Chapter 13

Advanced Topics in Recommender Systems

"*In the last fifty years, science has advanced more than in the two thousand previous years and given mankind greater powers over the forces of nature than the ancients ascribed to their gods.*"– John Boyd Orr

13.1 Introduction

Recommender systems are often used in a number of specialized settings that are not covered in previous chapters of this book. In many cases, the recommendations are performed in settings where there might be multiple users or multiple evaluation criteria. For example, consider a scenario where a group of tourists wish to take a vacation together. Therefore, they may want to obtain recommendations that match the overall interests of the various members in the group. In other scenarios, users may use multiple criteria to provide ratings to items. These variations in the problem formulation can sometimes make the prediction problem more challenging. In particular, we will study the following advanced variations of recommender systems in this chapter:

1. *Learning to rank:* Most of the models discussed in the previous chapters treat the recommendation problem as a rating prediction problem in which the squared error of prediction is minimized. However, in real-life settings, users are presented only with the top-k recommendations, and the other predictions are ignored. Therefore, it makes sense to explore whether one can directly optimize ranking-based evaluation criteria, such as the mean reciprocal rank or the area under curve of the receiver operating characteristic curve.

C.C. Aggarwal, *Recommender Systems: The Textbook*,
DOI 10.1007/978-3-319-29659-3_13

2. *Online learning with multi-armed bandits:* In many recommendation domains, such as that of recommending news articles, the cold-start problem is pervasive. New articles and stories appear all the time, and the effectiveness of various algorithms may also vary with time. In such cases, it is crucial for the approach to continuously explore the space of various choices as new data are received. At the same time, the learned data are exploited in order to optimize the payoff in terms of the conversion rate. This type of trade-off between exploration and exploitation is managed with the help of multi-armed bandit algorithms.

3. *Group recommender systems:* In many settings, the recommendations may not be made to individuals, but to groups of users. Such recommendations are typically associated with activities undertaken by groups of users. Examples include a visit to the movies by a group, travel services bought by a group, the choice of music or television programs played or watched by a group, and so forth. In these cases, users may have varying tastes and interests that are reflected in their different choices. Group recommender systems are designed to work with these various trade-offs in order to make meaningful recommendations.

4. *Multi-criteria recommender systems:* In multi-criteria systems, ratings might be specified on the basis of different criteria by a single user. For example, a user might rate movies based on the plot, music, special effects, and so on. Such techniques often provide recommendations by modeling the user's utility for an item as a vector of ratings corresponding to various criteria. In fact, it has been shown [271, 410] that some of the methods for group recommender systems can also be adapted to multi-criteria recommender systems. However, the two topics are generally considered different because they emphasize different aspects of the recommendation process.

5. *Active learning in recommender systems:* Active learning is a well-known technique that is used in classification to acquire labels of training examples so as to maximize classification accuracy. The acquisition of labels is generally expensive; therefore, one must choose the training examples judiciously in order to maximize the accuracy of the classifier for a given cost budget. As the problem of recommendation can be viewed as a generalization of classification, the available methods for active learning can also be generalized to recommendations. Active learning provides methods of acquiring ratings within a given budget in order to maximize prediction accuracy.

6. *Privacy preservation in recommender systems:* Recommender systems are deeply dependent on the ability of users to voluntarily provide information about their interests. Such information is quite sensitive because it may reveal information about political opinions, sexual orientation, and so on. Therefore, it is crucial to develop privacy-preserving methods for the recommendation process. When there is a risk of public disclosure, owners of ratings data become less likely to release it. An example is the case of the Netflix Prize, in which a sequel to the contest was not pursued because of privacy concerns [714].

In addition to the aforementioned topics, this chapter will also study the application of recommender systems technology to a variety of application domains, such as news recommendations, computational advertising, and reciprocal recommender systems. The idea of studying these topics is to provide an understanding of how the methods discussed in various chapters are applied to these different domains. In some cases, the techniques discussed in these chapters cannot be applied directly, and therefore new techniques must be developed. Therefore, one of our goals is to provide an understanding of the limitations of the various methods used in current settings.

This chapter is organized as follows. The next section will introduce the problem of learning to rank. Multi-armed bandit algorithms are introduced in section 13.3. Various techniques for designing group recommender systems will be discussed in section 13.4. Multi-criteria recommender systems are discussed in section 13.5. Active learning methods are introduced in section 13.6. Methods for privacy in collaborative filtering are discussed in section 13.7. A number of interesting application domains are discussed in section 13.8. A summary is given in section 13.9.

13.2 Learning to Rank

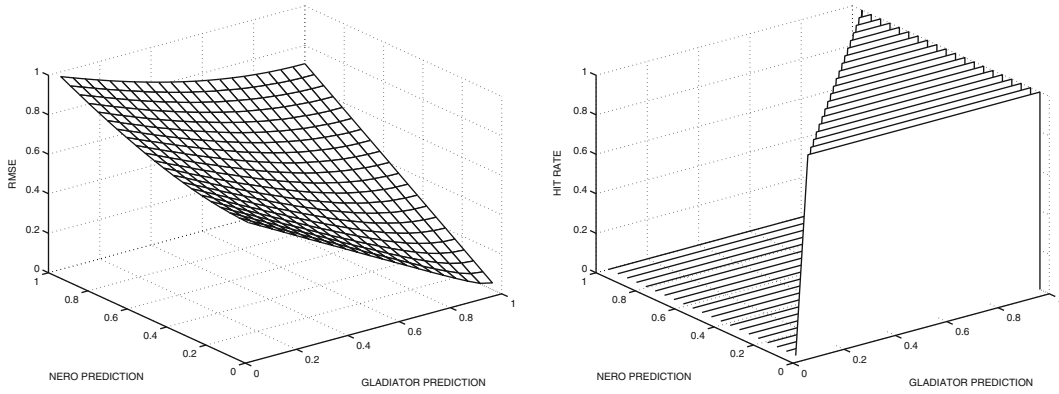
Most of the models discussed in previous chapters treat the recommendation problem as a prediction problem in which the squared error of rating prediction is optimized. However, in practice, recommender systems rarely present all the ratings to the user. In practice, only the top-k items are presented to the user as a ranked list. Furthermore, the user is more likely to pay attention to the items at the top of the list than the lower-ranked items. The predicted values for the items not included in the list are irrelevant from the user perspective. In many cases, optimizing predicted values of the ratings may not provide the best recommendation lists to the end user. For example, if all the low-ranked ratings are predicted very accurately, but significant errors are made on the higher-ranked ratings, the resulting solution will not provide a high-quality recommendation list to the end-user. On the other hand, a prediction-based objective function may report it as a high-quality solution because the lower-ranked items are given equal importance. This problem arises because the objective functions of prediction-based methods are not fully aligned with the end-user experience.

The classical objective function used in optimization models for recommender systems (such as matrix factorization) is the aggregate squared error. This type of objective function is optimized to the RMSE measure used for evaluating recommender systems. It is also particularly easy to optimize from an algorithmic standpoint. This is one of the reasons that such prediction-based objective functions dominate the recommendation modeling landscape. However, as discussed in Chapter 7 on evaluating recommender systems, there are many rank-centric measures used for evaluating recommender systems. Such rank-centric measures can also be directly optimized in the context of collaborative filtering (or even content-based) models. As discussed in the chapter on evaluating recommender systems (cf. Chapter 7), there are two primary types of ranking measures:

1. *Global ranking measures:* Such measures evaluate the entire ranked lists of all the items. Examples, include the Kendall coefficient, Spearman coefficient, and the area under curve (AUC) of the receiver operating characteristic (ROC).

2. *Top-heavy ranking measures:* These are typically utility-based measures in which the top-ranked items are given much more importance. Examples of such measures include the normalized cumulative discounted gain (NDCG) and mean reciprocal rank (MRR). Such measures are often the most realistic from the perspective of the end-user because they ignore the lower-ranked items. Such items are not visible to the end-user in the recommended list.

Many of the ranking-based measures are used for evaluating implicit data settings. Correspondingly, many of the ranking-based learning methods are also designed in the context of implicit data settings.



(a) Smooth RMSE objective (b) Non-smooth hit rate

Figure 13.1: Ranking objective is not a smooth function of predicted ratings (and underlying model parameters)

For example, consider the problem of factorizing the ratings matrix R into user- and item-factors, U and V, respectively. One would like to determine U and V such that a specific ranking objective is optimized. Then, one might pose the optimization problem as follows:

Optimize J = [Objective function quantifying *ranking* evaluation between R and UV^T]
subject to:
Constraints on U and V

As in traditional matrix factorization, it is possible to add a regularizer to improve the generalization power of the objective function. The constraints on U and V might depend on the specific application setting. For example, in an implicit feedback setting, one might impose nonnegativity constraints on U and V. The optimization objective function might be derived from ranking-based measures such as the NDCG, MRR, AUC, and so on. A specific example of such a matrix factorization method, which optimizes the AUC, is discussed in [432]. In this work, the link recommendation problem is solved with the use of the AUC-based objective.

The main challenge in ranking-based methods is that the underlying objective functions are often *non-smooth* [490], which can be hard to optimize with off-the-shelf gradient-descent techniques. Tiny changes in the predicted ratings can change the item rankings and the corresponding objective functions suddenly. For example, consider a setting where there are two movies *Nero* and *Gladiator*, with true ratings 0 and 1, respectively. The predicted ratings can be converted to ranks, and the top-1 recommended movie can be reported. The (smooth) RMSE for various combinations of predicted ratings is shown in Figure 13.1(a), whereas the (non-smooth) hit-rate of the top-1 predicted rating is shown in Figure 13.1(b). Note the sudden jump in objective function in the case of Figure 13.1(b) at particular values of the predicted ratings. In the case of ranking-based objective functions, such non-smooth jumps or drops can occur with small changes in not just the predicted values but also the underlying model parameters. For example, in matrix factorization methods, tiny changes in the parameters of the user and item factors can cause sudden jumps or drops in ranking-based objectives. Such non-smooth changes are not observed with conventional

measures such as the squared error, which are much easier to optimize. For example, a gradient-descent method would have difficulty in determining the correct descent direction with a non-smooth objective function because important changes in the objective function might occur at non-differentiable points in the parameter space. To get around this problem, smooth approximations of the underlying objective functions are often used. For each individual ranking-based objective, a specific lower-bound or approximation is used to design a smooth variation of the underlying objective function. Since these smooth variations are only approximations, the quality of the algorithm will often depend on that of the underlying approximation. In the following, we provide a brief discussion of some of the common ranking-based methods.

The traditional approach to ranking is to first predict the ratings with a loss function and then rank the items using the predicted ratings. One can view this approach as a *pointwise* methodology. Many of these methods are not specifically optimized to ranking because they focus on predicting the values of the ratings. A particularly notable work in this category is *OrdRec* [314], which treats ratings as ordinal values rather than as numerical values. There are two other primary types of methods that are specifically optimized to rank-centric learning, and they are referred to as *pairwise* or *listwise* rank-learning methods [128]. In the following, we will discuss these different types of rank learning methods.

13.2.1 Pairwise Rank Learning

In pairwise rank learning, pairs of items for which the users have provided preferences are used as the training data. Each pair contains only information about whether the first item of the pair is preferred to the second one or not, with a $+1$ or a -1, respectively. For example, consider a scenario where John has provided ratings for *Terminator*, *Alien*, and *Gladiator*, as 4, 3, and 5, respectively. Then, one can create the following pairs of training points:

John, *Terminator*, *Alien*, $+1$
John, *Terminator*, *Gladiator*, -1
John, *Alien*, *Gladiator*, -1

One can generate similar pairs for Peter, Bob, Alice, and so on, to create the training data across all users. For implicit feedback data sets, one can treat unobserved values as 0s. With this training data, one can now try to learn the relative item preferences such as the following:

Alice, *Terminator*, *Gladiator*, ?
Bob, *Terminator*, *Gladiator*, ?
John, *Nero*, *Cleopatra*, ?

