Chapter 6

Ensemble-Based and Hybrid Recommender Systems

"*What's better, a poetic intuition or an intellectual work? I think they complement each other.*"– Manuel Puig

6.1 Introduction

In the previous chapters, we discussed three different classes of recommendation methods. Collaborative methods use the ratings of a *community* of users in order to make recommendations, whereas content-based methods use the ratings of a *single* user in conjunction with attribute-centric item descriptions to make recommendations. Knowledge-based methods require the explicit specification of user requirements to make recommendations, and they do not require any historical ratings at all. Therefore, these methods use different sources of data, and they have different strengths and weaknesses. For example, knowledge-based systems can address cold-start issues much better than either content-based or collaborative systems because they do not require ratings. On the other hand, they are weaker than content-based and collaborative systems in terms of using *persistent personalization* from historical data. If a different user enters the same requirements and data in a knowledge-based interactive interface, she might obtain exactly the same result.

All these models seem rather restrictive in isolation, especially when multiple sources of data are available. In general, one would like to make use of all the knowledge available in different data sources and also use the algorithmic power of various recommender systems to make robust inferences. Hybrid recommender systems have been designed to explore these possibilities. There are three primary ways of creating hybrid recommender systems:

C.C. Aggarwal, *Recommender Systems: The Textbook*,
DOI 10.1007/978-3-319-29659-3_6

1. *Ensemble design:* In this design, results from off-the-shelf algorithms are combined into a single and more robust output. For example, one might combine the rating outputs from a content-based and a collaborative recommender into a single output. A significant variation exists in terms of the specific methodologies used for the combination process. The basic principle at work is not very different from the design of ensemble methods in many data mining applications such as clustering, classification, and outlier analysis.

   Ensemble design can be formalized as follows. Let $\hat{R_k}$ be an $m \times n$ matrix containing the *predictions* of the m users for the n items by the kth algorithm, where $k \in \{1 \ldots q\}$. Therefore, a total of q different algorithms are used to arrive at these predictions. The (u, j)th entry of $\hat{R_k}$ contains the predicted rating of user u for item j by the kth algorithm. Note that the observed entries of the original ratings matrix R are replicated in each $\hat{R_k}$, and only the unobserved entries of R vary in different $\hat{R_k}$ because of the different predictions of different algorithms. The final result of the algorithm is obtained by combining the predictions $\hat{R_1} \ldots \hat{R_q}$ into a single output. This combination can be performed in various ways, such as the computation of the weighted average of the various predictions. Furthermore, in some *sequential* ensemble algorithms, the prediction $\hat{R_k}$ may depend on the results of the previous component $\hat{R_{k-1}}$. In yet other cases, the outputs may not be directly combined. Rather, the output of one system is used as an input to the next as a set of content *features*. The common characteristics of all these systems are that (a) they use *existing* recommenders in *off-the-shelf* fashion, and (b) they produce a unified score or ranking.

2. *Monolithic design:* In this case, an integrated recommendation algorithm is created by using various data types. A clear distinction may sometimes not exist between the various parts (e.g., content and collaborative) of the algorithm. In other cases, existing collaborative or content-based recommendation algorithms may need to be modified to be used within the overall approach, even when there are clear distinctions between the content-based and collaborative stages. Therefore, this approach tends to integrate the various data sources more tightly, and one cannot easily view individual components as off-the-shelf black-boxes.

3. *Mixed systems:* Like ensembles, these systems use multiple recommendation algorithms as black-boxes, but the items recommended by the various systems are presented together side by side. For example, the television program for a whole day is a composite entity containing multiple items. It is meaningless to view the recommendation of a single item in isolation; rather, it is the combination of the items that creates the recommendation.

Therefore, the term"hybrid system" is used in a broader context than the term "ensemble system." All ensemble systems are, by definition, hybrid systems, but the converse is not necessarily true.

Although hybrid recommender systems usually combine the power of different types of recommenders (e.g., content- and knowledge-based), there is no reason why such systems cannot combine models of the same type. Since content-based models are essentially text classifiers, it is well known that a wide variety of ensemble models exist to improve the accuracy of classification. Therefore, any classification-based ensemble system can be used to improve the effectiveness of content-based models. This argument also extends to collaborative recommender models. For example, one can easily combine the predicted results of

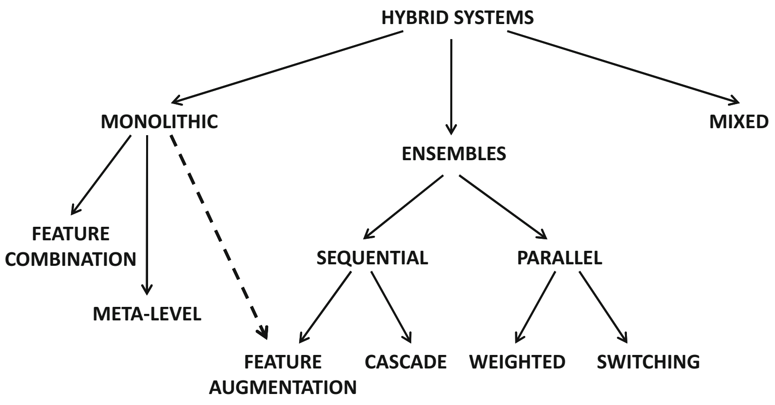


Figure 6.1: The taxonomy of hybrid systems

a latent factor model with those of a neighborhood model to obtain more accurate recommendations [266]. In fact, both[1] the winning entries in the Netflix Prize contest, referred to as "*Bellkor's Pragmatic Chaos*" [311] and "*The Ensemble*" [704], were ensemble systems.

At a broader level, hybrid recommender systems are closely related to the field of ensemble analysis in classification. For example, collaborative models are generalizations of classification models, as discussed in the introduction to Chapter 3. As we will discuss in section 6.2 of this chapter, the theoretical underpinnings of ensemble analysis in classification are similar to those in collaborative filtering. Therefore, this chapter will also focus on how the recommendation approach can be used to improve the effectiveness of collaborative recommender systems in much the same way as one might use ensembles in the field of data classification.

According to Burke [117], hybrid recommender systems can be classified into the following categories:

1. *Weighted:* In this case, the scores of several recommender systems are combined into a single unified score by computing the weighted aggregates of the scores from individual ensemble components. The methodology for weighting the components may be heuristic, or it might use formal statistical models.

2. *Switching:* The algorithm switches between various recommender systems depending on current needs. For example, in earlier phases, one might use a knowledge-based recommender system to avoid cold-start issues. In later phases, when more ratings are available, one might use a content-based or collaborative recommender. Alternatively, the system might adaptively select the specific recommender that provides the most accurate recommendation at a given point in time.

3. *Cascade:* In this case, one recommender system refines the recommendations given by another. In generalized forms of cascades, such as *boosting*, the training process of one recommender system is *biased* by the output of the previous one, and the overall results are combined into a single output.

[1]Both entries were tied on the error rate. The award was given to the former because it was submitted 20 minutes earlier.

4. *Feature augmentation:* The output of one recommender system is used to create input features for the next. While the cascade hybrid successively refines the recommendations of the previous system, the feature augmentation approach treats then as features as *input* for the next system. This approach shares a number of intuitive similarities with the notion of *stacking*, which is commonly used in classification. In stacking, the outputs of one classifier are used as features for the next. Because the different recommenders are (generally) used as off-the-shelf black-boxes, the approach is still an ensemble method (in most cases) rather than a monolithic method.

5. *Feature combination:* In this case, the features from different data sources are combined and used in the context of a single recommender system. This approach can be viewed as a monolithic system, and therefore it is not an ensemble method.

6. *Meta-level:* The model used by one recommender system is used as input to another system. The typical combination used is that of a content-based and collaborative system. The collaborative system is modified to use the content features to determine peer groups. Then, the ratings matrix is used in conjunction with this peer group to make predictions. Note that this approach needs to modify the collaborative system to use a content matrix for finding peer groups, although the final predictions are still performed with the ratings matrix. Therefore, the collaborative system needs to be modified, and one cannot use it in an off-the-shelf fashion. This makes the meta-level approach a monolithic system rather than an ensemble system. Some of these methods are also referred to as "collaboration via content" because of the way in which they combine collaborative and content information.

7. *Mixed:* Recommendations from several engines are *presented* to the user at the same time. Strictly speaking, this approach is not an ensemble system, because it does not explicitly combine the scores (of a particular item) from the various components. Furthermore, this approach is often used when the recommendation is a *composite* entity in which multiple items can be recommended as a related set. For example, a composite television program can be constructed from the various recommended items [559]. Therefore, this approach is quite different from all the aforementioned methods. On the one hand, it does use other recommenders as black-boxes (like ensembles), but it does not combine the predicted ratings of the same item from different recommenders. Therefore, mixed recommenders cannot be viewed either as monolithic or ensemble-based methods and are classified into a distinct category of their own. The approach is most relevant in complex item domains, and it is often used in conjunction with knowledge-based recommender systems.

