficient $P_{Alice,User}$ signifying the similarity between Alice and the respective users. Furthermore, it denotes the number of user ratings (n_{User}) and the number of overlapping ratings between Alice and the other users ($n_{Alice,User}$). The rating matrix for this example is complete, because it consists not only of users' actual ratings $r_{u,i}$, but also of content-based predictions $c_{u,i}$ in the case that a user rating

Table 5.5. *Hybrid input features.*

User	$v_{User,Item5}$	$P_{Alice,User}$	n_{User}	$n_{Alice,User}$
Alice	?		40	
User1	4	0.8	14	6
User2	2.2	0.7	55	28

is missing. Melville et al. (2002) therefore first create a pseudo-user-ratings vector $v_{u,i}$:

$$v_{u,i} = \begin{cases} r_{u,i} & : \text{if user } u \text{ rated item } i \\ c_{u,i} & : \text{else content-based prediction} \end{cases}$$

Based on these pseudo ratings, the algorithm computes predictions in a second step. However, depending on the number of rated items and the number of co-rated items between two users, weighting factors may be used to adjust the predicted rating value for a specific user–item pair a, i, as illustrated by the following equation for a content-boosted collaborative recommender.

$$rec_{cbcf}(a,i) = \left(sw_a c_{a,i} + \sum_{\substack{u=1 \\ u \neq a}}^{n} hw_{a,u} P_{a,u} v_{u,i} \right) \Bigg/ \left(sw_a + \sum_{\substack{u=1 \\ u \neq a}}^{n} hw_{a,u} P_{a,u} \right) \tag{5.2}$$

$hw_{a,u}$ is the hybrid correlation weight that is defined as follows:

$$hw_{a,u} = sg_{a,u} + hm_{a,u}, \text{ where}$$

$$sg_{a,u} = \begin{cases} \frac{n_{a,u}}{50} & : \text{if } n_{a,u} < 50 \\ 1 & : \text{else} \end{cases}$$

$$hm_{a,u} = \frac{2m_a m_u}{m_a + m_u} \text{ with}$$

$$m_i = \begin{cases} \frac{n_i}{50} & : \text{if } n_i < 50 \\ 1 & : \text{else} \end{cases}$$

The hybrid correlation weight ($hw_{a,u}$) adjusts the computed Pearson correlation based on a significance weighting factor $sg_{a,u}$ (Herlocker et al. 1999) that favors peers with more co-rated items. When the number of co-rated items is greater than 50, the effect tends to level off. In addition, the harmonic mean weighting factor ($hm_{a,u}$) reduces the influence of peers with a low number of user ratings, as content-based pseudo-ratings are less reliable if derived from a small number of user ratings.

Similarly, the self-weighting factor sw_i reflects the confidence in the algorithm's content-based prediction, which obviously also depends on the number of original rating values of any user i. The constant *max* was set to 2 by Melville et al. (2002).

$$sw_i = \begin{cases} \frac{n_i}{50} \times max & : \text{if } n_i < 50 \\ max & : \text{else} \end{cases}$$

Thus, assuming that the content-based prediction for Item5 is $c_{Alice,\ Item5} = 3$, $rec_{cbcf}(Alice,\ Item5)$ is computed as follows:

$$\frac{1.6 \times 3 + (0.535 \times 0.8 \times 4 + 1.45 \times 0.7 \times 2.2)}{1.6 + (0.535 \times 0.8 + 1.45 \times 0.7)} = \frac{8.745}{3.043} = 2.87$$

Consequently, the predicted value (on a 1–5 Likert scale) indicates that Alice will not be euphoric about Item5, although the most similar peer, User1, rated it with a 4. However, the weighting and adjustment factors place more emphasis on User 2 and the content-based prediction. Also note that we ignored rating adjustments based on users' rating averages as outlined in Chapter 2 and defined in Formula 2.3 for reasons of simplicity.

Additional applications of feature augmentation hybrids are presented in Mooney and Roy's (1999) discussion of a content-based book recommender and in Torres et al.'s (2004) recommendation of research papers. The latter employs, among other hybrid algorithm variants, a feature augmentation algorithm that interprets article citations as collaborative recommendations.