Note that this transformation essentially creates a binary classification problem, and the learning method implicitly tries the minimize the number of pairwise inversions in the training data. This objective is intimately related to the Kendall rank correlation coefficient. It is also possible to optimize other measures such as the AUC in this setting. One can use any off-the-shelf ranking classifier (such as ranking SVMs) to learn an appropriate ranking objective. The main challenge in doing so is that the underlying representation is very sparse, since each training example contains only three nonzero elements of the form $\langle User, Item1, Item2 \rangle$. Note that the base dimensionality might contain hundreds of

thousands of users and items. Such a setting is particularly well-suited to factorization machines (cf. section 8.5.2.1 of Chapter 8). With m users and n items, one can create a $p = (m+2 \cdot n)$-dimensional binary representation $x_1 \ldots x_p$, such that exactly three elements of the representation are set to 1. The remaining elements are set to 0. The m elements in the representation correspond to users and the $2 \cdot n$ elements correspond to pairs of items. The predicted value $y(\overline{x})$ is either $+1$ or -1 depending on whether or not the rating of the first item is larger than the second one. Then, the prediction function of Equation 8.9 is modified to a form used in logistic regression:

$$P(y(\overline{x}) = 1) = \frac{1}{1 + \exp(-[g + \sum_{i=1}^{p} b_i x_i + \sum_{i=1}^{p} \sum_{j=i+1}^{p} (\overline{v_i} \cdot \overline{v_j}) x_i x_j])} \tag{13.1}$$

The model parameters g, b_i, and $\overline{v_i}$ are defined in the same way as in section 8.5.2.1 of Chapter 8. A log-likelihood criterion can be optimized to learn the underlying model parameters with a gradient-descent approach. Factorization machines also provide the flexibility to do the feature engineering in other ways [493]. For example, one can use the $(m + n)$-dimensional binary representation $x_1 \ldots x_{m+n}$, in which two entries are nonzero (corresponding to user-item combination) and assume that the prediction $y(\overline{x})$ is equal to the value of the rating. Then, one can directly optimize a ranking objective function over pairs of predictions $(y(\overline{x_i}), y(\overline{x_j}))$, depending on which one is larger in the observed data. The main difference between this approach and the previous one is that the current approach optimizes pairwise ranking over all pairs $(y(\overline{x_i}), y(\overline{x_j}))$ (irrespective of whether $\overline{x_i}$ and $\overline{x_j}$ correspond to the same user), whereas the previous one does not allow the ratings of a particular pair to belong to different users.

Other well-known models used to learn these predictions include the Bayesian personalized ranking model (BPR) [499], EigenRank model [367], pLPA [368], and CR [59]. Many of these methods use ranking-based measures in the underlying objective function for learning.

13.2.2 Listwise Rank Learning

In listwise rank learning, the quality of the entire list is evaluated by using a ranking-based objective function. Examples of such objective functions include the normalized cumulative discounted gain (NDCG), mean reciprocal rank (MRR), and so on. One can view an ordered list as a permutation of items with a specific objective function value, depending on the ranking measure. Therefore, the key is to devise an optimization model that can determine this permutation directly. These methods generally tend to be more focussed on implicit feedback matrices because of the natural importance of ranking-based methods in these methods. Some examples of listwise methods are as follows:

1. *CoFiRank:* This approach [624, 625] is optimized for maximizing the NDCG measure with the use of *structured estimation* methods. A structured estimation method is designed to work for complex output domains such as sequences. One can view the output of a listwise method to belong to a structured output domain because a list is also an ordered sequence. The idea is to define a structured loss function that works on lists rather than individual points, and whose optimization results in the best possible ranking. The basic idea is that the dot product of a permutation of the predicted ratings of all items with the sorted vector $\overline{c} = (\frac{1}{\sqrt[4]{2}}, \frac{1}{\sqrt[4]{3}}, \ldots \frac{1}{\sqrt[4]{n+1}})$ is maximized when the predicted ratings are in decreasing order (based on the Polya-Littlewood-Hardy inequality). In other words, the dot product $\overline{c} \cdot \overline{r_u}^{\pi}$ of $\overline{c}$ with the corresponding permutation $\overline{r_u}^{\pi}$ of the estimated ratings $\overline{r_u}$ is maximized when the ratings in $\overline{r_u}^{\pi}$

are in decreasing order. The overall loss function is defined by maximizing a sum of $1 - NDCG(\pi)$ and $\overline{c} \cdot (\overline{r_u}^{\pi} - \overline{r_u})$ over all possible values of π. An upper bound on the loss function can be shown because of the Poly-Littlewood-Hardy inequality. The loss function is summed over all users, and a maximum margin optimization problem is defined in order to determine the optimal value of the predicted ratings.

2. *CLiMF:* This approach [545, 546] optimizes the mean-reciprocal rank (MRR), which has the tendency to obtain at least a few interesting items at the top of the list. The basic idea is to determine a smoothed version of the MRR for optimization and determine a lower bound on this smoothed version. Note that this approach is designed for implicit feedback data sets because of its use of the MRR. A related method, referred to as xCLiMF, is designed for explicit ratings.

Numerous other methods have been proposed for incorporating context into such methods [549].

One can further improve the quality of ranking methods with ensemble learning. Multiple techniques are used to learn the rankings, and the different sets of ranks are aggregated into a single ranked list. This problem is that of *rank aggregation* [190]. For example, one might use the average or median rank across different ensemble learners to re-rank the items. However, other sophisticated methods are possible, such as the use of the best rank across different learners or combining the two methods in some way. The median rank is known to have several desirable theoretical properties in terms of the quality of the aggregation. This area remains relatively unexplored, and is a good candidate for future research.

13.2.3 Comparison with Rank-Learning Methods in Other Domains

An excellent tutorial on ranking methods for recommendations may be found in [323]. It is noteworthy that the dichotomy between prediction-based and ranking-based models also exists in classification and regression modeling. For example, ranking support vector machines were introduced in [284] in the context of an internet search engine. Gradient-descent methods for ranking were discussed in [115] with a neural network model. Neural networks have the advantage that they are universal function approximators, and therefore multi-layer neural networks can often be quite effective with ranking-based cost functions. An elaborate tutorial on the ranking problem in the context of machine learning may be found in [15]. The typical application discussed in this class of works is that of internet search, which can also be viewed as a kind of recommendation. Since the recommendation problem can be viewed as a generalization of classification and regression modeling, it is natural to also design ranking variations of recommendation algorithms. In fact, ranking variations are much more important in the context of recommender design because most users are presented only with restricted sets of ranked lists rather than predicted values. Such methods have also been explored extensively in the context of ranking methods in the information retrieval domain. A tutorial on such methods may be found in [370], and the methods strongly overlap with those used in the machine learning literature for internet search [15, 115, 284]. The methods from information retrieval can be used to directly improve the effectiveness of content-based methods in the recommendation domain.

13.3 Multi-Armed Bandit Algorithms

An important challenge in many recommendation settings is that new users and items constantly appear in the system and it is important for the recommender system to constantly adapt to the changing patterns in the data. Therefore, unlike offline recommendation algorithms, the approach needs to simultaneously *explore* and *exploit* the search space of recommendations. Each time an opportunity arises to show a recommendation to a user, the recommender system has to choose between a number of strategies, objects, or algorithms that decide what is shown to the user. These choices may be different, depending on the application domain at hand. Some examples are as follows:

1. The system might use a number of different recommendation algorithms, which might be more or less effective with different users. For example, a user who prefers a high level of customization might be better served with a knowledge-based recommender system, whereas a "lazy" user might be better served with a collaborative recommender that does most of the work for her. Therefore, the approach may constantly need to learn the best choice of strategy for each user.

2. A special (and important) case of the aforementioned setting is one where each strategy corresponds to recommending a specific item. For example, a news portal might show articles from various topics to a particular user over a period of time, and then bias the article presentation depending on the previous history of interest (i.e., clicks) on the various articles. In the context-free case, the recommendation is independent of the user. However, in practice, a feature vector is associated with each user, which characterizes the interest of the user in a specific topic. This provides the means to incorporate personalization in multi-armed bandit algorithms. If a user is more interested in sports and entertainment, then the recommender system needs to learn this fact during the operation of the system, and frequently show recommendations belonging to these topics to that individual.

The main challenge in these systems is that new users and new articles constantly enter the system; therefore, one must *simultaneously* learn the user interests and exploit these interests *during the operation of the system*. This is different from the offline setting discussed in this book. This problem is related to that of *reinforcement learning*, in which exploration and exploitation of the search space are performed simultaneously. One such important class of reinforcement learning algorithms is that of the multi-armed bandit algorithms.

This class of algorithms derives its name from the fact that one can view the recommender system in a manner similar to a gambler in a casino, who is faced with a choice of a number of slot machines (recommendation algorithms or strategies). This scenario is illustrated in Figure 13.2. By pulling the arms of each of these machines, the gambler will receive a payoff with a specific probability distribution. The gambler suspects that one of these slot machines might have higher (expected) payoff than the others, although it is impossible for the gambler to identify this machine without playing all the machines. Playing these machines for learning purposes can be viewed as an *exploration* of the search space of strategies. Of course, this learning phase is likely to waste trials because it is not optimized to the best paying machine. However, once the gambler learns that one of these machines has a better payoff, he or she can play that machine to achieve a better payoff. Like all reinforcement learning algorithms, multi-armed bandit algorithms are faced with a natural trade-off between exploration and exploitation of the search space.

Let us explain this scenario in the context of Web page recommender systems. Whenever a recommender system has to decide on the recommendation of a Web page to a user, it is faced with a number of different choices of strategies. For example, the recommender system

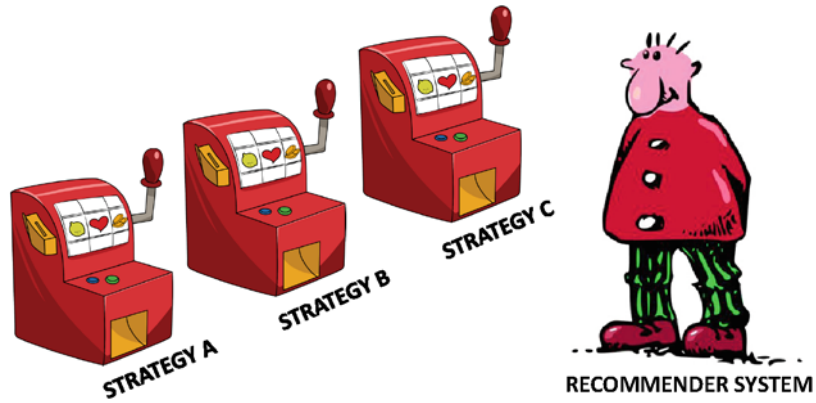


Figure 13.2: The multi-armed bandit analogy

may have to decide on the choice of Web pages to recommend. These choices correspond to the arms of various slot machines. When a user clicks on the link of a recommended page, the recommender system receives a payoff in terms of the success of the recommendation. In the simplest case, the click-through problem is modeled with binary payoffs, where a click amounts to a payoff of 1 unit. This payoff can be viewed in an analogous way to that received by a gambler from the slot machine. In most practical settings, additional contextual information may be available to the recommender system about the user or the context of the recommendation. Some examples of such contextual information are as follows:

1. A set of features describing the profile of the user or the item-context may be available. Examples of item context might include the content of the Web page on which a recommendation is displayed. For example, a recommendation on a Web page describing the movie *Terminator* might be very different from that on a page describing the movie *Nero*. This type of context is particularly common in settings such as computational advertising.

2. The users may be clustered into groups, and the cluster identifier of the group may be used as semantic knowledge about the user. This is because similar users might have similar payoffs, and therefore the analysis can be segmented in group-wise fashion.

In cases where contextual information is available about the users, it is often assumed that user identification mechanisms are available. In order to explain the use of multi-armed bandit algorithms, we will first discuss the traditional setting where no contextual information is available. We will then provide a basic understanding of how contextual information may be incorporated within multi-armed bandit algorithms.

There are a number of strategies that the gambler can use to regulate the trade-off between exploration and exploitation of the search space. In the following, we will briefly describe some of the common strategies used in multi-armed bandit systems.

13.3.1 Naive Algorithm

In this approach, the gambler plays each machine for a fixed number of trials in the exploration phase. Subsequently, the machine with the highest payoff is used forever in the

exploitation phase. This strategy shares a number of similarities with A/B-testing used for online evaluation of recommender systems. The difference is that A/B-testing uses only the exploration phase for evaluation purposes, whereas the bandit algorithm has an additional exploitation phase.

Although this approach might seem reasonable at first sight, it has a number of drawbacks. The first problem is that it is hard to determine the number of trials at which one can confidently predict whether a particular machine is better than the other. The process of estimation of payoffs might take a long time, especially in cases where the payoff events and non-payoff events are very unevenly distributed. For example, in a Web recommendation algorithm, the percentage of click-throughs might be low, as a result of which many trials will be required before one can confidently state whether one recommendation algorithm is better than the other. Using many exploratory trials will waste a significant amount of effort on suboptimal strategies. Furthermore, if the wrong strategy is selected in the end, the gambler will use the wrong slot machine forever. In practice, the payoffs of various machines (recommendation algorithms) might evolve over time. This is particularly true of the type of dynamic recommendation settings addressed by multi-armed bandit methods. Therefore, the approach of fixing a particular strategy forever is unrealistic in real-world problems.