The first four of the aforementioned categories are ensemble systems, the next two are monolithic systems, and the last one is a mixed system. The last category of mixed systems cannot be neatly categorized either as a monolithic or an ensemble system, because it presents multiple recommendations as a composite entity. A hierarchical categorization of these various types of systems is shown in Figure 6.1. Although we have used the higher level categorization of parallel and sequential[2] systems, as introduced by [275], we emphasize that our categorization of Burke's original set of six categories is slightly different from that of [275]. Unlike the taxonomy in [275], which classifies meta-level systems as sequential methods, we view meta-level systems as monolithic because one cannot use off-the-shelf recommendation algorithms, as in the case of a true ensemble. Similarly, the work in [275]

[2] This is also referred to as a *pipelined* system [275].

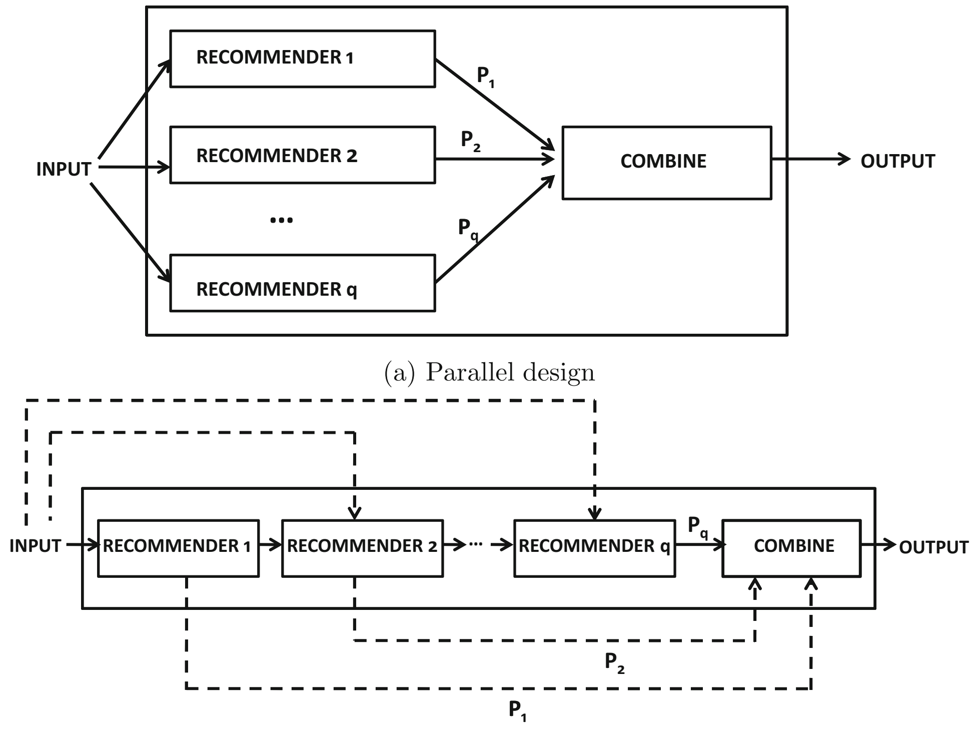
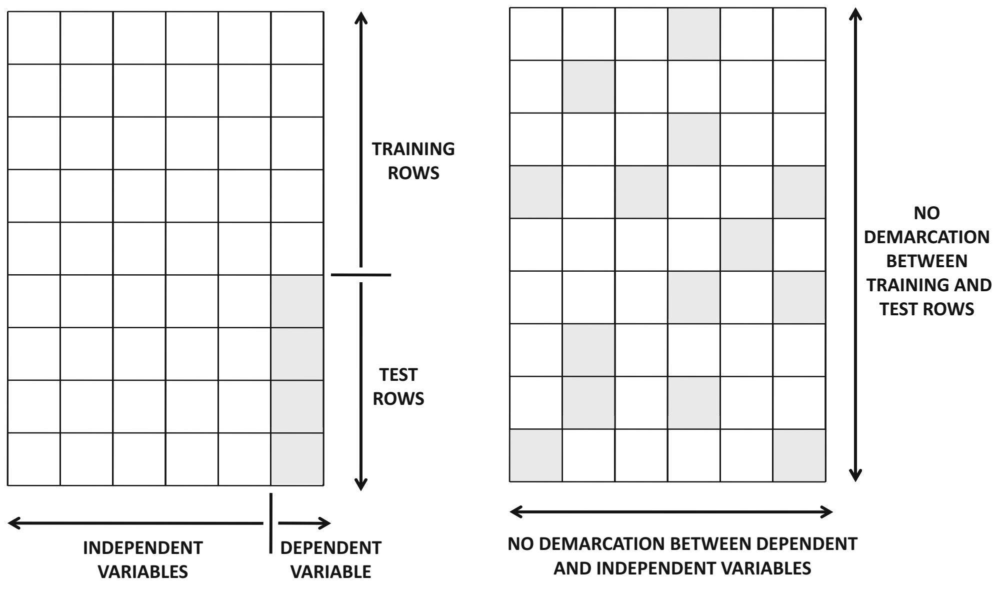


Figure 6.2: Parallel and sequential ensembles

views feature augmentation hybrids as monolithic systems. Although the individual recommenders are combined together in a more complex way in feature augmentation hybrids, the individual recommenders are still used as off-the-shelf black-boxes for the large part. This is the primary distinguishing characteristic of an ensemble system from a monolithic system, and the approach is highly reminiscent of stacking methods in classification. Therefore, we view feature augmentation hybrids as ensemble systems rather than monolithic systems. However, in some cases of feature augmentation hybrids, minor changes are required to the off-the-shelf recommender. In such cases, these systems may be technically considered to have a monolithic design. We have shown this possibility with a dotted line in Figure 6.1.

Aside from the monolithic and mixed methods, which are not truly ensembles, one can view all ensemble methods as having either sequential or parallel designs [275]. In parallel designs, the various recommenders function independently of one another, and the predictions of the individual recommenders are combined at the very end. The weighted and switching methods can be viewed as parallel designs. In sequential designs, the output of one recommender is used as an input to the other. The cascade and meta-level systems can be viewed as examples of sequential methods. A pictorial illustration of the combination process in sequential and parallel systems is shown in Figure 6.2. In this chapter, we will provide a detailed discussion of several recommender systems in each of these categories, although we will use Burke's lower-level taxonomy [117] to organize the discussion.

This chapter is organized as follows. In section 6.2, we discuss the classification perspective of ensemble-based recommender systems. We also explore the level to which the existing theories and methodologies for ensemble methods in the field of classification also


TRAINING ROWS
TEST ROWS
INDEPENDENT VARIABLES
DEPENDENT VARIABLE
NO DEMARCATION BETWEEN TRAINING AND TEST ROWS
NO DEMARCATION BETWEEN DEPENDENT AND INDEPENDENT VARIABLES

(a) Classification (b) Collaborative filtering

Figure 6.3: Revisiting Figure 1.4 of Chapter 1. Comparing the traditional classification problem with collaborative filtering. Shaded entries are missing and need to be predicted.

apply to recommender systems. In section 6.3, a number of different examples of weighted hybrids are discussed. In section 6.4, a number of switching hybrids are discussed. Cascade hybrids are discussed in section 6.5, whereas feature augmentation hybrids are discussed in section 6.6. Meta-level hybrids are discussed in section 6.7. Feature combination methods are introduced in section 6.8. Mixed systems are discussed in section 6.9. A summary is given in section 6.10.

6.2 Ensemble Methods from the Classification Perspective

Ensemble methods are commonly used in the field of data classification to improve the robustness of learning algorithms. As we will discuss below, much of this theory also applies to various forms of recommender systems. For example, content-based recommender systems are often straightforward applications of text classification algorithms. Therefore, a direct application of existing ensemble methods in classification is usually sufficient to obtain high-quality results.

As discussed in Chapter 1, collaborative filtering is a generalization of the problem of data classification. We have replicated Figure 1.4 of Chapter 1 in Figure 6.3 to illustrate the relationship between the two problems. It is evident from Figure 6.3(a) that the feature variables and class variable are clearly demarcated in classification. The main distinguishing features of collaborative filtering from classification are that the feature variables and the class variable are not clearly demarcated in the former and that the missing entries may occur in any column or row. The fact that missing entries may occur in any row implies that training and test instances are not clearly demarcated, either. A salient question arises as to whether the bias-variance theory developed in the field of classification [242] also applies to recommender systems. Repeated experiments [266, 311] have shown that combining multiple collaborative recommender systems often leads to more accurate results. This is because the bias-variance theory, which is designed for classification, also applies to the collaborative filtering scenario. This means that many traditional ensemble techniques from classification can also be generalized to collaborative filtering. Nevertheless, because of the fact that the missing entries might occur in any row or column of the data, it is sometimes algorithmically challenging to generalize the ensemble algorithms for classification to collaborative filtering.

We first introduce the bias-variance trade-off as it applies to the field of data classification. Consider a simplified classification or regression model, in which a specific field needs to be predicted, as shown in Figure 6.3(a). It can be shown that the error of a classifier in predicting the dependent variable can be decomposed into three components:

1. *Bias:* Every classifier makes its own modeling assumptions about the nature of the decision boundary between classes. For example, a linear SVM classifier assumes that the two classes may be separated by a linear decision boundary. This is, of course, not true in practice. In other words, any given linear support vector machine will have an inherent *bias*. When a classifier has high bias, it will make *consistently incorrect* predictions over particular choices of test instances near the incorrectly modeled decision-boundary, even when different samples of the training data are used for the learning process.