5.3 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.

5.3.1 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 $\langle score, k\rangle$ for each of its n constituting recommenders rec_k:

$$rec_{mixed}(u,i) = \bigcup_{k=1}^{n} \langle rec_k(u,i), k\rangle \tag{5.3}$$

The top-scoring items for each recommender are then displayed to the user next to each other, as in Burke et al. (1997) and Wasfi (1999). However, when composing the different results into a single entity, such as a television viewing schedule, some form of conflict resolution is required. In the personalized television application domain, Cotter and Smyth (2000) apply predefined precedence rules between different recommender functions. Zanker et al. (2007) describe another form of a mixed hybrid, which merges the results of several recommendation systems. It proposes bundles of recommendations from different product categories in the tourism domain, in which for each category a separate recommender is employed. A recommended bundle consists, for instance, of accommodations, as well as sport and leisure activities, that are derived by separate recommender systems. A Constraint Satisfaction Problem (CSP) solver is employed to resolve conflicts, thus ensuring that only consistent sets of items are bundled together according to domain constraints such as "activities and accommodations must be within 50 km of each other".

5.3.2 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 rec_k with associated relative weights β_k:

$$rec_{weighted}(u,i) = \sum_{k=1}^{n} \beta_k \times rec_k(u,i) \tag{5.4}$$

where item scores need to be restricted to the same range for all recommenders and $\sum_{k=1}^{n} \beta_k = 1$. 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 $\beta_1 = \beta_2 = 0.5$. The weighted hybrid recommender rec_w thus produces a new ranking by combining the scores from rec_1 and rec_2. 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	

If the weighting scheme is to remain static, the involved recommenders must also produce recommendations of the same relative quality for all user and item combinations. To estimate weights, an empirical bootstrapping approach can be taken. For instance, Zanker and Jessenitschnig (2009a) conducted a sensitivity analysis between a collaborative and a knowledge-based recommender in the cigar domain to identify the optimum weighting scheme. The P-Tango system (Claypool et al. 1999) is another example of such a system, blending the output of a content-based and a collaborative recommender in the news domain. This approach uses a dynamic weighting scheme and will be explained in detail in the following section. Starting from a uniform distribution, it dynamically adjusts the relative weights for each user to minimize the predictive error in cases in which user ratings are available. Furthermore, it adapts the weights on a per-item basis to correctly reflect the relative strengths of each prediction algorithm. For instance, the collaborative filtering gains weight if the item has been rated by a higher number of users.

We explain such a dynamic weighting approach by returning to the initial example. Let us assume that Alice purchased Item1 and Item4, which we will interpret as positive unary ratings. We thus require a weighting that minimizes a goal metric such as the mean absolute error (MAE) of predictions of a user for her rated items R (see Chapter 2).

$$MAE = \frac{\sum_{r_i \in R} \sum_{k=1}^{n} \beta_k \times |rec_k(u,i) - r_i|}{|R|} \tag{5.5}$$

If we interpret Alice's purchases as very strong relevance feedback for her interest in an item, then the set of Alice's actual ratings R will contain $r_1 = r_4 = 1$. Table 5.7 summarizes the absolute errors of rec_1 and rec_2's predictions (denoted in Table 5.6) for different weighting parameters. Table 5.7 shows that the MAE improves as rec_2 is weighted more strongly. However, when examining the situation more closely, it is evident that the weight assigned to

Table 5.7. *Dynamic weighting parameters, absolute errors, and MAEs for user Alice.*