13.3.2 ϵ-Greedy Algorithm

The ϵ-greedy algorithm is designed to use the best strategy as soon as possible, without wasting a significant number of trials. The basic idea is to choose a random slot machine for a fraction ϵ of the trials. These exploratory trials are also chosen at random (with probability ϵ) from all trials, and are therefore fully interleaved with the exploitation trials. In the remaining $(1-\epsilon)$ fraction of the trials, the slot machine with the best average payoff so far is used. An important advantage of this approach is that one is guaranteed not be trapped in the wrong strategy forever. Furthermore, since the exploitation stage starts early, one is often likely to use the best strategy a large fraction of the time.

The value of ϵ is an algorithm parameter. For example, in practical settings, one might set $\epsilon = 0.1$, although the best choice of ϵ will vary with the application at hand. It is often difficult to know the best value of ϵ to use in a particular setting. Nevertheless, the value of ϵ needs to be reasonably small in order to gain significant advantages from the exploitation portion of the approach. Selecting a small value of ϵ, however, poses a significant challenge in settings where new slot machines (items) constantly enter the system. In such cases, one would explore this new slot machine only occasionally and miss an opportunity to obtain a better payoff.

To provide a specific example of the challenge posed by this situation, consider a setting in which the slot machines correspond to the different items, and the users are clustered into similar groups based on their specified profiles. The ϵ-greedy strategy is executed independently for each group of similar users. Whenever an opportunity arises to serve a recommendation to a user, the accumulated statistics of that user's group are used to choose the item with the use of the ϵ-greedy algorithm. At some point, a new item enters the system, which would be of great interest to John's group. However, at small values of ϵ, this item will be shown to John's group *very* occasionally, especially when the number of other items is very large. In a system with $10,000$ items and $\epsilon = 0.1$, the new item would be shown to John's group approximately once every $100,000$ trials of that group. This means that a large number of trials would be wasted before the relevance of this item to John's group is learned.

13.3.3 Upper Bounding Methods

Even though the ϵ-greedy strategy is better than the naive strategy in dynamic settings, it is still quite inefficient at learning the payoffs of new slot machines. In a dynamic recommendation setting, this problem is pervasive because new items enter the system all the time. In upper bounding strategies, the gambler does not use the mean payoff of a slot machine. Rather, the gambler takes a more optimistic view of slot machines that have not been tried sufficiently, and therefore uses a slot machine with the best *statistical upper bound* on the payoff. Note that rarely tested slot machines will tend to have larger upper bounds (because of larger confidence intervals) and will therefore be tried more frequently. Furthermore, one no longer needs to explicitly use a parameter ϵ to divide the trials into two categories; the process of selecting the slot machine with the largest upper bound has the dual effect of encoding both the exploration and exploitation aspects within each trial.

An important problem here is that of determining a statistical upper bound on the payoff of each machine. This can often be achieved with the help of the central limit theorem, which states that the sum of a large number of i.i.d. random variables (payoffs) converges to the normal distribution. One can estimate the mean and standard deviation of the normal distribution over various trials, and then set the upper bound of each slot machine at the required level of statistical confidence. Note that new slot machines will have large confidence intervals, and therefore the upper bounds will also be correspondingly large. Increasing the number of trials reduces the width of the confidence interval and therefore the upper bounds will tend to reduce over time. When a new slot machine enters the system, it will often be tried repeatedly, until its upper bound falls below that of one of the existing slot machines. One can regulate the trade-off between exploration and exploitation by using a specific level of statistical confidence. For example, an algorithm at 99% level of statistical confidence will perform a larger proportion of exploration as compared to an algorithm at 95% level of statistical confidence.

Such upper bounding strategies have been used recently for designing recommendation algorithms [348]. Many of these algorithms use the contextual features of the users and recommendation setting to design the various multi-arm bandit strategies for exploration and exploitation of the search space. The basic idea is that the gambler is shown a feature vector relevant to that trial (e.g., user or item profile in recommender system), and the gambler makes decisions on the slot machine (choice of recommendation strategy or choice of item) based on the knowledge of the feature vector. Such algorithms are also referred to as *contextual bandit algorithms*. The main goal of the gambler is to learn how the contextual features and the rewards on the arms relate to one another based on previous experience. The contextual feature vectors can be extracted from side-information such as user-profiles or the Web page on which the recommendation is shown. Therefore, contextual features provide a useful tool to incorporate various types of personalization in multi-armed bandit algorithms.

Consider a setting where the arms of the slot machines correspond to recommending different items. The basic idea of these algorithms is to use the following steps repeatedly:

1. **(Incremental) training:** Train a classification or regression learning model based on past history of feature-payoff pairs to learn the expected payoff of each arm. In most cases, this phase is executed incrementally, as new feature-payoff pairs enter the system over the time. Whenever a particular arm is selected by the recommender system, its feature attributes and payoff value is added to a training data set *that is specific to the corresponding arm*. Therefore, there are as many training data sets (and incrementally updated models) as the number of arms. The number of training

examples for each arm is equal to the number of times that the arm was played in the past. A separate model is constructed for each arm using its training data. It is desired to use a probabilistic or statistical learning algorithm that outputs the expected payoff and an estimated standard deviation (or maximum deviation) measure of the payoff of each arm (item) for a particular feature vector (context). Note that arms corresponding to newly added items will have smaller training data sets. Smaller training data sets will lead to larger estimated deviation of prediction. In general, there are two criteria to keep in mind while selecting the base model for payoff prediction:

- The base model should be incrementally updatable because new feature-payoff pairs are continually added to the training data.
- The base model should have the ability to output some measure of (or tight upper bound on) the expected error of prediction.

2. **Upper-bound estimation:** For the current contextual profile being shown to the recommender system, use the learned model to construct an upper bound on the expected payoff of each arm. The upper bound is computed as a linear sum of the expected payoff and an appropriate multiple of the standard deviation. In some cases, a tight upper bound on the maximum deviation is used instead of the standard deviation. The choice of deviation measure often depends on the ease of computing such measures with the model at hand.

3. **Recommendation:** Select the arm with the largest upper bound. Recommend the corresponding item to the user.

These steps are executed continuously over time, as recommendations are made and additional examples are added to the training data. In cases, where the payoff is a binary value (e.g., clicking or not clicking a link), a classification model may be used instead of a regression model.

The LinUCB algorithm is an upper bounding algorithm, which is based on a similar approach [348]. This approach uses a linear regression algorithm to learn the expected payoff. Consider a setting, where the ith arm has been played n_i times so far. In particular, if $\overline{X}$ is a d-dimensional (row) vector corresponding to the current context, D_i is the $n_i \times d$ feature matrix of the training data set of the ith arm, and $\overline{y_i}$ is the n_i-dimensional payoff (column) vector of the ith arm, then one can use ridge regression to predict the expected payoff of $\overline{X}$ with the ith arm as follows:

$$\text{Payoff}_i = \underbrace{\overline{X}}_{d \text{ features}} \underbrace{\left[(D_i^T D_i + \lambda I)^{-1} D_i^T \overline{y_i}\right]}_{d \text{ coefficients}} \tag{13.2}$$

Here, $\lambda > 0$ is the regularization parameter and I is a $d \times d$ identity matrix. Furthermore, a tight upper bound on the expected deviation can be quantified under conditional independence assumptions on the payoff (response) variables with respect to the feature variables. In particular, it can be shown [348] that with probability at least $(1 - \delta)$, the following is true for the binary payoff[1] setting:

$$\text{Deviation}_i \leq \left(1 + \sqrt{\ln(2/\delta)/2}\right) \cdot \sqrt{\overline{X}(D_i^T D_i + \lambda I)^{-1} \overline{X}^T} \tag{13.3}$$

[1]If the payoffs lie in the range $[0, \Delta]$, then the deviation also needs to be scaled up by Δ.

The deviation will reduce when D_i has a larger number of rows (training examples), because the entries in $(D_i^T D_i + \lambda I)^{-1}$ typically become smaller as the entries in $D_i^T D_i$ become larger. Furthermore, the deviation increases for smaller values of δ. The arm with the largest value of $\text{Payoff}_i + \text{Deviation}_i$ is selected as the relevant one. By increasing or decreasing δ, one can select the desired point on the exploration-exploitation trade-off curve. In practice, one directly uses $\alpha = (1 + \sqrt{\ln(2/\delta)/2})$ as the relevant input parameter rather than δ, although the former's relationship to δ can be used to provide some intuitive guidance in selecting it. It is noteworthy that both $D_i^T D_i$ and $D_i^T \overline{y_i}$ can be maintained incrementally because they can expressed as linear sums of functions of the attributes/payoffs of individual training points. Nevertheless, it is still required to invert the $d \times d$ matrix $(D_i^T D_i + \lambda I)$ during each prediction. In cases where d is large, the inversion can be done periodically.

In practice, one can use virtually any probabilistic algorithm that outputs a robust measure of the expected payoff and maximum deviation for a given feature vector. It is noteworthy that LinUCB uses a tight upper bound on the deviation rather than the standard deviation because it is easier to estimate. In many settings, it may be desirable to present more than one recommendation at a time in the form of a ranked list. The simplest approach may be to use the top-k upper bounds as an approximation. A more sophisticated approach is to use the *slate* setting, and it is discussed in detail in [290].

13.4 Group Recommender Systems

Group recommender systems are designed to address scenarios in which items are consumed by groups of users, rather than a single user. Some examples of these scenarios and the systems developed to deal with them include the following:

1. *Movie domain:* In many scenarios, a group of users might wish to go out to see a set of movies. The recommendations must therefore be tailored to the composition of the group. An example of such a recommender system is *PolyLens* [168], which provides recommendations to groups of users. *PolyLens* can be viewed as an extension of the *MovieLens* system.

2. *Television domain:* Like movies, one might want to recommend programs to watch for groups of users. An example of such a television program recommender, which is based on user profile merging, is discussed in [653].

3. *Music domain:* Although it is less common for groups of users to hear music together, such scenarios arise when the music is to be played in a group setting, such as a fitness center or gym. An example of such a system is the *MusicFX* [412] group recommender system.

4. *Travel domain:* The travel domain is perhaps the most common one for group recommendations. This is because it is common for groups of tourists to make travel plans together. Some examples of such systems include *Intrigue* [52], *Travel Decision Forum* [272], and *Collaborative Advisory Travel System (CATS)* [413].

These processes lead to a natural question: why would one not use straightforward averaging to recommend items to a group in these situations? After all, if the goal is to maximize the overall utility, then using the average seems to be the most effective option. However, users can often influence one another based on social phenomena, such as *emotional contagion* and *conformity* [409]. These phenomena can be defined as follows:

1. *Emotional contagion:* The satisfaction of various users can have an impact on one another. For example, if a set of users are watching a movie together, and if some members of the group are not enjoying the movie, this can have a contagious effect on other users. In such cases, averaging does not work very well because the users infect one another with their tastes, and the final experience of the group may be very different from what the average rating might indicate.

2. *Conformity:* Conformity is closely related to the notion of emotional contagion, in that the *expressed* opinions of users have an impact on one another. However, the social phenomenon is slightly different, in that users either want to consciously have similar opinions as their peers (in spite of having a hidden difference of opinion), or their opinions unconsciously change because of peer influence. As a result, the final experience of the group may deviate significantly from what an average rating might indicate.

These two social phenomena, which are related to *social choice theory*, have significant effects on the performance of recommender systems. As a result, the averaging strategy will often not work well. For example, an evaluation of an averaging-based strategy for television recommendation service was performed in [654], and it was shown that the recommender performs well when the group had homogeneous tastes, but it does not perform quite as well when the tastes vary widely. Therefore, it is crucial to be able to use social phenomena in the modeling process. In addition, group recommenders are generally defined differently, depending on whether they are designed in the collaborative, content-based, or knowledge-based settings. Although the general principles of group recommendation in the collaborative and content-based settings are similar, the principles of knowledge-based systems are quite different. In the following, we will study these different settings.

13.4.1 Collaborative and Content-Based Systems

The collaborative and content-based systems are generally quite similar in terms of the approach used for creating the group recommendations. The general approach comprises the following two steps:

1. Perform the recommendation independently for each user as in any collaborative or content-based system. For a given group and a given universe of items, determine rating predictions for each user-item combination.

2. For each item, aggregate the ratings from the various members of the group into a single group rating by using a aggregation function of the ratings predicted for each member of the group. This function might use a simple weighted averaging over group members, an aggregation approach based on principles from social choice theory, or a combination of the two. All the items are then ranked for the group based on the predicted group rating of each item.

The main difference between the various methods is the implementation of the second aggregation step. A variety of different strategies are used to aggregate the diverse ratings into a single value in the second step. These strategies are as follows:

1. *Least misery strategy:* In the least misery strategy, the overall rating suggested to the group is the lowest rating of any member of the group. The basic idea of this approach is to prevent the negative effects of social contagion and conformity. An example a system using this approach is *PolyLens* [168].

