Recommender Systems in Industry: A Netflix Case study

Xavier Amatriain and Justin Basilico

Abstract The Netflix Prize put a spotlight on the importance and use of recommender systems in real-world applications. Many the competition provided many lessons about how to approach recommendation and many more have been learned since the Grand Prize was awarded in 2009. The evolution of industrial applications of recommender systems has been driven by the availability of different kinds of user data and the level of interest for the area within the research community. The goal of this chapter is to give an up-to-date overview of recommender systems techniques used in an industrial setting. We will give a high-level description the practical use of recommendation and personalization techniques. We will highlight some of the main lessons learned from the Netflix Prize. We will then use Netflix personalization as a case study to describe several approaches and techniques used in a real-world recommendation system. Finally, we will pinpoint what we see as some promising current research avenues and unsolved problems that deserve attention in this domain from an industry perspective.

1 Introduction

Recommender Systems are a prime example of the mainstream industry use of large-scale machine learning and data mining. Diverse applications in areas such as e-commerce, search, Internet music and video, gaming, and even online dating apply similar techniques that leverage large volumes of data to better fulfill a user's needs in a personalized fashion. These techniques have such wide applicability because they have been demonstrated to be effective in increasing core business metrics such as customer satisfaction and revenue. In this chapter, we will focus on approaches for applying recommendation algorithms with a focus on the problem formulations, algorithms, and metrics. Of course, other aspects such as user interaction design can have a deep impact on the effectiveness of an approach. Those topics are covered in other chapters in the book but are outside the scope of this one.

Xavier Amatriain
Netflix, 100 Winchester Cr., Los Gatos, CA 95032, USA. e-mail: xavier@netflix.com

Justin Basilico
Netflix, 100 Winchester Cr., Los Gatos, CA 95032, USA. e-mail: jbasilico@netflix.com

Given an existing application, an improvement in the recommendation system can have a value of millions of dollars and can be the factor that determines the success or failure of a business. In section 2 we will review some of the typical uses of recommendation systems in industry. While a lot of work in recommendation focuses on algorithms, there are other aspects of a recommendation approach that can have a significant impact. For example, adding new data sources or representations (features) to an existing algorithm. In section 4, we will use Netflix as the driving example to describe the use of data, models, and other personalization techniques.

Another important factor we consider is how to measure the success of a given personalization technique. Root mean squared error (RMSE) was the offline evaluation metric chosen for the Netflix Prize (see section 3). But there are many other relevant metrics that, if optimized, would lead to different solutions. For example, ranking metrics such as Normalized Discounted Cumulative Gain (NDCG), recall, or area under the ROC curve (AUC), often used in Information Retrieval, can also be used to evaluate recommendations. However, beyond the optimization of a given offline metric, what we are really pursuing is the impact of a technique on the business. To do this, we need to relate the quality of a recommendation to more customer-facing metrics such as click-through rate (CTR) or retention. We will describe an approach of how to make use of offline and online metrics to drive innovation, called "Consumer Data Science," in section 6.

A key aspect in building a successful large-scale recommendation system is to choose an appropriate architecture that is capable of running computationally complex algorithms and also produces fresh results with an acceptable latency. In section 7, we will describe a three-layer architecture that addresses these concerns. But before we dive into these details, it will be useful to understand the uses of recommender systems across industry. In the next section, we will briefly describe some of the most typical use cases.

2 Recommender Systems in Industry

Recommendation systems are used by many Internet-focused companies in a variety of application domains. Each domain has its own unique recommendation challenges. We provide here a short overview of some of the more well-known applications in industry.

Today most e-commerce sites and applications are likely to have some sort of recommendation engine powering their user experience. The first large company to be credited as having included a recommender system at the core of their experience is Amazon. Amazon initially employed a simple item-item collaborative filtering approach [48]. The current Amazon experience includes many different instances of recommendation at different levels: from listings on the homepage to many product pages having lists of other products bought or viewed. Other retail companies such as eBay have followed the lead and incorporated recommendations in their experience, such as post-purchase recommendations [88].

News is also an area that companies have applied recommendation approaches to personalize and focus on the interests of a user. For example, Google News was powered by some kind of recommendations for news articles from the beginning [49] [19]. Yahoo! has also invested in personalizing news and other web content [47] [1]. For news recommendation, some of the key challenges are freshness, where relevant articles may have a very limited time span, and diversity, where there can be a large number of articles about the same topic. However, news has the advantage of textual content, which allows for techniques from natural language processing to be applied to create features that can be used in recommendation, which is especially helpful when user behavioral data is sparse.

