PART I

Introduction

1
Introduction

Recommender systems (or recommendation systems) are computer programs that recommend the "best" items to users in different contexts. The notion of a best match is typically obtained by optimizing for objectives like total clicks, total revenue, and total sales. Such systems are ubiquitous on the web and form an integral part of our daily lives. Examples include product recommendations to users on an e-commerce site to maximize sales; content recommendations to users visiting a news site to maximize total clicks; movie recommendations to maximize user engagement and increase subscriptions; or job recommendations on a professional network site to maximize job applications. Input to these algorithms typically consists of information about users, items, contexts, and feedback that is obtained when users interact with items.

Figure 1.1 shows an example of a typical web application that is powered by a recommender system. A user uses a web browser to visit a web page. The browser then submits an HTTP request to the web server that hosts the page. To serve recommendations on the page (e.g., popular news stories on a news portal page), the web server makes a call to a recommendation service that retrieves a set of items and renders them on the web page. Such a service typically performs a large number of different types of computations to select the best items. These computations are often a hybrid of both offline and real-time computations, but they must adhere to strict efficiency requirements to ensure quick page load time (typically hundreds of milliseconds). Once the page loads, the user may interact with items through actions like clicks, likes, or shares. Data obtained through such interactions provide a feedback loop to update the parameters of the underlying recommendation algorithm and to improve the performance of the algorithm for future user visits. The frequency of such parameter updates depends on the application. For instance, if items are time sensitive or ephemeral, as in the case of news recommendations, parameter updates must be done frequently (e.g., every few minutes). For

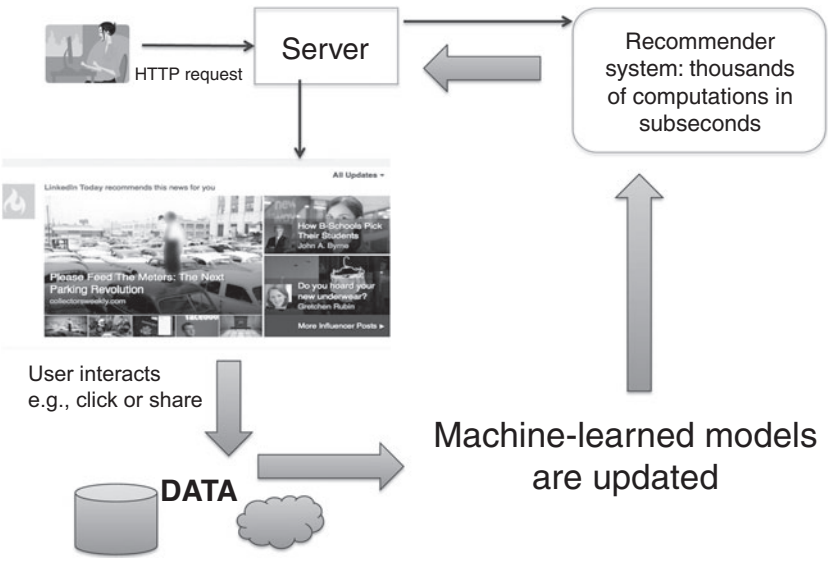

Figure 1.1. A typical recommender system.

other applications where items have a relatively longer lifetime (e.g., movie recommendations), parameter updates can happen less frequently (e.g., daily) without significant degradation in overall performance.

Algorithms that facilitate selection of best items are crucial to the success of recommender systems. This book provides a comprehensive description of statistical and machine learning methods on which we believe such algorithms should be based. For the sake of simplicity, we loosely refer to these algorithms as recommender systems throughout this book, but note that they only represent one component (albeit a crucial one) of the end-to-end process required to serve items to users in a scalable fashion.

1.1 Overview of Recommender Systems for Web Applications

Before developing a recommender system, it is important to consider the following questions.

- *What input signals are available?* When building machine-learned models of what items a user is likely to interact with in a given context, we can draw on many signals, including the content and source of each item; a

user's interest profile (reflecting both long-term interests based on prior visits and short-term interests as reflected in the current session); a user's declared information, such as demographics; and "popularity" indicators such as observed *click-through rates* or CTRs (the fraction of time in which the item is clicked on when a link to it is presented to users) and extent of social sharing (e.g., the number of times the item is tweeted, shared, or liked).