2. *Weighted averaging:* This approach uses the average rating of the individual ratings, and a weight is associated with each individual. The weight is often used to model specific types of situations that prevent extreme dislike or infeasibility. For example, a casino resort should not be suggested as a tourist destination to a group containing a child, and a physically strenuous trip should not be suggested to groups containing one or more disabled individuals. Providing greater weights to the preferences of such individuals automatically increases the overall acceptability and feasibility of the group recommendation. A variation of such a strategy was used in *Intrigue* travel recommender [52]. It has also been suggested [168] that ratings from experts might be assigned greater weight. Finally, it is also possible to combine the least misery strategy with the averaging strategy by using a weighted summation of the least misery and averaging prediction over each item.

3. *Average without misery:* This approach averages the predicted ratings of the group members after excluding the ratings of individuals with the lowest ratings. Note that this approach tends to have an opposite focus to the least misery strategy, because it averages only over members who experience the greatest pleasure over a specific item. This type of approach was used in the *MusicFX* system [412]. When considering this approach, it is worth noting that pleasurable experiences can be emotionally contagious in the same manner as unhappy experiences.

A variation of the averaging approach is to use the median instead of the mean. The advantage of using the median is that it is less susceptible to noise and outliers. For example, a single highly negative rating may affect the mean significantly but it may not affect the median much. Such an approach is particularly useful when users are aware of the recommendations that other users are giving and respond by attempting to selectively provide highly positive or negative ratings that would have an outsized influence on the overall group recommendation. As a result, the average no longer remains representative of the group rating. Such an approach is used by *Travel Decision Forum* [272]. A variety of other aggregation strategies are suggested in [407]. Refer to the bibliographic notes.

13.4.2 Knowledge-Based Systems

The aforementioned systems are all based on ratings specifications. However, knowledge-based systems are not based on user ratings, but rather on the specification of user *requirements*. Therefore, the natural approach in such systems is to have each user specify his or her requirements, which are aggregated into a single set. Then, the item that fulfills most of these requirements is recommended. Such an approach is used by the *Collaborative Advisory Travel System (CATS)* [413]. Such systems also allow interactive feedback that allows the group to explore its interests in an interactive style. Knowledge-based systems are particularly well suited to group recommendations, as they allow the group to come to a consensus in an interactive way before actually consuming the item. This reduces the likelihood of dissatisfaction in the final recommendation. Although knowledge-based systems are designed for complex product domains, they are also useful in the context of complex *user* domains. A group recommendation setting can be viewed as a complex user domain. Knowledge-based recommender systems are discussed in Chapter 5.

Table 13.1: Effects of multiple criteria in defining similarity

Criterion ⇒ / User ⇓	Visual Effects	Plot	Overall
Sayani	3	9	7
Alice	9	3	7
Bob	8	3	5

13.5 Multi-Criteria Recommender Systems

In many recommendation applications, users may be interested in items on the basis of different criteria. For example, in a movie recommender system, one user may be interested in visual effects, whereas another user may be interested in the plot. In such cases, the overall rating is often a poor reflection of the user's overall choices. Consider the hypothetical example illustrated in Table 13.1. In this case, three users have expressed their ratings for the movie *Gladiator* based on visual effects, plot, and overall rating. Note that the overall rating is specified directly by the users, and might not necessarily represent an average of all the ratings. Each rating value is specified on a scale from 1 to 10. It is clear that Alice and Sayani have exactly the same overall rating, but their patterns of ratings for the plot and visual effects are very different. On the other hand, Alice and Bob are slightly different in their overall ratings, but have similar ratings on the visual effects and plot. Therefore, for any peer-based prediction method, Alice and Bob should be considered more similar than Alice and Sayani. By using similarity computations based only on the overall rating, one can often obtain misleading predictions.

The overall rating in a multi-criteria system may be either explicitly specified by users, or it may be derived with the use of a global utility function (e.g., simple averaging). In cases where an overall rating is specified by users, it is possible to learn a user-specific utility function with the use of linear regression methods such as those discussed in Chapter 5 on knowledge-based recommender systems. For cases in which the overall rating is not specified by users, the items can be ranked directly by integrating the predictions from the various criteria without computing an overall rating. In other cases, one can implicitly average over various criteria in order to create the overall rating. If needed, the various criteria may be weighted using domain-specific knowledge (e.g., utility functions).

It should be pointed out that multi-criteria recommender systems are inherent to knowledge-based systems, which are designed for complex product domains such as cars. Such products have multiple criteria such as performance, interior design, luxury options, navigation, and so on. In such domains, users wish to rank items based on whether they satisfy certain user-specified criteria. As these methods are already discussed in Chapter 5, this chapter will primarily focus on content-based and collaborative filtering methods.

In the following, we will discuss some of the common methods used in multi-criteria recommender systems. Refer to the bibliographic notes for an up-to-date discussion of recent methods. For the purpose of the following discussion, we will assume that there are a total of c criteria, indexed by $\{1, 2, \ldots, c\}$. The $m \times n$ ratings matrix according to the kth criterion is denoted by $R^{(k)}$, and the rating of user i for item j in $R^{(k)}$ is denoted by $r_{ij}^{(k)}$. In the event that the user also specifies overall ratings, then the corresponding rating matrix is denoted by $R^{(0)}$, and the corresponding value of the overall rating of user i for item j is denoted by $r_{ij}^{(0)}$.

13.5.1 Neighborhood-Based Methods

Neighborhood-based methods can be easily adapted to work with multi-criteria systems because of the ease with which multiple criteria can be incorporated within the similarity function. Most of the existing neighborhood-based methods leverage *user-based* collaborative filtering methods rather than item-based collaborative filtering methods. However, it is possible, in principle, to generalize item-based methods to multi-criteria scenarios using similar techniques. In the following, we will discuss only user-based neighborhood methods because of its wider acceptance and available experimental results.

Let $\text{Sim}^k(i, j)$ represent the similarity between users i and j over criterion k, where $k \in \{1 \ldots c\}$. Furthermore, we will assume that the overall ratings matrix $R^{(0)}$ is available, and the corresponding similarity between users i and j is denoted by $\text{Sim}^0(i, j)$. Then, the neighborhood-based method can be implemented as follows:

1. Compute the similarity $\text{Sim}^k(i, j)$ between each pair of users i and j for each $k \in \{0 \ldots c\}$. Any of the methods introduced in Chapter 2, such as the Pearson correlation coefficient, may be used for computing $\text{Sim}^k(i, j)$.

2. Compute the aggregated similarity $\text{Sim}^{aggr}(i, j)$ between each pair of users i and j by aggregating the similarity values over the various criteria using an aggregation function $F(\cdot)$:

$$\text{Sim}^{aggr}(i, j) = F(\text{Sim}^0(i, j), \text{Sim}^1(i, j), \text{Sim}^2(i, j), \ldots \text{Sim}^c(i, j)) \tag{13.4}$$

   Determine the k-closest peers of each user with the aggregated similarity.

3. Use the similarity weighted values of the (overall) ratings of each peer of a user t for an item j in order to predict the rating of user t for item j. Typically, the approach is combined with row-wise mean-centering to prevent user-specific bias. Thus, this approach is equivalent to that of using Equation 2.4 of Chapter 2 on the overall ratings matrix $R^{(0)}$, except that aggregated similarities $\text{Sim}^{aggr}(\cdot, \cdot)$ are used for peer determination and weighting purposes within Equation 2.4.

It is noteworthy that the aggregation function of Equation 13.4 also uses $\text{Sim}^0(i, j)$ (similarity based on overall ratings) in the computation. The main differences among various methods arise in terms of how the aggregation of Equation 13.4 is computed. The common methods for aggregation are as follows:

1. *Average similarity:* This approach [12] is based on the averaging of the predictions of the $(c+1)$ different ratings (including the overall rating). Therefore, the function $F(\cdot)$ of Equation 13.4 is defined as follows:

$$\text{Sim}^{aggr}(i, j) = \frac{\sum_{k=0}^{c} \text{Sim}^k(i, j)}{c + 1} \tag{13.5}$$

2. *Worst-case similarity:* This approach [12] uses the smallest similarity across all the criteria (including the overall rating). Therefore, we have:

$$\text{Sim}^{aggr}(i, j) = \min_{k=0}^{c} \text{Sim}^k(i, j) \tag{13.6}$$

3. *Weighted aggregation:* This approach [596] is a generalization of the averaging technique and uses a weighted sum of the similarities across the different criteria.

Let $w_0 \ldots w_c$ be the weights of the various criteria. Then, the aggregated similarity is defined as follows:

$$\text{Sim}^{aggr}(i, j) = \sum_{k=0}^{c} w_k \cdot \text{Sim}^k(i, j) \tag{13.7}$$

The value of w_i determines the weight of criterion i, and the weights can be determined using straightforward parameter-tuning techniques such as cross-validation (cf. Chapter 7).

In addition to using similarities, it is also possible to use *distances* both for peer computation, and for the final step of weighted rating prediction. Note that similar items will have *smaller* distances, necessitating conversion of the distances into similarities in a heuristic way in order to perform the weighting. For any pair of users, the distances are computed based only on the items that the two users have rated in common. The distances are computed separately for each of the items by aggregating across various criteria. The distances across various items are averaged in a second aggregation step.

How is the first step of computing the distance $\text{ItemDist}^{aggr}(i, j, q)$ between users i and j with respect to a particular item q executed? Note that item q must be rated by both users i and j for this distance to be computed at all. A natural approach is to use the L_p-norm, which is defined as follows:

$$\text{ItemDist}^{aggr}(i, j, q) = \left(\sum_{k=0}^{c} |r_{iq}^k - r_{jq}^k|^p \right)^{(1/p)} \tag{13.8}$$

Commonly used values of p are $p = 1$ (Manhattan metric), $p = 2$ (Euclidean metric), and $p = \infty$ (L_∞-norm).

This approach is repeated over each of the items that the users i and j have rated in common. Let this set of items be denoted by $I(i, j)$. The overall distance $\text{Dist}^{aggr}(i, j)$ across all items is defined by the average distance over all items in $I(i, j)$:

$$\text{Dist}^{aggr}(i, j) = \frac{\sum_{q \in I(i,j)} \text{ItemDist}^{aggr}(i, j, q)}{|I(i, j)|} \tag{13.9}$$

One can convert the distances into similarity values with the use of simple kernel computations or inversion tricks:

$$\text{Sim}^{aggr}(i, j) = \frac{1}{1 + \text{Dist}^{aggr}(i, j)} \tag{13.10}$$

After the similarity values have been computed, one can leverage the user-based collaborative filtering methods as discussed above.

13.5.2 Ensemble-Based Methods

All the aforementioned methods make changes to a specific *algorithm*, such as the neighborhood algorithm, in order to perform the recommendations. However, it is possible to use ensemble-based methods, which can leverage any existing technique, to perform the recommendations [12]. The basic approach contains two steps:

1. For each value of $k \in \{1 \ldots c\}$, use any off-the-shelf collaborative filtering algorithm on ratings matrix $R^{(k)}$ to fill in the ratings for criterion k.

2. For each user i and item q, for which the ratings have been predicted, combine the predictions $\hat{r}(1)_{iq} \ldots \hat{r}_{iq}^{(c)}$ across the various criteria using an aggregation function $f()$ as follows:
$$\hat{r}_{iq}^{(0)} = f(r_{iq}^{(1)} \ldots r_{iq}^{(c)}) \tag{13.11}$$
The computed aggregation provides the overall predicted rating. The recommended items are then ranked for user i based on the overall predicted ratings.

The construction of the aggregation function $f()$ remains to be explained. There are three common techniques suggested in [12]:

1. *Domain-specific and heuristic methods:* In this case, the aggregation function is set by the domain expert depending on the perceived importance of the various criteria. The simplest possible approach is to use the average of the predicted ratings over the various criteria.

2. *Statistical methods:* These represent linear and non-linear regression methods. For example, the overall predicted rating can be expressed as a linear weighted sum of the predicted ratings over various criteria:
$$\hat{r}_{iq}^{(0)} = \sum_{k=1}^{c} w_k \cdot r_{iq}^{(k)} \tag{13.12}$$
The values of $w_1 \ldots w_c$ can be learned using linear regression techniques, as discussed in section 6.3 of Chapter 6. Note that the observed values of the ratings across various criteria can be used as the training data to learn the weights.

3. *Machine-learning methods:* This approach is not very different in principle from the second approach. Instead of using regression, any machine-learning method (such as a neural network) can be used. Note that simpler versions of neural networks can also be used to approximate linear regression. However, a neural network provides greater power in modeling arbitrarily complex functions.

The aforementioned discussion is based on the assumption of a global aggregation. However, it is also possible to learn user-specific or item-specific aggregation functions, if sufficient number of observed ratings about users and items are available. The ensemble-based approach is simple to implement because it provides the ability to use off-the-shelf tools in various phases of the process. This aspect of ensemble methods also provides it with greater flexibility in performing model selection, and tuning the system with an appropriate choice of learners.