2. *Variance:* Random variations in the choices of the training data will lead to different models. As a result, the dependent variable for a test instance might be inconsistently predicted by different choices of training data sets. Model variance is closely related to overfitting. When a classifier has an overfitting tendency, it will make *inconsistent* predictions for the same test instance over different training data sets.

3. *Noise:* The noise refers to the intrinsic errors in the target class labeling. Because this is an intrinsic aspect of data quality, there is little that one can do to correct it. Therefore, the focus of ensemble analysis is generally on reducing bias and variance.

The expected mean-squared error of a classifier over a set of test instances can be shown to be sum of the bias, variance, and noise. This relationship can be stated as follows:

$$\text{Error} = \text{Bias}^2 + \text{Variance} + \text{Noise} \tag{6.1}$$

It is noteworthy that by reducing either the bias or variance components, one can reduce the overall error of a classifier. For example, classification ensemble methods such as *bagging* [99] reduce the variance, whereas methods such as *boosting* [206] can reduce the bias. It is noteworthy that the only difference between classification and collaborative filtering is that missing entries can occur in any column rather than only in the class variable. Nevertheless, the bias-variance result still holds when applied to the problem of predicting a specific column, whether the other columns are incompletely specified or not. This means that the basic principles of ensemble analysis in classification are also valid for collaborative filtering. Indeed, as we will see later in this chapter, many classical ensemble methods in classification, such as bagging and boosting, have also been adapted to collaborative filtering.

6.3 Weighted Hybrids

Let $R = [r_{uj}]$ be an $m \times n$ ratings matrix. In weighted hybrids, the outputs of various recommender systems are combined using a set of weights. Let $\hat{R}_1 \dots \hat{R}_q$ be the $m \times n$ *completely specified* ratings matrices, in which the unobserved entries of R are predicted by q different algorithms. Note that the entries r_{uj} that are already observed in the original $m \times n$ ratings matrix R are already fixed to their observed values in each prediction matrix $\hat{R}_k$. Then, for a set of weights $\alpha_1 \dots \alpha_q$, the weighted hybrid creates a combined prediction matrix $\hat{R} = [\hat{r}_{uj}]$ as follows:

$$\hat{R} = \sum_{i=1}^{q} \alpha_i \hat{R}_i \tag{6.2}$$

In the simplest case, it is possible to choose $\alpha_1 = \alpha_2 = \dots = \alpha_q = 1/q$. However, it is ideally desired to weight the various systems in a differential way, so as to give greater importance to the more accurate systems. A number of methods exist for such differential weighting. One can also write the aforementioned equation in terms of individual entries of the matrix:

$$\hat{r}_{uj} = \sum_{i=1}^{q} \alpha_i \hat{r}^i_{uj} \tag{6.3}$$

Here $\hat{r}^i_{uj}$ denotes the prediction of the ith ensemble component for user u and item j and $\hat{r}_{uj}$ denotes the final prediction.

In order to determine the optimal weights, it is necessary to be able to evaluate the effectiveness of a particular combination of weights $\alpha_1 \dots \alpha_q$. While this topic will discussed in more detail in Chapter 7, we will provide a simple evaluation approach here for the purpose of discussion. A simple approach is to hold out a small fraction (e.g., 25%) of the known entries in the $m \times n$ ratings matrix $R = [r_{uj}]$ and create the prediction matrices $\hat{R}_1 \dots \hat{R}_q$ by applying the q different base algorithms on the remaining 75% of the entries in R. The resulting predictions $\hat{R}_1 \dots \hat{R}_q$ are then combined to create the ensemble-based prediction $\hat{R}$ according to Equation 6.2. Let the user-item indices (u, j) of these held-out entries be denoted by H. Then, for a given vector $\overline{\alpha} = (\alpha_1 \dots \alpha_q)$ of weights, the effectiveness of a particular scheme can be evaluated using either the mean-squared error (MSE) or the mean absolute error (MAE) of the predicted matrix $\hat{R} = [\hat{r}_{uj}]_{m \times n}$ over the held-out ratings in H:

$$MSE(\overline{\alpha}) = \frac{\sum_{(u,j)\in H} (\hat{r}_{uj} - r_{uj})^2}{|H|}$$

$$MAE(\overline{\alpha}) = \frac{\sum_{(u,j)\in H} |(\hat{r}_{uj} - r_{uj})|}{|H|}$$

These metrics provide an evaluation of a particular combination of coefficients $\alpha_1 \dots \alpha_q$. How can we determine the optimal values of $\alpha_1 \dots \alpha_q$ to minimize these metrics? A simple approach, which works well for the case of MSE, is to use linear regression. It is assumed that the ratings in the held-out set H provide the ground truth values of the dependent variable, and the parameters $\alpha_1 \dots \alpha_q$ are the independent variables. The idea is to select the independent variables so that the mean-squared error of the linear combination is minimized with respect to the known ratings in the held-out set. The basics of the linear regression model are discussed in section 4.4.5 of Chapter 4, albeit in a different context. The main difference here is in terms of how the dependent and independent variables are defined and in terms of how the linear regression problem is formulated. In this case, the independent variables correspond to the rating predictions of various models for the entry (u, j), and the dependent variable corresponds to the value of each predicted rating $\hat{r}_{uj}$ of the ensemble combination in the held-out set H. Therefore, each observed rating in the held-out set provides a training example for the linear regression model. The regression coefficients correspond to the weights of various component models, and they need to be learned from the (held-out) training examples. After the weights have been learned using linear regression, the individual component models are retrained on the entire training set without any held-out entries. The weights, which were learned using the held-out entries, are used in conjunction with these q models. It is important not to forget this final step in order to ensure that the maximum learning is obtained from all the information available in the ratings. The linear regression approach to model combination is discussed in [266]. A related approach, which can make good use of all the knowledge in the training data, is that of *cross-validation*. Cross-validation methods are discussed in Chapter 7.

Although many systems simply average the results of multiple models, the use of regression is important to ensure that the various models are weighted appropriately. Such regression-based algorithms were included among many of the highly performing entries in the Netflix Prize contest [311, 554], and they are closely related to the concept of stacking in data classification.

The linear regression approach is, however, sensitive to presence of noise and outliers. This is because the squared error function is overly influenced by the largest errors in the data. A variety of *robust regression* methods are available, which are more resistant to the presence of noise and outliers. One such method uses the mean absolute error (MAE) as the objective function as opposed to the mean-squared error. The MAE is well known to be more robust to noise and outliers because it does not overemphasize large errors. A common approach is to use gradient descent method to determine the optimal value of the parameter vector $(\alpha_1 \ldots \alpha_q)$ of Equation 6.3. The algorithm starts by setting $\alpha_1 = \alpha_2 = \ldots = \alpha_q = 1/q$. Subsequently, the gradient is computed over the held-out entries in H as follows:

$$\frac{\partial MAE(\overline{\alpha})}{\partial \alpha_i} = \frac{\sum_{(u,j)\in H} \frac{\partial |(\hat{r}_{uj} - r_{uj})|}{\partial \alpha_i}}{|H|} \tag{6.4}$$

The value of $\hat{r}_{uj}$ can be expanded using Equation 6.3, and the partial derivative may be simplified in terms of the ratings of individual ensemble components as follows:

$$\frac{\partial MAE(\overline{\alpha})}{\partial \alpha_i} = \frac{\sum_{(u,j)\in H} \text{sign}(\hat{r}_{uj} - r_{uj}) \hat{r}^i_{uj}}{|H|} \tag{6.5}$$

The gradient can be written in terms of the individual partial derivatives:

$$\overline{\nabla MAE} = \left(\frac{\partial MAE(\overline{\alpha})}{\partial \alpha_1} \ldots \frac{\partial MAE(\overline{\alpha})}{\partial \alpha_q} \right)$$

This gradient is then used to descend through the parameter space $\overline{\alpha}$ with an iterative gradient descent approach as follows:

1. Initialize $\overline{\alpha}^{(0)} = (1/q \ldots 1/q)$ and $t = 0$.

2. **Iterative Step 1:** Update $\overline{\alpha}^{(t+1)} \Leftarrow \overline{\alpha}^{(t)} - \gamma \cdot \overline{\nabla MAE}$. The value of $\gamma > 0$ can be determined using a line search so that the maximum improvement in MAE is achieved.

3. **Iterative Step 2:** Update the iteration index as $t \Leftarrow t + 1$.

4. **Iterative Step 3 (convergence check):** If MAE has improved by at least a minimum amount since the last iteration, then go to iterative step 1.

5. Report $\overline{\alpha}^{(t)}$.

Regularization can be added to prevent overfitting. It is also possible to add other constraints on the various values of α_i such as non-negativity or ensuring that they sum to 1. Such natural constraints improve generalizability to unseen entries. The gradient descent equations can be modified relatively easily to respect these constraints. After the optimal weights have been determined, all ensemble models are retrained on the entire ratings matrix without any held-out entries. The predictions of these models are combined with the use of the weight vector discovered by the iterative approach.