- *What objective(s) to optimize for?* There are many objectives a website could choose to optimize for, including near-term objectives, such as clicks, revenue, or positive explicit ratings by users, and long-term metrics, such as increased time spent on the site, higher return and user retention rates, increase in social actions, or increase in subscriptions.

Different recommendation algorithms need to be developed based on the answers to these questions.

1.1.1 Algorithmic Techniques

In general, a recommender system needs algorithmic techniques to address the following four tasks:

- *Content filtering and understanding*. We need to have sound techniques to filter out low-quality content from the *item pool* (i.e., the set of candidate items). Recommending low-quality content hurts user experience and the brand image of the website. The definition of low quality depends on the application. For a news recommendation problem, salacious content could be considered low quality by reputed publishers. An e-commerce site may not sell items from certain sellers with a low reputation rating. Defining and flagging low-quality content is typically a complex process that is addressed through a combination of methods, such as editorial labeling, crowdsourcing, and machine learning methods like classification. In addition to filtering low-quality content, it is important to analyze and understand the content of items that pass the quality bar. Creating item profiles (e.g., feature vectors) that capture the content with high fidelity is an effective approach. Features can be constructed using a variety of approaches, such as bag-of-words, phrase extraction, entity extraction, and topic extraction.
- *User profile modeling*. We also need to create user profiles that reflect the items that the users are likely to consume. These profiles could be based on demographics, user identity information submitted at the time of registration, social network information, or behavioral information about users.

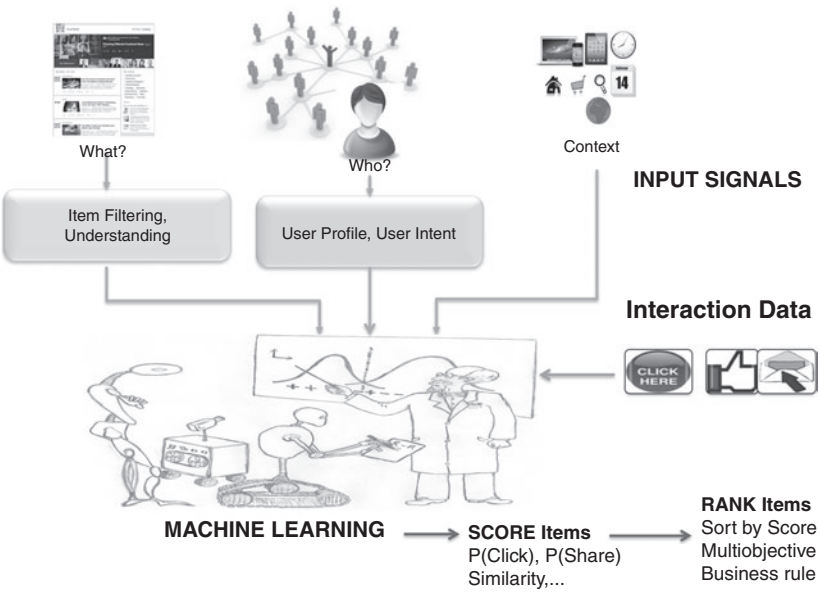

Figure 1.2. Overview of recommender system.

- *Scoring*. On the basis of user and item profiles, a scoring function needs to be designed to estimate the likely future "value" (e.g., CTRs, semantic relevance to the user's current goal, or expected revenue) of showing an item to a user in a given context (e.g., the page the user is viewing, the device being used, and the current location).
- *Ranking*. Finally, we need a mechanism to select a ranked list of items to recommend so as to maximize the expected value of the chosen objective function. In the simplest scenario, ranking may consist of sorting items based on a single score, such as the CTR of each item. However, in practice, ranking is more involved and is a blend of different considerations, such as semantic relevance, scores quantifying various utility measures, or diversity and business rules to ensure good user experience and preserve the brand image.