13.5.3 Multi-Criteria Systems without Overall Ratings

The aforementioned methods require the availability of overall ratings in order to perform the recommendations. In cases where overall ratings are not available, the methods discussed in the previous sections cannot be used in their current form. However, one can still use the first step of the ensemble-based method discussed in the previous section. The main difference is that the second step of aggregating the predicted ratings needs to performed without any available learning data. Therefore, methods such as linear regression, nonlinear regression, neural networks, or other machine-learning methods are no longer possible. However, it is still possible to use heuristic and domain-specific combination functions in

the aggregation step. The items can then be ranked on the basis of the aggregated value. A second approach for presenting the items to the user is to leverage the pareto-optimality of the predicted ratings across the various criteria. Only the pareto-optimal items are presented to the user along with an explanation of why they are presented. The bibliographic notes contain pointers to various multi-criteria systems that do not assume the availability of overall ratings.

13.6 Active Learning in Recommender Systems

Recommender systems are heavily dependent on historical data provided by the user. However, ratings matrices are sometimes excessively sparse, causing challenges in providing meaningful recommendations. This is especially true at start-up time, where *cold-start* problems are often encountered. In such cases, it is important to quickly acquire more ratings to build up the ratings matrix. The process of acquiring ratings is time-consuming and costly because users are often not willing to voluntarily provide ratings without a perceived benefit. Indeed, it has been argued [303] that users are willing to share private information in collaborative filtering applications only when they are fairly compensated. This implies that there is an inherent cost (often implicit) to acquiring ratings. An active learning system chooses specific user-item combinations for which to acquire ratings in order to maximize the accuracy of predicted ratings. For example, consider a scenario of a movie recommender system in which many action movies have already been rated, but no comedy movies have been rated. In such cases, it is intuitively fruitful to actively acquire ratings of comedy movies rather than action movies in order to maximize prediction accuracy. This is because the *incremental* improvement in accuracy by acquiring further ratings of other action movies is likely to be less than that obtained by acquiring ratings of comedy movies. After all, one can already predict the ratings of action movies reasonably well, whereas one cannot predict the ratings of comedy movies very well with the available ratings. The problem here is that one cannot acquire the rating of an arbitrary user-item combination. For example, a user who has not consumed an item cannot be reasonably expected to provide a rating.

Active learning is commonly used in classification applications [18]; therefore, the applicability of the approach to content-based methods is obvious. After all, content-based methods are essentially classification problems on user-specific training data. In the case of collaborative filtering applications, content or genre information is typically not specified, and one must make such predictions with the use of the currently available ratings matrix. In its simplest form, one can formulate the ratings acquisition problem as follows:

Given a ratings matrix R, a cost budget C, and a cost-per-acquisition c, determine the set of user-item combinations for which the ratings must be acquired in order to maximize the prediction accuracy.

It is evident that the active learning formulation for classification is similar to that of collaborative filtering. In the case of classification, *labels* of training points are queried. In collaborative filtering, the *ratings* of user-item combinations are queried. As collaborative filtering is a generalization of the classification problem (cf. Figure 1.4 of Chapter 1), the active learning methodologies of classification also generalize to the collaborative filtering scenario. However, there is one key difference between collaborative filtering and classification. In classification, it is assumed that an oracle exists that provides the label of any queried data point. This assumption cannot be made in collaborative filtering. For example,

if a user has not consumed an item, she cannot be expected to provide a rating for it. Nevertheless, the principles of active learning in collaborative filtering applications are similar to those in classification, at least in terms of determining which user-item combinations are most valuable to acquire. In many cases, incentives can be provided to a user to rate a specific item. For example, the merchant might offer a free product in exchange for a specific number of ratings from a particular user.

The simplest approach to active learning is to query for items that have been rated sparsely by the users. This can naturally help in the cold-start setting. However, such an approach is useful only in the initial stages of the recommender system setup. In later stages, more refined techniques are required in which the entries of the matrix are selected on the basis of the particular *combination* of users and items. Such methods are based on ideas already available in the classification literature.

Active learning is still an emerging area in the topic of collaborative filtering, and there are relatively few methods proposed in this area. Therefore, this section will briefly discuss two common methodologies [18, 22] used in classification and their applicability to collaborative filtering applications. These two methods are *heterogeneity-based models* and *performance-based models*. In the former case, the data points (user-item combinations) are queried, for which their *predicted* rating values are the most *uncertain before* performing the query. In performance-based models, the data points are queried, so that the prediction accuracy on the *remaining entries* provide the *best expected performance or certainty* after incorporating the newly queried rating in the matrix.

13.6.1 Heterogeneity-Based Models

In heterogeneity-based models, the goal is to query for the rating of the user-item combination for which the predicted rating is the most uncertain *before* performing the query. The specific method for judging the level of uncertainty depends on the model at hand. For example, if a numeric rating is predicted with a specific variance, each user should be queried with the item with the largest predicted variance. In the case of a binary rating prediction with a Bayesian approach, the item q whose posterior probability p_q is closest to 0.5 (i.e., smallest value of $|p_q - 0.5|$) is queried. Some specific examples of how this approach may be used in the context of specific models are as follows:

1. In a user-based neighborhood approach, the variance of the prediction of user-item combination (i, q) can be computed as the sample variance of the ratings of the peer users of i for item q. If none of the peer users have rated item q, the sample variance is ∞.

2. In an item-based neighborhood approach, the variance of the prediction can be computed from the ratings of user i of the most similar items of q. If user i has not rated any of the most similar items of q, the sample variance is ∞. Therefore, the approach tends to guide the user towards rating different items and naturally increases the coverage of the recommender system. In this sense, the approach is also able to adjust well in the cold-start setting.

3. In a Bayesian model, a Bayes classifier (cf. Chapter 3) is used to predict ratings. Consider the case of binary ratings, in which the prediction of a value of 1 has posterior probability p_q. In this case, the uncertainty is quantified as $1 - |p_q - 0.5|$. The item with the largest uncertainty value is selected for querying.

4. One can use multiple models to predict the ratings. A rating is said to be uncertain when the different models have different predictions. The variance of the predictions over the different models can be used to quantify the uncertainty.

The aforementioned methods are simple adaptations of techniques in the classification literature. One can adapt most collaborative filtering algorithms in a natural way to compute the uncertainty. In collaborative filtering, some additional factors can be combined with the uncertainty level in a heuristic way (e.g., multiplicatively):

1. One can include a factor for the probability that a user is likely to rate an item. This is because users cannot provide ratings for items they have not consumed. Consider an implicit feedback matrix in which a value of the entry is 1 if a user has rated an item (irrespective of actual rating value) and 0, otherwise. The predicted "rating" with the use of any collaborative filtering algorithm provides a probability that a user will actually rate the item.

2. It is suggested in [513] that very popular items should not be queried because their ratings are often not representative of the other items.

Note that few experimental results exist on how active learning methods actually perform in the context of collaborative filtering. Therefore, this area is open to significant opportunities for further research.

13.6.2 Performance-Based Models

The goal of querying for ratings is to increase prediction accuracy and also reduce the uncertainty of prediction on the *currently available* entries. In performance-based models, the data points are queried so that the prediction accuracy on the *remaining entries* provides the *best expected performance or certainty after* incorporating the newly queried rating in the matrix. Note that uncertainty-based models focus on the prediction characteristics of the currently queried instance, whereas performance-based models focus on the *differential* impact of the *added* instance to the predictions of the *currently available* entries. Determination of what would happen after querying the rating of a user-item combination is challenging because the *expected performance* must be computed before actually querying the rating. Bayesian methods are used to compute this expected performance. The corresponding techniques are described in [18, 22].

13.7 Privacy in Recommender Systems

Collaborative filtering applications are heavily dependent on the collection of feedback from multiple users. In collaborative filtering applications, users need to specify ratings for items. These ratings reveal important information about user interests, their political opinions, sexual orientation, and so on. This revealing of private information that comes with rating items poses numerous challenges because it makes users less willing to contribute ratings.

All privacy-preservation methods change the data in some way so as to reduce its accuracy of representation. This is done in order to increase the privacy. The trade-off is that the data becomes less accurately represented. Therefore, mining algorithms are no longer as effective. Two classes of techniques are used to preserve privacy:

1. *Privacy at data collection time:* In these techniques, the data collection approach is modified so that individual ratings are not collected. Rather, distributed protocols [133] or perturbation techniques [35, 38, 484, 485] are used to collect the data *only in a perturbed way or in the aggregate*. Typically, specialized (secure) user-interfaces and data collection plug-ins are required in order to implement the approach. Furthermore, specialized data-mining methods are used on the collected data, because many of these techniques use aggregate distributions for mining, rather than individual data records.

   The advantage of such an approach is that users are assured that no single entity has access to their private data, at least in its exact form. Although privacy at data collection provides the strictest form of privacy, much of the work in this area is at the research stage only. To the best of our knowledge, there are no large-scale commercial implementations of such systems. This is, in part, because such systems typically require more effort from the user in terms of gaining access to specialized interfaces/infrastructures and more effort from the data miner after the aggregated data becomes available.

2. *Privacy at data publication time:* In most practical settings, a trusted entity (e.g, Netflix or IMDb) has access to all the ratings data it has collected over time. In such cases, the trusted entity might wish to *publish* the data to the broader technical community to enable further advancements in the field of collaborative filtering. A specific example of such a publication was the Netflix Prize data set, which was released after de-identification of the ratings. In such cases, models like k-anonymity [521] are used to preserve privacy. Typically, such methods use *group-based* anonymization techniques in which records belonging to groups of a minimum size become indistinguishable. This is achieved by carefully perturbing selected attributes of the data records so that one cannot join such records with publicly available information in order to exactly identify the subjects of the data records. Such systems are more common, and have wider applicability than the first scenario.

The two aforementioned models have different trade-offs. The first model provides stronger privacy guarantees because the individual's ratings are not stored anywhere, at least in their exact form. In some cases, the ratings are stored only in *an aggregate sense*. Therefore, the approach provides greater privacy guarantees. On the other hand, it is generally harder to use off-the-shelf collaborative filtering algorithms with such forms of data collection. This is because the data is either perturbed very highly, or the fundamental *representation* of the data has been changed to some aggregate form. In the case of methods using group-based anonymization, the privacy guarantees are typically weaker. On the other hand, the released data records are typically in the same format as the original data. Therefore, it is easier to make use of off-the-shelf collaborative filtering algorithms in these cases. The following passage provides a brief overview of group-based anonymization models.

Group-based anonymization methods are typically used by trusted entities at *data publication time*. The typical goal of publishing entities is to prevent identification of the subjects of the data records. For example, when Netflix released their ratings data set, the subjects of the data records were de-identified. In addition, the attributes are typically perturbed in such a way that groups of data records become indistinguishable. The basic idea in these methods is to perturb the data records sufficiently that attackers cannot match the records with other publicly available data sources in order to determine the identity of the subjects of the records. Some common models for perturbing the data records in group-wise fashion include k-anonymization [521], condensation [27], ℓ-diversity [386], and t-closeness [352].

The reader is encouraged to refer to the bibliographic notes for further details related to common privacy-preservation methods. In the following, we briefly discuss a condensation-based method that is easy to apply to the collaborative filtering setting. We will also discuss some challenges that surface when these methods are used for high-dimensional data.

13.7.1 Condensation-Based Privacy

The condensation-based approach was originally designed for multi-dimensional data records, which are *completely specified* [27]. However, the approach can easily be used for incompletely specified data records as well. One of the inputs to the algorithms is an anonymity level p, which defines the number of rows we wish to be indistinguishable from one another. Larger values of p result in a greater level of anonymity, but they reduce the accuracy of the modified data. Consider an $m \times n$ ratings matrix R, which is incompletely specified:

1. Partition the rows of R into clusters $\mathcal{C}_1 \ldots \mathcal{C}_k$, such that each cluster contains at least m records.

2. For each cluster $\mathcal{C}_r$, generate $|\mathcal{C}_r| > m$ synthetic data records matching the data distribution of the records in the cluster.

Both the two steps need to account for the fact that the rows in the matrix R are incompletely specified. Clustering methods can be modified relatively easily to work for incomplete data. For example, a k-medians algorithm can be modified by using only the specified values of the entries in the median computation. Similarly, the distances are computed using only the specified entries and then normalized by the number of observed dimensions. Similarly, while generating the synthetic data records from $\mathcal{C}_r$, one can use a simple multivariate Bernoulli distribution over the values of the ratings to model each item. This multivariate Bernoulli distribution is derived from the rating distribution of the records in the cluster. One must take care to generate the rating of an item the same number of times that it is present in that cluster.