There are other ways of performing parameter searches. A simpler approach is to try several judiciously chosen combinations of parameters on a held-out set of ratings. For example, one might tune the various values of α_i in succession by trying different values and holding the others constant. Such an approach is generally applied to various types of parameter tuning [311], and it can often provide reasonably accurate results. Examples of various search techniques are provided in [162, 659].

These methods can be enhanced further with different types of meta-level content features [65, 66, 554]. These methods are discussed in section 6.8.2. Many of the existing ensemble methods do not use these sophisticated combination schemes. Often, these techniques use a simple average of the predictions of different components. It is particularly important to weight the different components when the predicted utility values are on different scales, or when some of the ensemble components are much more accurate than others. In the following, we will provide specific examples of how different types of models are often combined.

6.3.1 Various Types of Model Combinations

In weighted model combinations, a variety of recommendation engines can be combined. There are typically two forms of model combinations:

1. *Homogeneous data type and model classes:* In this case, different models are applied on the same data. For example, one might apply various collaborative filtering engines such as neighborhood-based methods, *SVD*, and Bayes techniques on a ratings matrix. The results are then aggregated into a single predicted value. Such an approach is robust because it avoids the specific bias of particular algorithms on a given data set even though all the constituent models belong to the same class (e.g., collaborative methods). An example of such a blend is provided in [266]. It was shown in [637], how the combination of an ensemble of three different matrix factorization methods can provide high-quality results. In particular, regularized matrix factorization, non-negative matrix factorization, and maximum margin factorization were used as the ensemble components, and the corresponding results were averaged. An interesting *fusion ensemble*, discussed in [67], uses the same recommendation algorithm for various ensemble components, but with different choices of parameters or algorithmic design choices. For example, different numbers of latent factors may be used in an *SVD* algorithm, different numbers of nearest neighbors may be used in a neighborhood-based algorithm, or the choice of the similarity metric may be varied. A simple average

of the predictions of various systems is used. As shown in [67], this simple approach almost always improved the performance of the base model. An earlier variation of this approach [180] uses ensembles of maximum margin matrix factorization methods but with different parameter settings. The work in [338] combines a user-based and item-based neighborhood algorithm.

2. *Heterogeneous data type and model classes:* In this cases, different *classes of* models are applied to different data sources. For example, one component of the model might be a collaborative recommender that uses a ratings matrix, whereas another component of the model might be a content-based recommender. This approach essentially fuses the power of multiple data sources into the combination process. The idea is to leverage the complementary knowledge in the various data sources in order to provide the most accurate recommendations. For example, the work in [659] combines a collaborative and knowledge-based recommender, whereas the work in [162] combines a content-based and collaborative recommender. When working with different data types, it becomes particularly important to carefully weight the predictions of various ensemble components.

These different forms of models provide excellent flexibility in exploring several types of model combinations.

6.3.2 Adapting Bagging from Classification

As discussed earlier in this chapter, the theoretical results on the bias-variance trade-off also hold for the collaborative filtering problem, because the latter problem is a direct generalization of classification. One of the common weighted combination techniques used in the classification problem is that of bagging. Therefore, this method can be used in collaborative filtering as well. However, the bagging approach needs to be slightly modified in order to adjust for the fact that the collaborative filtering problem is formulated somewhat differently from that of classification. First, we discuss bagging in the context of classification.

The basic idea in bagging is to reduce the variance component of the error in classification. In bagging, q training data sets are created with *bootstrapped sampling.* In bootstrapped sampling, rows of the data matrix are sampled *with replacement* in order to create a new training data set of the same size as the original training data set. This new training data set typically contains many duplicates. Furthermore, it can be shown that the expected fraction of rows from the original data matrix that is not represented in a given bootstrapped sample is given by $1/e$, where e is the base of the natural logarithm. A total of q training models are created with each of the sampled training data sets. For a given test instance, the average prediction from these q models is reported. Bagging generally improves the classification accuracy because it reduces the variance component of the error. A particular variant of bagging, known as *subagging* [111, 112], *subsamples* the rows, rather than sampling with replacement. For example, one can simply use all the distinct rows in a bootstrapped sample for training the models. The bagging and subagging methods can be generalized to collaborative filtering as follows:

1. *Row-wise bootstrapping:* In this case, the rows of the ratings matrix R are sampled with replacement to create a new ratings matrix of the same dimensions. A total of q such ratings matrices $R_1 \dots R_q$ are thus created. Note that rows may be duplicated in the process of sampling, although they are treated as separate rows. An existing collaborative filtering algorithm (e.g., latent factor model) is then applied to each of

the q training data sets. For each training data set, an item rating can be predicted for a user only if that user is represented at least once in the matrix. In such a case, the predicted rating from that ensemble component is the average rating[3] of that item over the duplicate occurrences of that user. The predicted rating is then averaged over all the ensemble components in which that user is present. Note that for reasonably large values of q, each user will typically be present in at least one ensemble component with a high probability value of $1 - (1/e)^q$. Therefore, all users will be represented with high probability.

2. *Row-wise subsampling:* This approach is similar to row-wise bootstrapping, except that the rows are sampled without replacement. The fraction f of rows sampled is chosen randomly from $(0.1, 0.5)$. The number of ensemble components q should be significantly greater than 10 to ensure that all rows are represented. The main problem with this approach is that it is difficult to predict all the entries in this setting, and therefore one has to average over a smaller number of components. Therefore, the benefits of variance reduction are not fully achieved.

3. *Entry-wise bagging:* In this case, the *entries* of the original ratings matrix are sampled with replacement to create the q different ratings matrices $R_1 \dots R_q$. Because many entries may be sampled repeatedly, the entries are now associated with weights. Therefore, a base collaborative filtering algorithm is required that can handle entries with weights. Such algorithms are discussed in section 6.5.2.1. As in the case of row-wise bagging, the predicted ratings are averaged over the various ensemble components.

4. *Entry-wise subsampling:* In entry-wise subsampling, a fraction of the entries are retained at random from the ratings matrix R to create a sampled training data set. Typically, a value of f is sampled from $(0.1, 0.5)$, and then a fraction f of the entries in the original ratings matrix are randomly chosen and retained. This approach is repeated to create q training data sets $R_1 \dots R_q$. Thus, each user and each item is represented in each subsampled matrix, but the number of specified entries in the subsampled matrix is smaller than that in the original training data. A collaborative filtering algorithm (e.g., latent factor model) is applied to each ratings matrix to create a predicted matrix. The final prediction is the simple average of these q different predictions.

In the aforementioned methods, the final step of the ensemble uses a simple average of the predictions rather than a weighted average. The reason for using a simple average is that all model components are created with an identical probabilistic approach and should therefore be weighted equally. In many of these cases, it is important to choose unstable base methods to achieve good performance gains.

Although the aforementioned discussion provides an overview of the various possibilities for variance reduction, only a small subset of these possibilities have actually been explored and evaluated in the research literature. For example, we are not aware of any experimental results on the effectiveness of subsampling methods. Although subsampling methods often provide superior results to bagging in the classification domain [658], their effect on collaborative filtering is difficult to predict in sparse matrices. In sparse matrices, dropping entries could lead to the inability to predict some users or items at all, which can sometimes worsen the overall performance. A discussion of bagging algorithms in the context of collaborative

[3]It is possible for the unspecified values in duplicate rows to predicted differently, even though this is relatively unusual for most collaborative filtering algorithms.

filtering can be found in [67]. In this work, a row-wise bootstrapping approach is used, and duplicate rows are treated as weighted rows. Therefore, the approach assumes that the base predictor can handle weighted rows. As discussed in [67], significant improvements in the error were achieved with bagging, although the approach seemed to be somewhat sensitive to the choice of base predictor. In particular, according to the results in [67], the bagging approach improved the accuracy over most of the base predictors, with the exception of the factorized neighborhood model [72]. This might possibly be a result of a high level of correlation between the predictions of the various bagged models, when the factorized neighborhood method is used. In general, it is desirable to use uncorrelated base models with low bias and high variance in order to extract the maximum benefit from bagging. In cases where bagging does not work because of high correlations across base predictors, it may be helpful to explicitly use *randomness injection*.

6.3.3 Randomness Injection

Randomness injection is an approach that shares many principles of random forests in classification [22]. The basic idea is to take a base classifier and explicitly inject randomness into the classifier. Various methods can be used for injecting the randomness. Some examples [67] are as follows:

1. *Injecting randomness into a neighborhood model:* Instead of using the top-k nearest neighbors (users or items) in a user-based or item-based neighborhood model, the top-$\alpha \cdot k$ neighbors are selected for $\alpha \gg 1$. Then, k elements are randomly selected from these $\alpha \cdot k$ neighbors. This approach can, however, be shown to be an indirect variant of row-wise subsampling at factor $1/\alpha$. The average prediction from the various components is returned by the approach.

2. *Injecting randomness into a matrix factorization model:* Matrix factorization methods are inherently randomized methods because they perform gradient descent over the solution space after randomly initializing the factor matrices. Therefore, by choosing different initializations, different solutions are often obtained. The combinations of these different solutions often provide more accurate results.