Figure 1.2 illustrates how the previously described algorithmic components are related. Input signals based on user information, item information, and historical user-item interaction data are used by machine-learned statistical models to produce scores that quantify users' affinity to items. The scores are combined by the ranking module to produce a sorted list of items based on descending order of priority obtained by considering single or multiple objectives.

Content filtering and understanding techniques depend to a large extent on the types of items to be recommended. For example, techniques for processing text are quite different from those used to process images. We do not intend to cover all such techniques, but we provide a brief review in Chapter 2. We also do not intend to cover a large variety of techniques for generating user profiles. However, we describe techniques that automatically "learn" both user and item profiles from historical user-item interaction data and that can also incorporate any existing profile information produced by some other existing techniques in a seamless fashion.

1.1.2 Metrics to Optimize

Among the considerations important to determining appropriate solutions for web recommendation problems, the first and foremost is to ascertain the metric(s) we want to optimize. In many applications, there is a single metric to optimize, for example, maximizing the total clicks or total revenue or total sales in a given time period. However, some applications may require simultaneous optimization of multiple metrics, for example, maximizing total clicks on content links subject to constraints on downstream engagement. An example constraint could be to ensure that the number of *bounce clicks* (clicks that do not materialize into a read) is less than some threshold. We may also want to balance other considerations, such as diversity (ensuring a user sees a range of topics over time) and serendipity (ensuring that we do not overfit our recommendations to the user, thereby limiting the discovery of new interests) to optimize long-term user experience.

Given the definition of metrics to optimize, the second consideration is to define scores that serve as the input to the optimization problem. For instance, if the goal is to maximize the total clicks, CTR is a good measure of the value of an item to a user. In the case of multiple objectives, one may have to use multiple scores, such as CTR and expected time spent. Statistical methods that can estimate the scores in a reliable fashion have to be developed. This is a nontrivial task that requires careful consideration. Once score estimates are available, they are combined in the ranking module based on the optimization problem under consideration.

1.1.3 The Explore-Exploit Trade-off

Reliably estimating scores is a fundamental statistical challenge in recommender systems. This often involves estimating expected rates of some positive response, such as click rate, explicit rating, share rate (probability of sharing an

item), or like rate (probability of clicking the "like" button associated with an item). The expected response rates can be weighted according to the utility (or value) of each possible response. This provides a principled approach for ranking items based on expected utility. Response rates (appropriately weighted) are the primary scoring functions we consider in this book.

To accurately estimate response rates for each candidate item, we could *explore* each item by displaying it to some number of user visits to collect response data on all items in a timely manner. Then, we can *exploit* the items with high response rate estimates to optimize our objectives. However, exploration has an opportunity cost of not showing items that are empirically better (based on the data collected so far); balancing these two aspects constitutes the explore-exploit trade-off.

Explore-exploit is one of the main themes of this book. We provide an introduction in Chapter 3 and discuss the technical details in Chapter 6. The methods described in Chapters 7 and 8 are also developed to address this issue.

1.1.4 Evaluation of Recommender Systems

To understand whether a recommender system achieves its objectives, it is important to evaluate its performance at different stages during the development cycle. From the perspective of evaluation, we divide the development of a recommendation algorithm into two phases:

- *Predeployment phase* includes steps before the algorithm is deployed online to serve some fraction of user visits to the website. During this phase, we use past data to evaluate the performance of the algorithm. The evaluation is limited because it is *offline*; users' responses to items recommended by the algorithm are not available.
- *Postdeployment phase* starts when we deploy the algorithm *online* to serve users. It consists primarily of online bucket tests (also called A/B experiments) to measure appropriate metrics. Although this is far more close to reality, there is a cost to running such tests. A typical approach is to filter out algorithms with poor performance based on offline evaluation in the predeployment stage.

Different evaluation methods are used to evaluate various components of a recommender system:

- *Evaluation of scoring*. Scoring is usually done through statistical methods that predict how a user would respond to an item. Prediction accuracy is often used to measure the performance of such statistical methods. For example,

if a statistical method is used to predict the numeric rating that a user would give to an item, we can use the absolute difference between the predicted rating and the true rating, averaged across users, to measure the error of the statistical method. The inverse of error is accuracy. Other ways of measuring accuracy are described in Section 4.1.2.