This method of synthetic data generation comes with two primary advantages. The first advantage is that the data are generated in the same format as that of the original ratings matrix, allowing application of any off-the-shelf collaborative filtering algorithm; the second is that the anonymity of synthetic data is generally harder to compromise. This approach can also be generalized to dynamic settings [27].

13.7.2 Challenges for High-Dimensional Data

Ratings data is typically high-dimensional. For example, a typical ratings matrix may contain thousands of dimensions. Furthermore, some users might easily specify more than 10 or 20 ratings. In such cases, it is harder to preserve the privacy of such users with group-based anonymization methods, even when the data records are perturbed. For example, if a particular source releases a set of de-identified ratings, an attacker might use a different source of ratings that are not de-identified and match the two data sets in order to determine the subjects of the de-identified records. The larger the number of specified ratings is, the easier it is to de-identify the records. It has been shown in [30] that only about 10 to 20 specified values in a row are necessary in order to generate a powerful attack. The well-known Netflix Prize data set was attacked using this methodology [451]. The challenges for high-dimensional data are not trivial, and there are theoretical barriers [30] to the limits of anonymization. The development of new anonymization methods for high-dimensional and sparse data sets remains an open area of research.

13.8 Some Interesting Application Domains

In this section, we will study a number of interesting application domains for recommender systems. The goal of this section is to study the application of recommender systems to various application domains, and the specific challenges that arise in the context of each domain. Some examples are as follows:

1. *Query recommendation:* An interesting question is how Web logs can be used to recommend queries to users. It is not quite clear whether query recommendation should be considered a personalization application because the recommendations are typically *session-specific* (i.e., dependent on the history of user behavior in a short session) and do not use *long-term* user behavior. This is because queries are often issued in scenarios in which user re-identification mechanisms are not available over multiple sessions. We will not discuss this topic in detail, although relevant pointers are included in the bibliographic notes.

2. *Portal content and news personalization:* Many online portals have strong user identification mechanisms by which returning users can be identified. In such cases, the content served to the user can be personalized. This approach is also used by news personalization engines, such as Google News, in which Gmail accounts are used for user identification. News personalization is usually based on implicit feedback containing user behavior (clicks), rather than explicit ratings.

3. *Computational advertising:* Computational advertising is a form of recommendation, because it is desirable for companies to be able to identify advertisements for users based on a relevant context (Web page or search query). Therefore, many ideas from recommendation systems are directly used in the area of computational advertising.

4. *Reciprocal recommender systems:* In these cases, both the users and items have preferences (and not just the users). For example, in an online dating application, both parties (men and women) have preferences, and a successful recommendation can be created only by satisfying the preferences of both parties. Reciprocal recommender systems are closely related to the link-prediction methods discussed in Chapter 10.

This chapter will provide an overview of some of these different applications, with a specific focus on portal content personalization, computational advertising, and reciprocal recommender systems. The basic idea is to give the reader a sense of how recommendation technology can be used in diverse settings.

13.8.1 Portal Content Personalization

Many news portals personalize the news for their users by using their history of past accesses. An example of such a personalization system is the Google news engine. Google has strong user-identification mechanisms with the use of Gmail accounts. This mechanism is used to track the past history of user click behavior. This past history is used to recommend news of interest to users. Similar methods can be used to recommend content to users in many types of Web portals. The main assumption in all these cases is that a user log of their past actions is available.

13.8.1.1 Dynamic Profiler

Dynamic Profiler [636] is a portal content personalization engine that uses a combination of collaborative and content-based techniques. The system can be used for any form of portal content personalization, including news personalization. The approach contains several steps, most of which are periodically repeated to refresh summary statistics which need to be updated over time in order to prevent them from becoming stale. These statistics are used to make recommendations in real time. The overall approach contains the following broad steps:

1. **(Periodically updated)** A sample of documents from the portal is used to create a group of clusters. The clustering is done using a partially supervised clustering scheme [29]. The supervision of the clustering is done with the help of samples of documents belonging to semantically relevant topics. These samples are used as seeds for creating the clusters with a combination of an agglomerative and a k-means approach. As a result, the clusters contain the semantically important categories in the collection.

2. **(Periodically updated)** The user access log is used in conjunction with the aforementioned clusters to create user profiles. The user profile contains a count of a number of accesses of the user to documents belonging to each cluster. Therefore, the user profile is a multidimensional record with as many dimensions as the number of clusters.

3. **(Periodically updated)** The user profiles are then clustered into peer groups with the use of high-dimensional clustering methods. Several high-dimensional clustering methods are discussed in [19].

4. **(Online phase at recommendation time)** A neighborhood-based approach is used in conjunction with these peer groups to perform the recommendations. For any given target user, the frequent categories in the closest clusters form the relevant recommended categories. It is also possible to recommend individual documents to the target user using an approach described below.

It remains to be explained how the final step of performing the recommendations is executed. For a given user, the first step is to determine her closest peer group. This is achieved by computing the distance between her profile and the centroids of the various peer groups. The closest peer group is referred to as her *community*. The frequency of all the documents accessed by this community is efficiently determined from an indexed version of the logs. The most frequently accessed documents in this community, which have also not been accessed by the target user, are then presented as the relevant recommendations.

13.8.1.2 Google News Personalization

The Google news personalization engine [175] is based on a similar problem statement as the dynamic profiler model. Therefore, an implicit feedback data set of user clicks is available in this case. The Gmail accounts of users provide a strong identification mechanism in Google news. When users are signed in and access Web pages, their click behavior is stored. The goal is to use the stored statistics about user clicks to make recommendations to these users from a candidate list L of items. For the time being, we will assume that the candidate list L is given. Later, we will discuss how the candidate list can be generated.

The Google news system uses very different algorithms from *Dynamic Profiler*. Whereas the *Dynamic Profiler* is designed to work for individual Web sites, the Google news system is designed to work in a Web-scale environment. The basic idea of the approach is to use a similarity-based mechanism to make recommendations. As with user-based neighborhood algorithms, a weighted similarity of users to other users who have accessed a particular item is used to make the recommendations. Let r_{iq} be an indicator variable, which takes on the value 1, if user i has accessed item q, and 0, otherwise. Note that r_{iq} can be viewed as the implicit feedback version of a ratings matrix. Similarly, let w_{ij} be the computed similarity between users i and j based on the similarity of their access patterns to Web pages. Then, the *predicted* propensity p_{iq} of user i to access the news item q is defined as follows:

$$p_{iq} = \sum_{j \neq i} w_{ij} \cdot r_{jq} \tag{13.13}$$

Since the rating r_{jq} is assumed to be binary, the prediction propensity p_{iq} can also be binarized using an appropriate threshold. The similarity can be computed in a variety of ways. For example, one might compute the Pearson correlation coefficient or the cosine similarity between the item accesses of the two users.

The aforementioned formula is a straightforward generalization of the user-based collaborative filtering mechanism. Note that it is expensive to compute this predicted propensity in a Web-scale setting because the similarity w_{ij} between every pair of users needs to be pre-computed. The pairwise computation can be rather expensive, and the summation on the right-hand side will also contain as many terms as the number of users. Therefore, the work in [175] also proposes a number of more efficient model-based alternatives. These methods use clustering to speed up computation. Furthermore, clustering methods have some advantages in noise reduction for more effective collaborative filtering.

In model-based techniques, the users are either *probabilistically* or *deterministically assigned* to clusters with similar access behavior. In other words, users who have similar access patterns typically belong to similar clusters with high probability. Two clustering schemes are used, corresponding to *MinHash* and *PLSI*, and either of them can be used to implement the approach. The former users a *hard* assignment of users to clusters, and the latter uses a *soft assignment* to clusters. More details of these methods are discussed later in this section.

Assume that a total of m clusters are defined, and the *fraction* of user i assigned to cluster k is given by f_{ik}. In the case of deterministic clustering, the value of f_{ik} is either 0 or 1, whereas the value of f_{ik} lies in $(0, 1)$ in the case of soft clustering. Then, the propensity of user i to access item q is defined as follows:

$$p_{iq} = \sum_{k=1}^{m} f_{ik} \sum_{j: f_{jk} > 0} r_{jq} \tag{13.14}$$

It is also possible to further refine this formula by incorporating f_{jk}, although this is not mentioned in [175]:

$$p_{iq} = \sum_{k=1}^{m} f_{ik} \sum_{j} f_{jk} r_{jq} \tag{13.15}$$

In the case where the clustering is a hard assignment, such as (*MinHash* scheme), this expression reduces to the following:

$$p_{iq} = \sum_{j} \text{CommonClusters}(i, j) \cdot r_{jq} \tag{13.16}$$

Here, CommonClusters(i, j) correspond to the number of common clusters in which users i and j co-occur. Furthermore, if the clustering is executed only once as a strict partitioning, the value of CommonClusters(i, j) is either 0 or 1. On the other hand, if the clustering is repeated several times with a fast randomized approach, the value of CommonClusters(i, j) is equal to the number of times that users i and j occur in the same cluster. For dynamic data sets, the value of the implicit feedback "rating" r_{jq} can be multiplied with a time-decay value.

In addition, a co-visitation score is added to the scores generated from the clustering-based computation. The co-visitation score is similar in principle to an item-based algorithm. Two items are co-visited when they are visited by the same user within a pre-defined span of time. For each item, the number of (time-decayed) co-visits to every other item is dynamically maintained. For the target user i and target item q, it is determined whether the frequent co-visits of item q are present in the recent item history of user i. For each such presence, a normalized value is added to the recommendation score of Equation 13.14. A specialized data structure is used to implement this operation efficiently.

Clustering Methods

As discussed earlier, *MinHash* and *PLSI* are used as the two clustering schemes. The *MinHash* scheme implicitly clusters users based on intra-similarity defined by the Jaccard coefficient of the sets of items they have visited in common. Although the *MinHash* scheme is a randomized clustering method, it creates deterministic clusters in which the probability of two users belonging to the same cluster is proportional to their Jaccard coefficient. The *PLSI* scheme, on the other hand, is a probabilistic clustering method in which each point is assigned to a cluster with a certain probability. Both the *MinHash* and *PLSI* methods are described in detail in [175]. The work in [175] describes *MapReduce* methods to implement these operations efficiently. The *MapReduce* approach is required to scale the approach to massive settings.

Candidate List Generation

So far, the generation of the candidate list L for a particular target user i has not been described in detail. The candidate list can be generated in one of two ways. The News Front-end can generate a list of candidates based on the news edition, language preferences of the user i, story freshness, customized sections selected by the user i, and so on. Alternatively, the candidates can also be generated as the union of (i) all stories that have been clicked by members of the same cluster as user i, and (ii) the set of stories that have been co-visited with the set of stories in the click-history of user i.

13.8.2 Computational Advertising versus Recommender Systems

In recent years, online computational advertising has received increasing attention because of the greater importance of the internet as a medium for content consumption, information search, and business transactions. These represent typical *activities* that users are often engaged in, and they also represent an opportunity for online advertisers, because the content consumed and the transactions completed can provide a context within which advertisements can be served. An activity that a user is engaged in typically reveals a lot about the user and can be leveraged to target the products specific to the activity at hand. For example, when a user queries a search engine such as Google or Bing with a keyword like

"golf," it is common to see many "sponsored search results," in addition to the true search results. These sponsored search results are advertisements, which are placed by the search engine, and are typically related to the search engine query (i.e., "golf"). This advertisement methodology is referred to as *sponsored search*. In general, the two most common computational advertising models are as follows:

1. *Sponsored search:* In this case, the search engine serves as a *match-maker*, and it serves to place advertisements adjacent to the *query search results* posed by users. The query search results provide the *context* for the advertisements, because the goal of both advertisers and the match-maker is to display advertisements related to the returned search results. This is because the users are more likely to click on contextually relevant sponsored search results. This is helpful in increasing business revenue for the advertiser and also the advertising revenue for the match-maker because match-makers are often paid on the basis of successful click-throughs from the sponsored search result or the number of times the search result is shown. A combination of these payoffs may also be used.

2. *Display advertising:* In this case, publishers of content (e.g., news portals) physically place advertisements on the Web page corresponding to their content. Thus, the content publisher plays the role of the match-maker. The content of Web page serves as the context. For example, a news portal, which is displaying an article on a golf tournaments, might display an advertisement related to golf on the same page. The match-maker can paid by the advertiser with the use of a variety of metrics. For example, the match-maker might be paid for successful click-throughs on the advertisement, a successful transaction on the basis of the advertisement, or the number of times the advertisement is shown (i.e., number of *impressions*). A combination of these payoffs may also be used. Therefore, the model of display advertising shares many similarities with that of sponsored search.