A simple average of the predictions of the different components is returned by the randomized ensemble. Like random forests, this approach can reduce the variance of the ensemble without affecting the bias significantly. In many cases, this approach works quite well where bagging does not work because of a high level of correlation between various predictors. As shown in [67], the randomness injection approach works quite well when the factorized neighborhood model is used as the base predictor [72]. It is noteworthy that the bagging approach does not work very well in the case of the factorized neighborhood model.

6.4 Switching Hybrids

Switching hybrids are used most commonly in recommender systems in the context of the problem of *model selection*, but they are often not formally recognized as hybrid systems. The original motivation for switching systems [117] was to handle the cold-start problem, where a particular model works better in earlier stages when there is a paucity of available data. However, in later stages, a different model is more effective, and therefore one switches to the more effective model.

It is also possible to view switching models in the more general sense of *model selection*. For example, even the parameter selection step of most recommender models requires the running of the model over multiple parameter values and then selecting the optimal one. This particular form of model selection is adapted from the classification literature, and it is also referred to as the *bucket-of-models*. In the following, we discuss both these types of hybrids.

6.4.1 Switching Mechanisms for Cold-Start Issues

Switching mechanisms are often used to handle the cold-start problem, in which one recommender performs better with less data, whereas the other recommender performs better with more data. One might use a knowledge-based recommender, when few ratings are available because knowledge-based recommender systems can function without any ratings, and they are dependent on user specifications of their needs. However, as more ratings become available, one might switch to a collaborative recommender. One can also combine content-based and collaborative recommenders in this way, because content-based recommenders can work well for new items, whereas collaborative recommenders cannot effectively give recommendations for new items.

The work in [85] proposes the *Daily Learner* system in which various recommenders are used in an ordered strategy. If sufficient recommendations are not found by earlier recommenders, then later recommenders are used. In particular, the work in [85] uses two content-based recommenders and a single collaborative recommender. First, a nearest neighbor content classifier is used, followed by a collaborative system, and finally a naive Bayes content classifier is used to match with the long-term profile. This approach does not fully address the cold-start problem because all the underlying learners need some amount of data. Another work [659] combines hybrid versions of collaborative and knowledge-based systems. The knowledge-based system provides more accurate results during the cold-start phase, whereas the collaborative system provides more accurate results in later stages. Incorporating knowledge-based systems is generally more desirable for handling the cold-start problem.

6.4.2 Bucket-of-Models

In this approach, a fraction (e.g., 25% to 33%) of the specified entries in the ratings matrix are held out, and various models are applied to the resulting matrix. The held-out entries are then used to evaluate the effectiveness of the model in terms of a standard measure, such as the MSE or the MAE. The model that yields the lowest MSE or MAE is used as the relevant one. This approach is also commonly used for parameter tuning. For example, each model may correspond to a different value of the parameter of the algorithm, and the value providing the best result is selected as the relevant one. Once the relevant model has been selected, it is retrained on the *entire* ratings matrix, and the results are reported. Instead of using a hold-out approach, a different technique known as *cross-validation* is also used. You will learn more about hold-out and cross-validation techniques in Chapter 7. The bucket-of-models is the single most useful ensemble approach in recommender systems, although it is rarely recognized as an ensemble system unless the different models are derived from heterogeneous data types. When the bucket-of-models is used in the context of a dynamically changing ratings matrix, it is possible for the system to switch from one component to the other. However, when it is used for static data, the system can also be viewed as a special case of weighted recommenders in which the weight of one component is set to 1, and the weights of the remaining components are set to 0.

6.5 Cascade Hybrids

In Burke's original work [117], cascade hybrids were defined in a somewhat narrow way, in which each recommender actively refines the recommendations made by the previous recommender. Here, we take a broader view of cascade hybrids in which a recommender is allowed to use recommendations of the previous recommender in any way (beyond just direct refinement), and then combine the results to make the final recommendation. This broader definition encompasses larger classes of important hybrids, such as boosting, which would not otherwise be included in any of the categories of hybrids. Correspondingly, we define two different categories of cascade recommenders.

6.5.1 Successive Refinement of Recommendations

In this approach, a recommender system successively refines the output of recommendations from the previous iteration. For example, the first recommender can provide a rough ranking and also eliminate many of the potential items. The second level of recommendation then uses this rough ranking to further refine it and break ties. The resulting ranking is presented to the user. An example of such a recommender system is *EntreeC* [117], which uses knowledge of the user's stated interests to provide a rough ranking. The resulting recommendations are then partitioned into buckets of roughly equal preference. The recommendations within a bucket are therefore considered ties at the end of the first stage. A collaborative technique is used to break the ties and rank the recommendations within each bucket. The first knowledge-based recommender is clearly of higher priority because the second-level recommender cannot change the recommendations made at the first level. The other observation is that the second level recommender is much more efficient because it needs to focus only on the ties within each bucket. Therefore, the item-space of each application of the second recommender is much smaller.

6.5.2 Boosting

Boosting has been used popularly in the context of classification [206] and regression [207]. One of the earliest methods for boosting was the *AdaBoost* algorithm [206]. The regression variant of this algorithm is referred to as *AdaBoost.RT* [207]. The regression variant is more relevant to collaborative filtering because it easier to treat ratings as numeric attributes. In traditional boosting, a sequence of training rounds is used with weighted training examples. The weights in each round are modified depending on the performance of the classifier in the previous round. Specifically, the weights on the training examples with error are increased, whereas the weights on the correctly modeled examples are reduced. As a result, the classifier is biased towards correctly classifying the examples that it was unable to properly classify in the previous round. By using several such rounds, one obtains a sequence of classification models. For a given test instance, all models are applied to it, and the weighted prediction is reported as the relevant one.

Boosting needs to be modified to work for collaborative filtering, in which there is no clear demarcation between the training and test rows, and there is also no clear distinction between the dependent and independent columns. A method for modifying boosting for collaborative filtering is proposed in [67]. Unlike classification and regression modeling, in which weights are associated with rows, the training example weights in collaborative filtering are associated with individual *ratings*. Therefore, if the set S represents the set of observed ratings in the training data, then a total of $|S|$ weights are maintained. Note that

S is a set of positions (u, j) in the $m \times n$ ratings matrix R, such that r_{uj} is observed. It is also assumed that the base collaborative filtering algorithm has the capacity to work with weighted ratings (cf. section 6.3). In each iteration, the weights of each of these ratings are modified depending on how well the collaborative filtering algorithm is able to predict that particular entry.

The overall algorithm is applied for a total of T iterations. In the tth iteration, the weight associated with the (u, j)th entry of the ratings matrix is denoted by $W_t(u, j)$. The algorithm starts by equally weighting each entry and predicts all ratings using a baseline model. The prediction of an entry $(u, j) \in S$ is said to be "incorrect," if the predicted rating $\hat{r}_{uj}$ varies from the actual rating r_{uj} by at least a predefined amount δ. The error rate ϵ_t in the tth iteration is computed as the fraction of specified ratings in S for which the predicted value is incorrect, according to this definition. The weights of correctly predicted examples are reduced by multiplying them with ϵ_t, whereas the weights of the incorrectly predicted examples stay unchanged. In each iteration, the weights are always normalized to sum to 1. Therefore, the *relative* weights of incorrectly classified entries always increase across various iterations. The baseline model is applied again to the re-weighted data. This approach is repeated for T iterations, in order to create T different predictions for the unspecified entries. The weighted average of these T different predictions is used as the final prediction of an entry, where the weight of the tth prediction is $\log\left(\frac{1}{\epsilon_t}\right)$. It is noteworthy that the weight update and model combination rules in [67] are slightly different from those used in classification and regression modeling. However, there are very few studies in this area, beyond the work in [67], on using boosting methods for collaborative filtering. It is conceivable that the simple strategies proposed in [67] can be further improved on with experimentation.

6.5.2.1 Weighted Base Models

The boosting and bagging methods require the use of weighted base models, in which entries are associated with weights. In this section, we show how existing collaborative filtering models can be modified so that they can work with weights.

Let us assume that the weight w_{uk} be associated with a particular entry in the ratings matrix for user u and item k. It is relatively straightforward to modify existing models to work with weights on the entries:

1. *Neighborhood-based algorithms:* The average rating of a user is computed in a weighted way for mean-centering the ratings. Both the Pearson and the cosine measures can be modified to take weights into account. Therefore, Equation 2.2 of Chapter 2 can be modified as follows to compute the Pearson coefficient between users u and v:

$$\text{Pearson}(u, v) = \frac{\sum_{k \in I_u \cap I_v} \max\{w_{uk}, w_{vk}\} \cdot (r_{uk} - \mu_u) \cdot (r_{vk} - \mu_v)}{\sqrt{\sum_{k \in I_u \cap I_v} w_{uk}(r_{uk} - \mu_u)^2} \cdot \sqrt{\sum_{k \in I_u \cap I_v} w_{vk}(r_{vk} - \mu_v)^2}} \quad (6.6)$$

The reader should refer to section 2.3 of Chapter 2 for the details of the notations. A different way[4] of modifying the measure is as follows:

$$\text{Pearson}(u, v) = \frac{\sum_{k \in I_u \cap I_v} w_{uk} \cdot w_{vk} \cdot (r_{uk} - \mu_u) \cdot (r_{vk} - \mu_v)}{\sqrt{\sum_{k \in I_u \cap I_v} w_{uk}^2(r_{uk} - \mu_u)^2} \cdot \sqrt{\sum_{k \in I_u \cap I_v} w_{vk}^2(r_{vk} - \mu_v)^2}} \quad (6.7)$$

[4]The work in [67] proposes only the first technique for computing the similarity.