β_1	β_2	item	r_i	rec_1	rec_2	error	MAE
0.1	0.9	Item1	1	0.5	0.8	0.23	
		Item4	1	0.1	0	0.99	**0.61**
0.3	0.7	Item1	1	0.5	0.8	0.29	
		Item4	1	0.1	0	0.97	0.63
0.5	0.5	Item1	1	0.5	0.8	0.35	
		Item4	1	0.1	0	0.95	0.65
0.7	0.3	Item1	1	0.5	0.8	0.41	
		Item4	1	0.1	0	0.93	0.67
0.9	0.1	Item1	1	0.5	0.8	0.47	
		Item4	1	0.1	0	0.91	0.69

rec_1 should be strengthened. Both rated items are ranked higher by rec_1 than by rec_2 – Item1 is first instead of second and Item4 is third, whereas rec_2 does not recommend Item4 at all. Thus, when applying a weighted strategy, one must ensure that the involved recommenders assign scores on comparable scales or apply a transformation function beforehand. Obviously, the assignment of dynamic weighting parameters stabilizes as more rated items are made available by users. In addition, alternative error metrics, such as mean squared error or rank metrics, can be explored. Mean squared error puts more emphasis on large errors, whereas rank metrics focus not on recommendation scores but on ranks.

In the extreme case, dynamic weight adjustment could be implemented as a switching hybrid. There, the weights of all but one dynamically selected recommenders are set to 0, and the output of a single remaining recommender is assigned the weight of 1.

5.3.3 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:

$$\exists_1 k : 1 \ldots n \;\; rec_{switching}(u, i) = rec_k(u, i) \tag{5.6}$$

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 (Billsus and Pazzani 2000). There, two content-based variants and a collaborative strategy are employed in an ordered manner to recommend news articles. First, a content-based nearest neighbor recommender is used. If it does not find any closely related articles, a collaborative filtering system is invoked to make cross-genre propositions; finally, a naive Bayes classifier finds articles matching the long-term interest profile of the user. Zanker and Jessenitschnig (2009a) proposed a switching strategy that actually switches between two hybrid variants of collaborative filtering and knowledge-based recommendation. If the first algorithm, a cascade hybrid, delivers fewer than *n* recommendations, the hybridization component switches to a weighted variant as a fallback strategy. Even more adaptive switching criteria can be thought of that could even take contextual parameters such as users' intentions or expectations into consideration for algorithm selection. For instance, van Setten (2005) proposed the domain-independent Duine framework that generalizes the selection task of a prediction strategy and discusses several machine learning techniques in that context. To summarize, the quality of the switching mechanism is the most crucial aspect of this hybridization variant.

5.4 Pipelined hybridization design

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.

5.4.1 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 rec_1 represents the recommendation function of the first technique and rec_n 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 (5.7).

$$rec_{cascade}(u, i) = rec_n(u, i) \tag{5.7}$$

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. One such hybridization step that switches to weighted strategy was proposed by Zanker and Jessenitschnig (2009a).

As knowledge-based recommenders produce recommendation lists that are either unsorted or contain many ties among items' scores, cascading them with another technique to sort the results is a natural choice. EntreeC is a knowledge-based restaurant recommender that is cascaded with a collaborative filtering algorithm to recommend restaurants (Burke 2002b). However, in contrast to the definition of cascade hybrid given here, EntreeC uses only the second recommender to break ties. Advisor Suite is a domain independent knowledge-based recommender shell discussed in Chapter 4 (Felfernig et al. 2006–07). It includes an optional utility-based sorting scheme that can further refine recommendations in a cascade design.

5.4.2 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 Δ that has been built by its predecessor. However, in all reported systems so far, n has always been 2.

$$rec_{meta\text{–}level}(u, i) = rec_n(u, i, \Delta_{rec_{n-1}}) \tag{5.8}$$

For instance, the Fab system (Balabanović and Shoham 1997) 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. Pazzani (1999b) referred to this approach as *collaboration via content* and presented a small user study based on restaurant recommendation. It showed that the hybrid variant performs better than base techniques, especially when users have only a few items in common. Zanker (2008) evaluated a further variant of meta-level hybridization that combines collaborative filtering with knowledge-based recommendation. The hybrid generates binary user preferences of the form $a \rightarrow b$, where a represents a user requirement and b a product feature. If, for example, a user of the cigar advisor described by Zanker (2008) were looking for a gift and finally bought a cigar of the brand Montecristo, then the constraint $for\_whom = \text{"}gift\text{"} \rightarrow brand = \text{"}Montecristo\text{"}$ becomes part of the user's profile. When computing a recommendation, a collaborative filtering step retrieves all such constraints from a user's peers. A knowledge-based recommender finally applies these restrictions to the product database and derives item propositions. An evaluation showed that this approach produced more successful predictions than a manually crafted knowledge base or an impersonalized application of all generated constraints.