In both cases, an *advertisement* (analogous to an item) is recommended to a *user*, in a specific *context* (defined by either the search results or topic of the page on which the display advertisement is placed). In both cases, the match-maker is a publisher of the content which provides the context for the advertisement. Note that a search query result is also a form of content publication, albeit it is *dynamically generated*, and it is *reactive* to a specific user query. Furthermore, it is in the interest of both the advertiser and the match-maker to ensure that the recommended advertisements are as relevant as possible. This relationship between the various entities in the online advertising scenario is illustrated in Figure 13.3.

There are several important similarities and distinctions between computational advertising methods and recommender systems. The advertisements are like items and the match-maker plays the role of the recommender to the users. However, before discussing ways in which recommendation technology can be used for computational advertising, we need to first understand the distinctions among them. This provides an understanding of the scenarios in which one can effectively use this approach, and the changes needed to achieve these goals. The specific distinctions between recommendations and computational advertising are as follows:

1. In traditional recommender systems, it is in the best interest of a recommender system, such as Amazon.com, to provide the most relevant recommendations to users. Therefore, the user and recommender system interests are perfectly aligned. In computational advertising, the match-maker is *paid* by the advertiser to recommend items to users. While this provides a motivation for publishers (match-makers) to increase

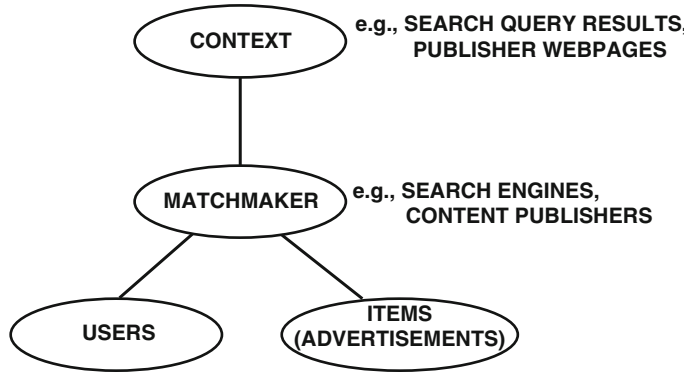


Figure 13.3: Relationships between various parties in the computational advertisement setting

the click-throughs to the advertisement, the interests of the advertiser, publisher, and user might not always be perfectly aligned. This is particularly true when publishers are paid by advertisers on the basis of the number of impressions. The cost models can be understood only in a game-theoretic sense, where the three entities try to maximize their own utility. However, in many cases, the interests of the three entities are more or less aligned.

2. Traditional recommender systems have strong user identification mechanisms. Even when users are anonymized, the *long-term history* of a returning user is known. This is not necessarily true in the case of computational advertising, where it is highly likely that no information is known about the long-term history of a user submitting a search on a search engine. In many cases, the data about past user interactions with advertisements (items) is not even available. This is particularly important because recommendations are all about *personalization*, whereas computational advertising is all about *immediate context*. Nevertheless, in some sites with strong user-identification mechanisms, both context and personalization are important. For example, if an online newspaper has a login mechanism, it can leverage the user identification to provide more relevant advertisement results. Similarly, Google does provide the ability to perform personalized search with the use of Gmail-based identification mechanisms.

3. Items have a long lifetime within a recommendation system. However, in a computational advertisement system, a particular advertisement campaign may have only a very short lifetime. Therefore, advertisements are inherently transient. However, it is possible to logically represent advertisements on the same subject as a "pseudo-item" in order to use recommendation technology.

It is clear from the aforementioned discussion that significant distinctions exist between the computational advertising and recommendation model. Nevertheless, there are a few scenarios in which one can adapt recommendation technology to computational advertising.

For cases in which strong user identification mechanisms are available and the advertiser interests can be properly aligned with publisher interests, the advertising model can be conceived as a recommendation process. The steps required to perform the modeling are as follows:

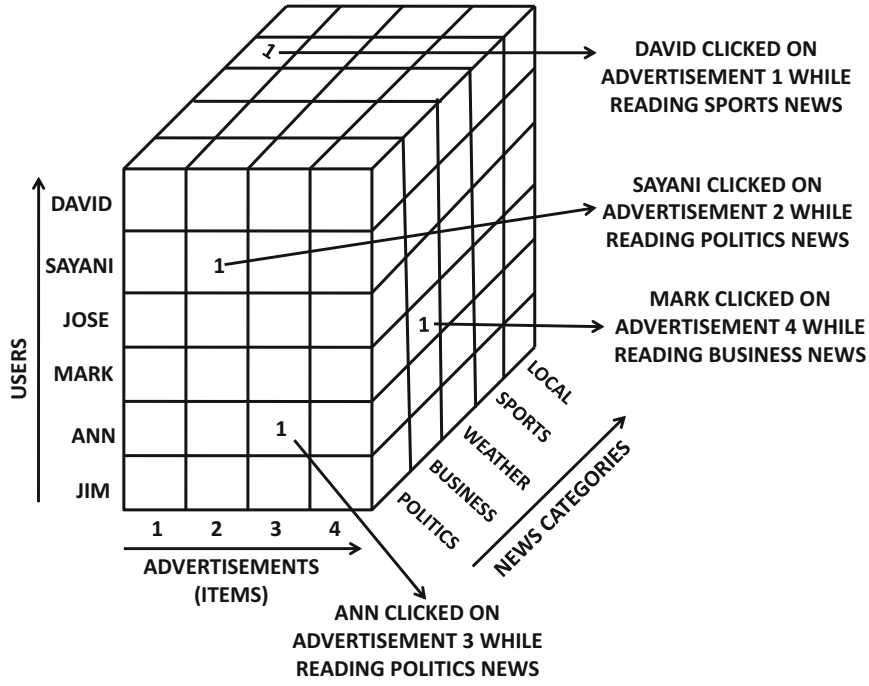


Figure 13.4: Representing advertisements as contextual recommendations for newspaper display advertisements (Note the similarity to Figures 8.1 and 11.6)

1. It is assumed that the set of (identified) users U participating in the system is known in advance for long-term tracking and analytical purposes.

2. Even though advertising campaigns are short-lived, they are all classified into sets of items. For example, two different advertisements on the same type of golf club are treated as a single item. The overall set of items is denoted by I.

3. The user *actions*, such as the act of clicking on an advertisement, are treated as implicit feedback. As advertisements have already been consolidated into items, the user actions can be used to create an implicit feedback between users and items. This implicit feedback could correspond to the frequency of user actions and can be effectively treated as "ratings."

4. All the publication sources (e.g., search phrases, or Web pages) are classified into a discrete set of categories at an appropriate level of granularity. These categories are treated as a fixed set of contexts, denoted by C. As discussed in Chapter 8, this additional set of contexts can be used to define a 3-dimensional ratings mapping function h_R:

$$h_R : U \times I \times C \rightarrow \textit{implicit feedback rating}$$

This relationship is shown in Figure 13.4. In this case, a hypothetical example of a newspaper is shown, where all the articles have been categorized into specific topics. When users click on advertisements relating to a particular topic on that page, this information is

recorded. The result is a multidimensional contextual representation, as discussed in Chapter 8. The level of similarity of Figure 13.4 to Figures 8.1 of Chapter 8 and Figure 11.6 of Chapter 11 is particularly striking. The use of multidimensional methods for context-sensitive recommendations [7] is a powerful technique, and it has recurred several times in this book in different scenarios.

Many of the same techniques of Chapter 8 may be used for recommending advertisements by treating them as items. However, the use of such techniques may need to be further enhanced with the cost information, such as the amount by which a publisher is paid for having a successful click-through on an advertisement. In other words, cost-sensitive variants of contextual collaborative filtering algorithms may be used, in which items with higher payoffs are prioritized over others. This can be achieved by ranking the predictions in terms of the expected payoffs, rather than in terms of the expected probability of a click. Content-based methods are particularly popular [105, 142, 327], and they use content similarity to match the context of the Web page with that in the advertisements.

13.8.2.1 Importance of Multi-Armed Bandit Methods

Multi-armed bandit methods are particularly useful for computational advertising. It is noteworthy that multi-armed bandit methods are particularly useful in settings where (a) new items enter the system all the time, and (b) the payoffs of selecting a particularly strategy can be precisely computed. Computational advertising in a domain in which the items are *extremely* transient and therefore it is particularly important to use exploration and exploitation simultaneously. Each arm of a slot machine can be viewed as one of the advertisements. Therefore, slot machines will constantly be added to and removed from the system. Also, since various types of context are associated with advertisements, it is particularly useful to leverage contextual bandit methods, where the context of the advertisement (e.g., search engine query keywords or the Web page on which an advertisement is displayed) is used in order to make decisions on whether to serve the advertisement. Refer to section 13.3 for a discussion of multi-armed bandit methods. A discussion of contextual bandit algorithms is also found in [348].

In many cases, the setting of computational advertising does not neatly fit into the traditional multi-armed bandit framework. For example, a publisher might present more than one advertisement at a time on a page, and a user might click on more than one advertisement presented to them. To handle this variation, the *slate problem* is proposed [290] for multi-armed bandits. In this variation of multi-armed bandits, the gambler is allowed to play more than one slot machine in a single try before he or she becomes aware of the rewards associated with that attempt. The simultaneous plays correspond to the different advertisements that are placed on a given page. The reward associated with a particular attempt is equal to the sum of the rewards obtained from the individual slot machines. In the advertisement setting, it translates to the placement of different advertisements (slot machine arms) on a Web page. In an ordered variation of this problem, different payoffs are associated with different placements of the advertisements on the Web page. For example, a higher placement in the ranked list will have a higher expected payoff than a lower placement. Refer to [290] for details of a randomized algorithm for computing the optimal policy.

13.8.3 Reciprocal Recommender Systems

The problem of computational advertising is related to the problem of *reciprocal recommendations* [481]. The basic idea is that the task of recommendation changes when one needs to consider the utility of the recommendation to multiple stakeholders with asymmetric interests. An example of such a scenario is that of online dating [480, 482], although the basic approach can be used in the context of various scenarios such as employer-employee matching [253] and mentor-mentee [103, 621] matching. Even the link-prediction problem discussed in Chapter 10 can be viewed as a form of reciprocal recommender system. A particularly relevant variation of link prediction is that of *reciprocal relationship prediction* [254], in which one attempts to predict the likelihood of the occurrence of bidirectional "follower" links in a directed social-network setting. There are several key differences between traditional recommender systems and reciprocal recommender systems. These differences [480] impact the nature of the algorithms that can be used in these settings:

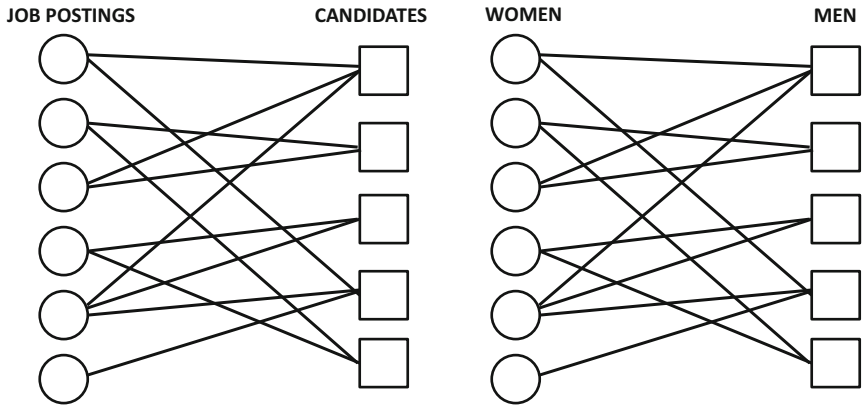
1. In traditional recommender systems, the user receives recommendations about items and is the sole decider of the use or purchase of the items. On the other hand, in a reciprocal recommender system such as online dating, the user is aware that the success of the transaction depends on the agreement of the other party. In fact, the other party is the "item" in the reciprocal setting. Therefore, in traditional recommender systems, items are abundant and there is no need for the agreement of any other party to consume the item. This is not true in reciprocal recommender systems.

2. In traditional recommender systems, users and products constantly recur in the system. As a result, it is much easier to collect data about user preferences. In reciprocal recommender systems (such as online dating) users and items might occur only once in the system and they might never recur after a successful transaction. Therefore, the cold-start problem is much more significant in the reciprocal setting. However, this problem is not universal to all reciprocal domains. For example, in the link prediction problem for social networks, the nodes are typically persistent.

The term "reciprocal" is motivated by the fact that both users and "items" have preferences and successful transactions can be initiated only by satisfying both. Furthermore, one can view the problem in a symmetric sense. In an employer-employee matching, one can view the (potential) employer as the user and the (potential) employee as the item, or one can view the employer as the item and the employee as the user. Therefore, there are two different recommendations occurring in parallel, which need to be harmonized in order to maximize the likelihood of successful transactions. For example, if an employee is very interested in a specific employer, but the employer is not interested in the skill-set of that employee, it makes little sense to introduce them to one another.