For item-item similarity measures, the adjusted cosine measure can be modified in a similar way. These weighted similarity measures are used both for computing the nearest neighbors and for (weighted) averaging of the ratings in the peer group.

2. *Latent factor models:* Latent factor models are defined as optimization problems in which the sum of the squares of the errors of the specified entries are minimized. In this case, the *weighted* sum of the squares of the optimization problem must be minimized. Therefore, the objective function in section 3.6.4.2 of Chapter 3 can be modified as follows:

$$\text{Minimize } J = \frac{1}{2} \sum_{(i,j) \in S} w_{ij} e_{ij}^2 + \frac{\lambda}{2} \sum_{i=1}^{m} \sum_{s=1}^{k} u_{is}^2 + \frac{\lambda}{2} \sum_{j=1}^{n} \sum_{s=1}^{k} v_{js}^2 \qquad (6.8)$$

Here, $U = [u_{ij}]$ and $V = [v_{ij}]$ are the $m \times k$ and $n \times k$ user-factor and item-factor matrices, respectively. Note the weights associated with the errors on the entries. The corresponding change in the gradient descent method is to weight the relevant updates:

$$u_{iq} \Leftarrow u_{iq} + \alpha(w_{ij} \cdot e_{ij} \cdot v_{jq} - \lambda \cdot u_{iq})$$
$$v_{jq} \Leftarrow v_{jq} + \alpha(w_{ij} \cdot e_{ij} \cdot u_{iq} - \lambda \cdot v_{jq})$$

Many other base collaborative filtering algorithms can be modified to work with weighted entries. These types of weighted base algorithms are useful for many collaborative filtering ensembles, such as boosting and bagging.

6.6 Feature Augmentation Hybrids

The feature augmentation hybrid shares a number of intuitive similarities with the stacking ensemble in classification. In stacking [634], the first level classifier is used to create or augment a set of features for the second level classifier. In many cases, off-the-shelf systems are used like an ensemble. However, in some cases, changes may be required to the component recommender system to work with the modified data, and therefore the hybrid system is not a true ensemble of off-the-shelf systems.

The *Libra* system [448] combines Amazon.com's recommender system with its own Bayes classifier. The approach uses the "related authors" and "related titles" that Amazon generates as features describing the items. Note that Amazon generates these recommendations with the use of a collaborative recommender system. These data are then used in conjunction with a content-based recommender to make the final predictions. Note that any off-the-shelf content-based system can be used in principle, and therefore the approach can be viewed as an ensemble system. The approach in [448] opts for a naive Bayes text classifier. It was found through experiments that the features generated by Amazon's collaborative system were of high quality, and they contributed significantly to better quality recommendations.

Instead of using a collaborative system first, it is also possible to use the content-based system first. The basic idea is to use the content-based system to fill in the missing entries of the ratings matrix so that it is no longer sparse. Thus, the missing entries are estimated by the content-based system to create a denser ratings matrix. These newly added ratings are referred to as *pseudo-ratings.* Then, a collaborative recommender is used on the dense ratings matrix to make rating predictions. Finally, the collaborative prediction is combined with the original content-based prediction in a weighted way to yield the overall prediction of the missing entries in the matrix [431]. The incorporation of missing ratings in the first phase allows for a more robust application of the second phase in terms of similarity computation. However, the similarity computation does need to be modified to give less weight to pseudo-ratings compared to true ratings. This is because pseudo-ratings were inferred, and they might be error-prone.

How can such weights be determined? The weight of a pseudo-rating intuitively represents the algorithm's certainty in the prediction of the first phase, and it is an increasing function of the number of ratings $|I_i|$ of that user. A number of heuristic functions are used to weight various ratings, and the reader is referred to [431] for details. Note that this approach requires modifications to the second phase of collaborative filtering, and off-the-shelf algorithms cannot be used. Such methods can be viewed as monolithic systems.

Feature augmentation has a long history in recommender systems. One of the earliest example of feature augmentation was implemented in the context of the *GroupLens* system [526], in which a knowledge-based system was used to create a database of artificial ratings. The agents, known as filterbots, used specific criteria such as the number of spelling errors or the message size to assign ratings to items, while acting as artificial users. Subsequently, these ratings were used in the context of a collaborative system to make recommendations.

6.7 Meta-Level Hybrids

In a meta-level hybrid, the *model* learned by one recommender is used as input to the next level. An important example of collaboration via content was the early work by Pazzani [475]. A content-based model [363] is constructed that describes the discriminative features predicting restaurants. The discriminative features may be determined using any of the feature selection methods discussed in section 4.3 of Chapter 4. Each user is defined by a vector representation of discriminative words. An example of the possible user-word matrix for a restaurant recommender systems is shown below:

Word $\Rightarrow$ **User** $\Downarrow$	beef	roasted	lamb	fried	eggs
Sayani	0	3	0	2.5	1.7
John	2.3	1.3	0.2	1.4	2.1
Mary	0	2.8	0.9	1.1	2.6
Peter	2.4	1.7	0	3.5	1.9
Jack	1.6	2.2	3.1	1.0	0

The weights in the aforementioned table may be obtained using the descriptions of the items that the user has accessed. Note that the irrelevant words have already been removed because the content-based feature selection in the first phase creates a discriminative vector-space representation for each user. Furthermore, the representation is significantly denser than a typical ratings matrix. Therefore, one can robustly compute the similarities between users with this new representation. The main idea here is that the *content-based* peer group is used to determine the most similar users of the target user. Once the peer group has been determined, then the weighted average of the ratings of the peer group are used to determine the predicted ratings. Note that this approach does require a certain amount of change to the original collaborative recommender, at least in terms of how the similarity is computed. The peer group formation must use the user-word matrix (which was the model created in the first phase), whereas the final recommendation uses the ratings matrix. This is different from a collaborative system in which both stages use the same matrix. Furthermore, the first phase of the approach cannot use off-the-shelf content-based models in their entirety because it is mostly a feature selection (preprocessing) phase. Therefore, in many cases, these systems cannot be considered true ensembles, because they do not use existing methods as off-the-shelf recommenders.

Another example of a meta-level system was *LaboUr* [534] in which an instance-based model is to used to learn the content-based user's profile. The profiles are then compared using a collaborative approach. These models are compared across users to make predictions. Many of these methods fall within the category of "collaboration via content," though that is not the only way in which such hybrids can be constructed.

6.8 Feature Combination Hybrids

In feature combination hybrids, the idea is to combine the input data from various sources (e.g., content and collaborative) into a unified representation before applying a predictive algorithm. In most cases, this predictive algorithm is a content-based algorithm that uses collaborative information as additional features. An example of such an approach was presented in [69], where the *RIPPER* classifier was applied to the augmented data set. It was shown in [69] that the methodology achieved significant improvements over a purely collaborative approach. However, the content features need to be hand picked in order to achieve this result. Therefore, the approach can be sensitive to the choice of data set and feature representation. The approach reduces the sensitivity of the system to the number of users that have rated an item. This is, of course, the property of any content-based system, which is robust to the cold-start problem from the perspective of new items.

Note that it is possible for the combination to be performed in a variety of different ways with different types of background knowledge. For example, consider the case where each item is associated with a higher-level taxonomy representing the genres of the items. The representation profile of the user and items can be augmented in terms of the relevant genres in the hierarchy. The ratings matrix can then be constructed in terms of genres rather than items. In sparse matrices, such an approach can provide more effective results because it reduces the number of columns, and because most entries are likely to be populated in the compressed matrix.

Another approach is to augment a ratings matrix and add columns for keywords in addition to items. Therefore, the ratings matrix becomes an $m \times (n + d)$ matrix, where n is the number of items and d is the number of keywords. The weights of "keyword items" are based on the weighted aggregation of the descriptions of the items accessed, bought, or rated by the user. A traditional neighborhood or matrix factorization approach can be used with this augmented matrix. The relative weights of the two types of columns can be learned through cross-validation (see Chapter 7). This type of combination of two optimization models is common in hybrid settings, where the objective function is set up as follows in terms of a parameter vector $\overline{\theta}$:

$$J = \text{CollaborativeObjective}(\overline{\theta}) + \beta\, \text{ContentObjective}(\overline{\theta}) + \text{Regularization} \tag{6.9}$$

The objective function is then optimized over the parameter vector $\overline{\theta}$. A specific example, which is discussed below, is the generalization of sparse linear models (cf. section 2.6.5 of Chapter 2) with side information.