Golovin and Rahm (2004) applied a reinforcement learning (RL) approach for exploiting context-aware recommendation rules. The authors exploited different top-N recommendation procedures as well as sequence patterns and frequent-item sets to generate weighted recommendation rules that are used by a reinforcement learning component to make predictions. Rules specify which item should be presented in which situation, where the latter is characterized by the product content and the user model, including contextual parameters such as daytime or season of the year. Thus, RL acts as a principal recommender that adjusts the weights of the recommendation rule database based on user feedback.

5.5 Discussion and summary

In this chapter we discussed the opportunities for combining different algorithm variants and presented a taxonomy for hybridization designs. In summary, no single hybridization variant is applicable in all circumstances, but it is well accepted that all base algorithms can be improved by being hybridized with other techniques. For instance, in the Netflix Prize competition, the winners employed a weighted hybridization strategy in which weights were determined by regression analysis (Bell et al. 2007). Furthermore, they adapted the weights based on particular user and item features, such as number of rated items, that can be classified as a switching hybrid that changes between different weighted hybrids according to the presented taxonomy of hybridization variants.

One of the main reasons that little research focuses on comparing different recommendation strategies and especially their hybrids is the lack of appropriate datasets. Although collaborative movie recommendations or content-based news recommenders are comparably well researched application domains, other application domains for recommender systems and algorithm paradigms receive less attention. Therefore, no empirically backed conclusions about the advantages and disadvantages of different hybridization variants can be drawn, but, depending on the application domain and problem type, different variants should be explored and compared. Nevertheless, enhancing an existing recommendation application by exploiting additional knowledge sources will nearly always pay off. With respect to the required engineering effort, the following can be said.

Monolithic designs are advantageous if little additional knowledge is available for inclusion on the feature level. They typically require only some additional preprocessing steps or minor modifications in the principal algorithm and its data structures.

Parallelized designs are the least invasive to existing implementations, as they act as an additional postprocessing step. Nevertheless, they add some additional runtime complexity and require careful matching of the recommendation scores computed by the different parallelized algorithms.

Pipelined designs are the most ambitious hybridization designs, because they require deeper insight into algorithm's functioning to ensure efficient runtime computations. However, they typically perform well when two antithetic recommendation paradigms, such as collaborative and knowledge-based, are combined.

5.6 Bibliographical notes

Only a few articles focus specifically on the hybridization of recommendation algorithms in general. The most comprehensive work in this regard is Burke's article, "Hybrid recommender systems: Survey and experiments", which appeared in *User Modeling and User-Adapted Interaction* in 2002. It developed the taxonomy of recommendation paradigms and hybridization designs that guided this chapter, and is the most referenced article in this respect. A revised version appeared as a chapter in the Springer state-of-the-art survey *The Adaptive Web* by the same author in 2007. It not only constitutes a comprehensive source of reference for published works on hybrid algorithms, but also includes the most extensive comparative evaluation of different hybrid algorithm variants. Burke (2007) compared forty one different algorithms based on the Entree dataset (Burke 1999). In contrast to many earlier comparative studies on the movie domain (Balabanović and Shoham 1997, Pazzani 1999b, Sarwar et al. 2000b), the Entree dataset also allows the exploration of knowledge-based algorithm variants.

Adomavicius and Tuzhilin (2005) provide an extensive state-of-the-art survey on current recommender systems' literature that includes a taxonomy that differentiates between collaborative and content-based recommenders and hybrid variants thereof; however, it lacks a discussion about knowledge-based algorithms.

Zanker et al. (2007) present a recent evaluation of several algorithm variants that compares knowledge-based variants with collaborative ones on a commercial dataset from the cigar domain. These experiments were further developed by Zanker and Jessenitschnig (2009a) with the focus being placed on explicit user requirements, such as keywords and input, to conversational requirements elicitation dialogs as the sole type of user feedback. They explored weighted, switching, and cascade hybridization variants of knowledge-based and collaborative recommendation paradigms.