Explicit ratings are less common in such systems as compared to implicit feedback caused by user actions. Therefore, most of these systems are based on implicit feedback data in which user actions are used in lieu of the ratings. For example, in an online dating application, the initiation of a contact, exchange of a message, or response to a message may be given varying levels of weight as an implicit indication of interest. The main challenge in such systems is the cold-start problem because successful transactions have a tendency to remove users and items from the system.

In cases where the cold-start problem is significant, content-centric methods may play a key role either directly or indirectly. In direct methods, content-centric methods can be used within the recommendation technique in order to compensate for the paucity of ratings.



(a) Job matching (b) Online dating

Figure 13.5: Relating link prediction to reciprocal recommendations

Content-centric methods are also facilitated by the fact that both[2] users and items are likely to have descriptive profiles in such systems. A second (indirect) method to handle the non-persistent nature of users and items is to create persistent *representatives*. For example, consider a job-matching application. For each posted job in the system, one might treat other similar jobs posted in the past as instantiations of this job. This "similarity" is defined on the basis of content-centric attributes. Similarly, for each candidate in the system, one might treat other similar candidates in the past as instantiations of this candidate. In an online dating application, one might treat (expired) users with similar profiles as instantiations of a current profile. Successful transactions between past representatives can be treated as pseudo-transactions between their current avatars. The weights of such pseudo-transactions can be computed as a function of the similarities between the current avatars and past instantiations of the representative users and "items." This augmented data set can be used in conjunction with various collaborative filtering and link-prediction methods to predict the most likely links between pairs of nodes. It is often possible to recommend a user and item pair to one another even when a pseudo-transaction already exists between. Note that some of the pseudo-transactions might be quite noisy and unreliable. However, since the underlying inference methods use the aggregate structure of the data set, the predictions are likely to be reasonably robust. In cases where the pseudo-transactions are noisy, the corresponding user-item pairs are less likely to be recommended by prediction algorithms such as robust matrix factorization.

In the following, we will give a brief description of two key methods that are common to reciprocal recommender systems. However, since this is an emerging area, we recognize that these methods only scratch the surface of what is really possible in this area. Significant opportunities exist for further research in this domain.

13.8.3.1 Leveraging Hybrid Methods

In these methods, two *traditional* recommender methods are constructed corresponding to the preferences of the two reciprocal parties. Then, the predictions from these two parties are combined. For example, in a job-matching application, a traditional recommender system

[2]In traditional recommender systems, items are more likely to have descriptive profiles than users.

$\mathcal{R}_1$ may be used to create a ranked list of potential employees for an employer. Then, a traditional recommender system $\mathcal{R}_2$ may be used to create a ranked list of potential employers for an employee. The results from these two recommendations are combined to maximize the likelihood of a successful transaction. The combination method may use the weighted hybrid methods discussed in Chapter 6. As discussed in Chapter 6, the weights can be learned using linear regression methods, where the observed data is defined by successful transactions in the past. In cases where sufficient observed data is not available because of cold-start issues, either simple averages or domain-specific weights may be used. In cases where the preference if one party is more important than that of the other, a cascade hybrid can be used. For example, in settings where the number of job-seekers is far greater than the number of job-postings, the recommender system can choose to prioritize employer interests over employee interests. In such a setting, the cascade hybrid is ideal because it naturally prioritizes the first cascade in the hybrid over the second one.

There are many other factors that can play an important role in deciding how to combine the recommendations. For example, it is possible that one of the two parties might be naturally proactive (i.e., initiating contact), and the other party might be naturally reactive (i.e., responding to initial contact). In such cases, the nature of the hybrid can depend on the system's relative interests in satisfying the proactive and reactive parties. For example, one might assume primacy of interests of the proactive party but only ensure that the reactive party does not reject the recommendations. Repeated rejections from the reactive party can be costly and can affect the popularity of the system. Therefore, two models can created: the first model $\mathcal{R}_1$ computes the "items" that the proactive party will like, and the second model $\mathcal{R}_2$ computes the users that reactive party (i.e., "items") will dislike. The idea of the second model is to prune the recommended items from the first model that the reactive party will dislike. A variety of combination methods for these models are discussed in [482].

The recommender systems $\mathcal{R}_1$ and $\mathcal{R}_2$ are often content-centric systems because of the cold-start problem. However, in some case, the ratings data can be augmented by treating past users and items as instantiations of similar users in the system and constructing pseudo-transactions between users and items. In such cases, collaborative filtering methods can also be used because one can use the additional data resulting from the pseudo-transactions.

13.8.3.2 Leveraging Link Prediction Methods

In cases where the cold-start problem is not a serious issue or the ratings data can be augmented with the data from similar users and items, link prediction methods can be adapted to this setting. Matrix factorization methods for directed and undirected link prediction are discussed in section 10.4.5 of Chapter 10. In these cases, one can construct a bipartite network in which the two reciprocal parties form the two partitions of the network. For example, one partition might be a set of employers and another partition might be a set of employees. In the dating application, one partition might correspond to men and the other partition might correspond to women. The edges in this network correspond to (previous) successful transactions between the nodes in these partitions (or their similar representatives). These scenarios are illustrated in Figure 13.5(a) and (b), respectively. However, in other applications, the underlying graph might not be bipartite. For example, in a same-gender dating application, the underlying preference graph might not be bipartite. In some cases, when the preferences are specified in an asymmetric way, the underlying graph might be directed. In all these cases, the asymmetric and symmetric matrix factorization methods discussed in section 10.4.5 can be very useful. This is not particularly surprising, considering the fact that the link-prediction problem is a special case of reciprocal recommender

systems. In cases where the links are constructed in a noisy way using representatives, robust matrix factorization methods may be used to improve accuracy based on the ideas in Chapter 12.

13.9 Summary

This chapter reviews several advanced topics in recommender systems, such as group recommendations, multi-criteria recommendations, active learning, and privacy. In addition, some interesting applications of recommender systems have been covered.

Group recommendations are designed to provide recommendations to groups of users with possibly diverse interests. In general, straightforward averaging methods might not always work in these scenarios because of various social factors in the recommendation process. In multi-criteria recommender systems, diverse user interests are used to provide more robust recommendations. The basic idea is that the user behavior can be more accurately modeled when details for the user ratings of various criteria are available.

The problem of active learning, studies the issue of ratings acquisition in recommender systems. Ratings acquisition is sometimes expensive. Therefore, techniques need to be designed to judiciously query specific user-item combinations for ratings. The approach of active learning in recommender systems is very similar to that in classification.

Privacy remains a significant challenge for recommender systems, as in any other domain. Privacy-preserving methods can be applied either at data-collection time, or at data-publication time. Methods that preserve privacy at data-collection time generally provide better guarantees, but they are harder to implement from an infrastructure point of view.

Numerous applications have been proposed in recent years for recommender systems. Some examples include query recommendations, news personalization, computational advertising, and reciprocal recommendations. This chapter introduces some of the basic methods in these domains.

13.10 Bibliographic Notes

The problem of learning to rank is widely studied in the classification, internet search, and information retrieval [15, 115, 284, 370]. A tutorial on learning to rank from the perspective of recommender systems may be found in [323]. Ranking methods can be either pairwise methods or listwise methods [136]. Pairwise methods include the Bayesian personalized ranking model (BPR) [499], EigenRank model [367], pLPA [368], and CR [59]. Listwise methods include CoFiRank [624], CLiMF, xCLiMF and several other variations [545–548]. Some of these methods have also been generalized to the contextual scenario [549].

Multi-arm bandit methods can be viewed as a class of reinforcement learning algorithms [579]. A simple discussion of several bandit algorithms may be found in [628], although the book is written in the context of Website optimization. Bandit algorithms for recommender systems are discussed in [92, 348]. The work in [349] introduces the problem of evaluating bandit algorithms in the offline setting. The use of multi-armed bandits for computational advertising is discussed in [160, 290].

Group recommender systems are discussed in detail in [271, 272, 407, 408]. A review of social factors for group recommender systems may be found in [489]. Case-based methods for group recommendations are discussed in [413, 415]. Group recommendations have been used in a variety of domains, such as movies [168], television [653], music [412], and travel [52, 272, 413]. The limitations of the averaging strategy for group recommender systems are discussed

in [409, 654]. A variety of aggregation strategies for group recommender systems, such as plurality voting, multiplicative aggregation, Borda count, Copeland rule, approval voting, and fairness are suggested in [407]. An experimental study comparing the various strategies is also included in the same work. In some cases, one is interested in recommending complex items containing sequences of items. An example is the case of a television program for a set of viewers, where the overall program may contain several components of various types. In such cases, the ordering of items is also important. Such systems are discussed in [407].

Surveys on multi-criteria recommender systems may be found in [11, 398, 604]. The multi-criteria recommendation problem was first defined in the seminal work of [12]. Neighborhood-based methods for multi-criteria recommendations are discussed in [12, 399, 596]. The work in [399] proposes three different methods to perform the aggregated similarity computation in neighborhood methods. However, the overall approach is not different in principle from that discussed in [12]. An ensemble-based method was also proposed in [12]. A number of model-based methods have also been proposed in the context of multi-criteria recommender systems. These include the flexible mixture model [514] and a multi-linear singular value decomposition (*MSVD*) approach [353]. Methods have also been proposed for cases in which overall ratings are not available. For example, the work in [328] proposes a method to combine the predicted ratings across various criteria with the use of a UTilities Additive method (UTA). The work in [276] uses a support vector regression model to determine the relative importance of different criteria. These are used to combine the user-based and item-based regression models with a weighted approach. A pareto-optimal approach with the use of skyline queries on a restaurant rating system is proposed in [340].

A detailed review of active learning methods is provided in [513]. However, this review is mostly based on the classification problem, since the available work on recommender systems is limited. Only a limited amount of work [192–194, 257, 295, 330, 578] has been proposed in recent years on this topic. The area of active learning is still quite open as far as the recommendation problem is concerned. An interesting class of algorithms, related to temporal collaborative filtering, is the multi-arm bandit class of algorithms in which the recommender trades off exploration vs exploitation in the recommendation space [92, 348].

Privacy-preserving techniques may include the use of perturbation techniques [35, 38, 484, 485], group-based anonymization methods [27, 352, 386, 521], or distributed methods [75, 133, 334, 551, 606]. Both perturbation methods and distributed techniques have a common aspect that they tend to preserve the privacy at data-collection time. This provides a greater level of privacy. On the other hand, these systems are generally harder to implement because of the greater infrastructural and customization issues involved in the final use of the stored data. These issues surface because the stored data is in a form that cannot be used by a traditional collaborative filtering algorithm. Group-based anonymization techniques are designed to publish the data, which is collected by a centralized entity. These techniques are more popular and the output can be used in conjunction with traditional collaborative filtering algorithms. All these methods are affected by the curse of dimensionality [30], which prevents effective privacy preservation for high-dimensional data. Some methods for anonymization of high-dimensional and sparse data sets are proposed in [657]. Recently, the notion of differential privacy has been proposed [189] that is theoretically very popular, although its practical and commercial use remains limited. A differentially private matrix factorization has recently been proposed in [372]. A privacy-preserving approach that treats the collecting system as a distrusted entity is proposed in [642].

Recommender systems have many specialized applications in the Web domain. Query recommendation methods attempt to recommend similar queries to those already issued by the user in a particular session. The work in [57] returns the most similar queries to the

queries *current query*, which also have sufficient popularity (support). Support is measured in terms of the number of times the query was issued by other users, and the corresponding results were found relevant. The work in [137] uses not just the current query, but the current session of queries as the context for the query suggestion. An interesting idea in this area is that of *query flow graphs* [90], which uses a graphical representation of the user's latent querying behavior to make recommendations. The work in [429] performs query recommendations with the use of random walks on the query-URL graph. The use of Markov models for query recommendations is discussed in [244].

The dynamic profiler system is discussed in [636]. Methods for Web portal personalization are discussed in [34]. The use of semantic contextualization for news recommendation was discussed in [134]. This work is based on contextual recommendation ideas presented in Chapter 8. The Google news personalization engine is described in more detail in [175]. Mobile recommender systems are discussed in [504].

One of the earliest systems for computational advertising was discussed in [28]. However, this system was not based on modern models of computational advertising. More recent discussions of such systems may be found in [106, 107]. The slate method for computational advertising is discussed in [290]. In some cases, linear payoffs are associated with the features of Web pages and advertisements. A variant of the LinUCB algorithm is proposed in [160] to handle this setting. The problem of computational advertising is related to the problem of *reciprocal recommendations* [481]. The basic idea is that the task of recommendation changes when one needs to consider the utility of the recommendation to multiple stakeholders with asymmetric interests. Examples of such applications include online dating [480, 482], job matching [253], and mentor-mentee recommendations [103, 621].