6.8.1 Regression and Matrix Factorization

Let R be an $m \times n$ implicit feedback ratings matrix, and C be a $d \times n$ content matrix, in which each item is described by non-negative frequencies of d words. Examples include descriptions of items or short reviews of items. Since R is an implicit feedback matrix, missing entries are assumed to be 0s. As in section 2.6.5, let W be an $n \times n$ item-item coefficient matrix in which the ratings are predicted as $\hat{R} = RW$. However, in this case we can also predict the ratings as $\hat{R} = CW$. Therefore, instead of optimizing only $||R - RW||^2$, we add an additional content-based term $||R - CW||^2$. Together with elastic-net regularization, and non-negativity/diagonal constraints, the enhanced optimization model is stated as follows [456]:

$$\begin{aligned} \text{Minimize } J = ||R - RW|||^2 + \beta \cdot ||R - CW||^2 + \lambda ||W||^2 + \lambda_1 \cdot ||W||_1 \\ \text{subject to:} \\ W \geq 0 \\ \text{Diagonal}(W) = 0 \end{aligned}$$

The weight parameter β can be determined in a tuning phase. Although the ratings can be predicted either as $\hat{R} = RW$ or as $\hat{R} = CW$, only the former prediction function is used. Therefore, the term $||R-CW||^2$ is used only to refine the objective function as an additional regularizer. In other words, the goal of the additional term is to improve the *generalization power* of the model for predicting future (and as yet unknown) actions of the user. Some variations of this basic objective function are discussed in [456].

This type of approach can be used for combining any other type of collaborative filtering (optimization) model with content-based methods. For example, in the case of matrix factorization, one can use an $m \times k$ user factor matrix U, an $n \times k$ *shared* item factor matrix V, and a $d \times k$ content factor matrix Z to set up the optimization model as follows [557]:

$$\text{Minimize } J = ||R - UV^T|||^2 + \beta \cdot ||C - ZV^T||^2 + \lambda(||U||^2 + ||V||^2 + ||Z||^2)$$

Note that the item factor matrix V is shared between the factorizations of the ratings matrix and content matrix. Such shared matrix factorization models are also used for incorporating other types of side information such as social trust data (cf. section 11.3.8 of Chapter 11). An overview of combining matrix factorization methods with arbitrary models is provided in section 3.7.7 of Chapter 3.

6.8.2 Meta-level Features

It is not necessary to use feature combination in the context of multiple types of recommenders (e.g., content and collaborative). New meta-features can be extracted from features of a particular type of recommender and then combined within the ensemble model. For example, one can extract *meta-level features* from a ratings matrix corresponding to the number of ratings given by various users and items. When a user rates many movies, or when a movie is rated by many users, it affects the recommendation accuracy of the various algorithms in different ways. Different recommender systems will be more or less sensitive to these characteristics, and will therefore do better or worse for various users and items. The basic idea of meta-level features is to account for these *entry-specific* differences in the model combination process with the use of meta-features. The resulting meta-features can be paired with other ensemble algorithms to create an ensemble design, which incorporates characteristics from various types of hybrids, but it does not neatly fall into any of Burke's seven original categories [117]. However, it is most closely related to feature combination hybrids in the sense that it combines meta-features with ratings.

Id.	Description
1	Constant value of 1 (using only this feature amounts to using the global linear regression model of section 6.3)
2	A binary variable indicating whether the user rated more than 3 movies on this particular date
3	The log of the number of times a movie has been rated
4	The log of the number of distinct dates on which a user has rated movies
5	A Bayesian estimate of the mean rating of the movie after having subtracted out the user's Bayesian estimated mean
6	The log of the number of user ratings
16	The standard deviation of the user ratings
17	The standard deviation of the movie ratings
18	The log of (Rating Date − First User Rating Date +1)
19	The log of the number of user ratings on the date +1

Table 6.1: A subset of the meta-features used in [554] for ensemble combination on the Netflix Prize data set

The meta-feature approach has proven to be a potentially powerful method for robust ensemble design. In fact, both winning entries in the Netflix Prize content, corresponding to *Bellkor's Pragmatic Chaos* [311] and *The Ensemble* [704], used such an approach. We will describe the use of such meta-level features in collaborative filtering algorithms. In particular, we will discuss the methodology of *feature weighted linear stacking* [554], which combines such meta-level features with the stacking methods discussed earlier in section 6.3. This approach is based on the blending technique used in *The Ensemble* [704]. A subset of the meta-features used in [554] for the stacking process on the Netflix Prize data set is provided in Table 6.1 for illustrative purposes. The identifier in the left column corresponds to the identifier used in the original paper [554]. These features are particularly instructive because one can usually extract analogous features for other ratings data sets. Note that each feature in Table 6.1 is specific to an entry in the ratings matrix.

Let us assume that a total of l (numeric) meta-features have been extracted, and their values are $z_1^{ut} \dots z_l^{ut}$ for user-item pair (u, t). Therefore, the meta-features are specific to each entry (u, t) in the ratings matrix, although some features may take on the same values for varying values of u or varying values of t. For example, feature 3 in Table 6.1 will not vary with the user u, but it will vary with the item t.

Let us assume that there are a total of q base recommendation methods, and the weights associated with the q recommendation methods are denoted by $w_1 \dots w_q$. Then, for a given entry (u, t) in the ratings matrix, if the predictions of the q components are $\hat{r}_{ut}^1 \dots \hat{r}_{ut}^q$, then the prediction $\hat{r}_{ut}$ of the overall ensemble is given by the following:

$$\hat{r}_{ut} = \sum_{i=1}^{q} w_i \hat{r}_{ut}^i \tag{6.10}$$

We would like the estimated prediction $\hat{r}_{ut}$ of the ensemble to match the observed ratings r_{ut} as closely as possible. Note that the approach in section 6.3 uses a linear regression model to learn the weights $w_1 \dots w_q$ by holding out a pre-defined fraction of the entries during the process of training the q models, and then using the held-out entries as the observed values in the linear regression model. Such an approach is pure stacking, and it can be considered a weighted hybrid. However, it can be enhanced further using meta-features. The main idea is that *the linear regression weights $w_1 \ldots w_q$ are specific to each entry in the ratings matrix and they are themselves linear functions of the meta-features.* In other words, the weights now need to be super-scripted with (u, t) to account for the fact that they are specific to each entry (u, t) in the ratings matrix:

$$\hat{r}_{ut} = \sum_{i=1}^{q} w_i^{ut} \hat{r}_{ut}^i \tag{6.11}$$

This is a more refined model because the nature of the combination is local to each entry in the ratings matrix, and it is not blindly global to the entire matrix. The problem is that the number, $m \times n \times q$, of different parameters w_i^{ut} becomes too large to be learned in a robust way. In fact, the number of parameters (weights) is larger than the number of observed ratings, as a result of which overfitting will occur. Therefore, the weights are assumed to be linear combinations of the meta-features under the assumption that these meta-features regulate the relative importance of the various models for the individual user-item combinations. Therefore, we introduce the parameters v_{ij} that regulate the importance of the jth meta-feature to the ith model. The weights for entry (u, t) can now be expressed as linear combination of the meta-feature values of entry (u, t) as follows:

$$w_i^{ut} = \sum_{j=1}^{l} v_{ij} z_j^{ut} \tag{6.12}$$

We can now express the regression modeling problem in terms of a fewer number, $q \times l$, of parameters v_{ij}, where v_{ij} regulates the impact of the jth meta-feature on the relative importance of the ith ensemble model. Substituting the value of w_i^{ut} from Equation 6.12 in Equation 6.11, we obtain the relationship between the ensemble rating and the component ratings as follows:

$$\hat{r}_{ut} = \sum_{i=1}^{q} \sum_{j=1}^{l} v_{ij} z_j^{ut} \hat{r}_{ut}^i \tag{6.13}$$

Note that this is still a linear regression problem in $q \times l$ coefficients corresponding to the variables v_{ij}. A standard least-squares regression model can be used to learn the values of v_{ij} on the held-out[5] ratings. The independent variables of this regression are given by the quantities $z_j^{ut} \hat{r}_{ut}^i$. Regularization can be used to reduce overfitting. After the weights have been learned using linear regression, the individual component models are retrained on the entire training set without any held-out entries. The weights, which were learned using the held-out entries, are used in conjunction with these q models.

6.9 Mixed Hybrids

The main characteristic of mixed recommender systems is that they combine the scores from different components in terms of *presentation*, rather than in terms of combining the predicted scores. In many cases, the recommended items are presented next to one another [121, 623]. Therefore, the main distinguishing characteristic of such systems is the combination of presentation rather than the combination of predicted scores.

[5]In the context of the Netflix Prize contest, this was achieved on a special part of the data set, referred to as the *probe set*. The probe set was not used for building the component ensemble models.

Most of the other hybrid systems focus on creating a unified rating, which is extracted from the various systems. A classical example is illustrated in [559], in which a personalized television listing is created using a mixed system. Typically, a *composite* program is presented to the user. This composite program is created by combining items recommended by different systems. Such composite programs are typical in the use of mixed systems, although the applicability of mixed systems goes beyond such scenarios. In many of these cases, the basic idea is that the recommendation is designed for a relatively complex item containing many components, and it is not meaningful to recommend the individual items. The new item startup problem is often alleviated with a mixed recommender system. Because a television program has many slots, either the content-based or collaborative recommender might be successful in filling the different slots. In some cases, a sufficient number of recommendations for the slots may be achieved only with multiple recommenders of different types, especially at the very beginning when there is a paucity in the available data. However, conflict resolution may be required in some cases where more choices are available than the available slots.