- *Evaluation of ranking*. The goal of ranking is to optimize for the objectives of a recommender system. In the postdeployment stage, we can evaluate a recommendation algorithm by directly computing the metrics of interest (e.g., CTR and time spent on recommended items) using data collected from online experiments. In Section 4.2, we discuss how to set up the experiments and analyze the results properly. However, in the predevelopment stage, we do not have data from users served by the algorithm – estimating the performance of the algorithm offline to mimic its online behavior is challenging. In Sections 4.3 and 4.4, we describe a couple of approaches to addressing this challenge.

1.1.5 Recommendation and Search: Push versus Pull

To set the scope of this book, we note that user intent is an important factor that differentiates various web applications. If the intent of a user is explicit and strong (e.g., query in web search), the problem of finding or "recommending" items that match the user's intent can be solved through a *pull* model – by retrieving items that are relevant to the explicit information needs of the user. However, in many recommendation scenarios, such explicit intent information is not available; at best, it can be inferred to some extent. In such cases, it is typical to follow the *push* model, where the system pushes information to the user – the goal is to serve items that are likely to engage the user.

Actual recommendation problems encountered in practice fall somewhere in the continuum of pull versus push. For instance, recommending news articles on a web portal is predominantly through a push model because explicit user intent is generally unavailable. Once the user starts reading an article, the system can recommend news stories related to the topic of the article that the user is reading, which provides some explicit intent information. Such a related news recommender system is usually based on a mix of pull and push models; we retrieve articles that are topically related to the article that the user is currently reading and then rank them to maximize user engagement.

We do not focus much on applications, such as web search, that require mostly a pull model and rely heavily on methods that estimate semantic similarity between a query and an item. Our focus is more on applications where

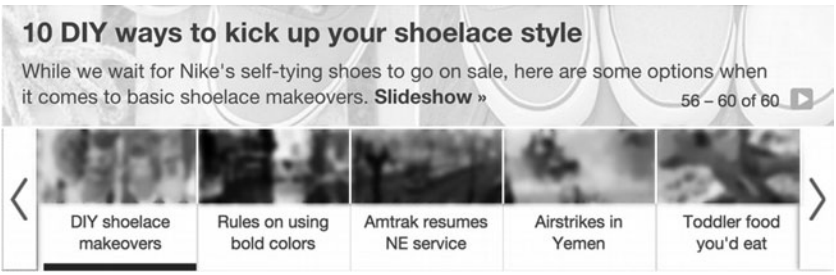

Figure 1.3. Today Module on the Yahoo! front page.

user intent is relatively weak and it is important to score items for each user based on response rates estimated through previous user-item interactions.

1.2 A Simple Scoring Model: Most-Popular Recommendation

To illustrate the basic idea of scoring, we consider the problem of recommending the most popular item (i.e., the item with the highest CTR) on a single slot of a web page to all users to maximize the total number of clicks. Although simple, this problem of *most-popular recommendation* includes the basic ingredients of item recommendation and also provides a strong baseline for the more sophisticated techniques that we describe in later chapters. We assume the number of items in the item pool is small relative to the number of visits and clicks. We do not make any assumption on the composition of the item pool; new items may get introduced and old ones may disappear over time.

Our example application is recommending new stories on the Today Module of Yahoo! front page (Figure 1.3 shows a snapshot). This application is used throughout the book for the purposes of illustration. The module is a panel with several slots, where each slot displays an item (i.e., story) selected from an item pool consisting of several items that are created through editorial oversight. For simplicity and ease of exposition, we focus on maximizing clicks on the single most prominent slot of the module, which gets a large fraction of the clicks.

Let p_{it} denote the instantaneous CTR of item i at time t. If we knew p_{it} for each candidate item i, we could simply serve the item with the highest instantaneous CTR to all user visits that occur at the given time point t. In other words, we select item $i_t^* = \arg\max_i p_{it}$ for visits at time t. However, instantaneous CTRs are not known; they have to be estimated from data. Let $\hat{p}_{it}$ denote the estimated CTR from data. Is it enough to serve the item with the highest estimated