Another example of a mixed hybrid has been proposed in the tourism domain [660, 661]. In this case, bundles of recommendations are created, where each bundle contains multiple categories of items. For example, in a tourism recommender system, the different categories may correspond to accommodations, leisure activities, air-tickets, and so on. Tourists will typically buy bundles of these items from various categories in order to create their trips. For each category, a different recommender system is employed. The basic idea here is that the recommender system that is most appropriate for obtaining the best accommodations, may not be the one that is most appropriate for recommending tourism activities. Therefore, each of these different aspects is treated as a different category for which a different recommender system is employed. Furthermore, it is important to recommend bundles in which the items from multiple categories are not mutually inconsistent. For example, if a tourist is recommended a leisure activity that is very far away from her place of accommodation, then the overall recommendation bundle will not be very convenient for the tourist. Therefore, a knowledge base containing a set of domain constraints is incorporated for the bundling process. The constraints are deigned to resolve inconsistencies in the product domain. A constraint satisfaction problem is employed to determine a mutually consistent bundle. More details of the approach are discussed in [660, 661].

It is noteworthy that many of the mixed hybrids are often used in conjunction with knowledge-based recommender systems as one of the components [121, 660]. This is not a coincidence. Mixed hybrids are generally designed for complex product domains with multiple components like knowledge-based recommender systems.

6.10 Summary

Hybrid recommender systems are used either to leverage the power of multiple data sources or to improve the performance of existing recommender systems within a particular data modality. An important motivation for the construction of hybrid recommender systems is that different types of recommender systems, such as collaborative, content-based, and knowledge-based methods, have different strengths and weaknesses. Some recommender systems work more effectively at cold start, whereas other work more effectively when sufficient data are available. Hybrid recommender systems attempt to leverage the complementary strengths of these systems to create a system with greater overall robustness.

Ensemble methods are also used to improve the accuracy of collaborative filtering methods in which the different components use the same ratings matrix. In these cases, the individual models use the same base data rather than different sources of data. Such methods are much closer to the existing ideas on ensemble analysis in the classification domain. The basic idea is to use the various models to incorporate diversity and reduce model bias. Many of the existing theoretical results on the bias-variance trade-off in classification are also applicable to collaborative filtering applications. Therefore, many techniques, such as bagging and boosting, can be adapted with relatively minor modifications.

Hybrid systems are designed as monolithic systems, ensemble systems, or mixed systems. Ensemble systems are typically designed by using either sequential or parallel arrangement of recommenders. In monolithic design, either existing recommenders are modified, or entirely new recommenders are created by combining the features from multiple data modalities. In mixed systems, recommendations from multiple engines are presented simultaneously. In many cases, meta-features can also be extracted from a particular data modality in order to combine the predictions of various recommenders in an entry-specific way. The great strength of hybrid and ensemble systems arises from their ability to leverage complementary strengths in various systems. The top entries in the Netflix Prize contest were all ensemble systems.

6.11 Bibliographic Notes

Although hybrid systems have a long and rich history in the development of recommender systems, a formal categorization of these methods was not performed until the survey by Burke [117]. A discussion of hybrid recommender systems in the specific context of the Web is provided in [118]. Burke originally categorized recommender systems into seven different categories. Subsequently, Jannach *et al.* [275] created a higher-level categorization of these lower-level categories into pipelined and parallel systems. The hierarchical taxonomy in this book roughly follows the work of [275] and [117], although it makes a number of modifications to include several important methods, such as boosting, into one of these categories. It is important to note that this taxonomy is not exhaustive because many ensemble systems, such as those winning the Netflix Prize, use ideas from many types of hybrids. Nevertheless, Burke's original categorization is very instructive, because it covers most of the important building blocks. Recently, ensemble methods have received a lot of attention, especially after the winning entries in the Netflix Prize contest were both ensemble systems [311, 704].

Ensemble methods have been used extensively in the classification literature. A detailed discussion of the bias-variance trade-off in the context of the classification problem is provided in [22]. Bagging and subsampling methods for classification are discussed in [111–113]. A recent work [67] shows how one might leverage ensemble methods from the classification literature to recommender systems by adapting methods such as bagging and *AdaBoost.RT*. While some ensemble systems are developed with this motivation, other systems combine the power of different data types. Weighted models are among the most popular classes of models. Some of the models combine models built on homogeneous data types. Methods for constructing homogeneous weighted ensembles are discussed in [67, 266]. The winners [311, 704] of the Netflix Prize contest also used a weighted ensemble system, although the combination uses additional meta-features, which imbues it with some of the properties of a feature combination approach. The work in [180] uses ensembles of maximum margin matrix factorization methods with different parameter settings. User-based and item-based neighborhood algorithms are combined in [338]. Other recent work on weighted models shows how to combine systems built on top of different data types. The work in [659] combines a collaborative and knowledge-based recommender, whereas the work in [162] combines a content-based and collaborative recommender.

A performance-based switching hybrid is discussed in [601]. An interesting machine-learning approach to switching mechanisms is discussed in [610]. Other switching mechanisms for handling cold-start issues are discussed in [85]. Another combination of a knowledge-based and collaborative system to create a switching hybrid is discussed in [659].

Cascade systems use sequential processing of the ratings to make recommendations. Such systems can either use refinements or they can use boosting methods. The *EntreeC* recommender [117] is the most well-known example of a cascade system that uses refinements. A cascade system that uses boosting is discussed in [67]. The latter methods uses a weighted version of the *AdaBoost.RT* algorithm in order to create the hybrid recommender.

Feature augmentation hybrids use the recommenders of one type to augment the features of another. The *Libra* system [448] combines Amazon.com's recommender system with its own Bayes classifier. The output of the Amazon system is used to create a content-based recommender. The method in [431] uses a content-based system to estimate the missing entries of the ratings matrix and uses the estimated values in the context of a collaborative system. In the *GroupLens* system [526], a knowledge-based system was used to create a database of artificial ratings. These ratings were used in the context of a collaborative system to make recommendations. The work in [600] shows how to use a feature augmentation hybrid to recommend research papers.

Many techniques have been used recently to create fused feature spaces or unified representations from ratings matrices and content matrices. This unified representation or feature space forms the basis on which machine learning tools can be applied. One of the earliest works along this line constructs joint feature maps [68] from rating and content information and then uses machine learning models in order to perform the prediction. A tensor-based approach is used to achieve this goal. An analogous approach is also used in [557], which jointly factorizes the user-item purchase profile matrix and the item-feature content matrix into a common latent space. This latent representation is then used for learning. The work in [411] uses a latent factor model in which the review text is combined with ratings. A regression-based latent factor model is proposed in [14] for rating prediction, which uses content features for factor estimation. The user and item latent factors are estimated through independent regression on user and item features. Then, a multiplicative function is used on the user and item factors for prediction. Sparse regression models have also been used for fused prediction in [456]. Finally, graph-based models have been used to create unified representations. The work in [238] leans the interaction weights between user actions and various features such as user-item profile information and side information. Unified Boltzmann machines are used to perform the prediction. A unified graph-based representation has been proposed in [129]. A Bayesian network is created containing item nodes, user nodes, and item feature nodes. This Bayesian network is used to perform combined content-based and collaborative recommendations.

In a meta-level hybrid, the *model* learned by one recommender is used as input to the next level. In the early work by Pazzani [475], a content-based model [363] is constructed that describes the discriminative features predicting restaurants. Each user is defined by a vector representation of discriminative words. The content-based model is used to determine the peer group, which is then used for the purpose of recommendation. Meta-level combinations of content-based and collaborative systems are discussed in [475, 534]. A two-stage Bayesian meta-level hybrid is discussed in [166]. A different type of hierarchical Bayes model that combines collaborative and content-based systems is presented in [652]. Methods for stacking recommender systems with meta-features are discussed in [65, 66, 311, 554]. The *STREAM* system [65, 66] was one of the earliest systems to leverage meta-level features.

A number of mixed recommender systems have been proposed in [121, 559, 623, 660, 661]. A mixed recommender system for creating television programs is discussed in [559], whereas a system for providing tourism bundles is discussed in [660]. It is noteworthy that many mixed recommender systems are used in complex product domains like knowledge-based recommender systems [121, 660].

6.12 Exercises

1. How does the rank of the latent factor model affect the bias-variance trade-off in a recommender system? If you had to use a latent factor model as the base model for a bagging ensemble, would you choose a model with high rank or low rank?

2. Does your answer to Exercise 1 change if you had to use boosting in conjunction with a latent factor model?

3. Implement an entry-wise bagging model by using a weighted latent factor model as the base model.

4. Suppose that you created a collaborative system in which the user-item matrix contained word frequencies as additional rows of the matrix. Each additional row is a word, and the value of the word-item combination is a frequency. An item-based neighborhood model is used with this augmented representation. What kind of hybrid would this be considered? Discuss the possible impact of using such a model on the accuracy and diversity of the recommender system.

5. Discuss how you would control the relative strength of collaborative and content-based portions in Exercise 4 with a single weight parameter. How would you determine the optimal value of the weight parameter in a data-driven way?