Video recommendation has always been an active area of research, so it is not surprising that it is used in industry to recommend a variety of types of video spanning movies, TV shows, and user-generated content. For instance, recommendations have been an important component of the YouTube experience to help navigate the vast amounts of user-generated video [20]. With video, it can be hard to extract good content information without harnessing sophisticated computer vision approaches. While metadata may be available for professionally-created content, harnessing user behavior has been key to building recommendations for videos. In such domains, transitioning from ratings to learning-to-rank has shown to be important. Google, for instance, uses a new family of loss functions and shows its applicability in YouTube and Google Music [90].

Music recommendation is also an active application area where there have been interesting developments in the past years. Pandora, for instance, created a complete business model around the idea of creating personalized music stations. They created an approach that combined traditional collaborative filtering techniques with a curated approach called the Music Genome Project [72]. Similarly, Apple's iTunes application uses information about a user's music library to drive personalized mixes and playlists. More recently, Spotify started getting in the business of providing personalized music recommendations in its service. Their approach to recommendations is currently mostly based on standard matrix factorization techniques [9]. Finally, EchoNest was a well-known startup that provided music recommendation engines powering many different services before being acquired by Spotify. They combined different approaches including collaborative filtering, metadata, and audio signal analysis of the music [46]. Music recommendation has some unique aspects [16], such as the multi-level nature of artists, albums, tracks, and playlists that recommendations can be done across. Music tracks also typically short and often listened to repeatedly, which can lead to interesting approaches for leveraging this data and behavior.

More recently, social network companies have introduced a number of different recommendation avenues. Twitter, for example, introduced their Who to Follow recommendation algorithm to recommend new social connections [29]. LinkedIn used a Survival Analysis approach to understand how likely a user is to change jobs [89]. Google has also published work in recommendation systems for some of their social networks such as Orkut [18]. Yahoo! has also worked on personalizing aspects such as comments on social sites [2] or tags in image collections for Flickr [77].

Recommendation algorithms are also very important in online dating sites. Those recommender systems have some particular requirements. For example, the success of the system is not determined by one user receiving a good recommendation but rather by both parties accepting it [61].

In addition to focusing on a single domain, some companies are applying recommendation approaches across domains and even potentially using data from one domain to recommend in another. Microsoft developed a distributed Bayesian approach to recommendation systems called Matchbox [81]. This solution was deployed in several different contexts. For example, it is the main building block of the content recommendations for the XBox game console [39], including games, apps, video, and music content.

Beyond companies building recommendation systems into their own products, there are also many smaller companies that have focused their activity around developing general recommendation systems or technologies. Commendo [32] and Gravity R&D [83], for example, are recommender system consulting firms that emerged from some of the teams that competed in the Netflix Prize.

While many of the previous examples are interesting from an algorithmic perspective, they also represent the different or complimentary requirements that recommender systems have from an industrial point of view. For example, some important issues that are mentioned in most of these publications, unlike in more academic settings, are scalability, business metrics, and integration of the system in the overall user experience. We will go into these issues in the remainder of this chapter.

3 The Netflix Prize

In 2006, Netflix announced the Netflix Prize, a machine learning and data mining competition to predict movie ratings on a 5-star scale. We conducted this competition to find new ways to improve the recommendations we provide to our members, which is a key part of our business. However, we had to come up with a proxy question that was easier to evaluate and quantify: the root mean squared error (RMSE) of the predicted rating. We offered a $1 million prize to whomever came up with a solution that reduced RMSE by 10% beyond what was obtained by Cinematch, our existing system.

The Netflix Prize put the spotlight on the Recommender Systems area and the value of generating personalized recommendations from user data. It did so by providing a crisp problem definition that enabled thousands of teams to focus on improving a single metric. While this was a simplification of the recommendation problem, many valuable lessons were learned.

3.1 Lessons from the Prize

After the first year of competition, the KorBell team won the first Progress Prize with an 8.43% improvement. They reported more than 2000 hours of work in order to come up with the final combination of 107 algorithms that put them at the top of the leaderboard and resulted in this prize. As per the terms of the competition, they shared their resulting solution with the team at Netflix. We looked at the two underlying algorithms with the best performance in the ensemble: Matrix Factorization (MF) [44] [1] (see section 5.3) and Restricted Boltzmann Machines (RBM) [71]. Matrix Factorization by itself provided a 0.8914 RMSE, while RBM alone provided a competitive but slightly worse 0.8990 RMSE. A linear blend of these two reduced the error to 0.88.

Given that a combination of these two algorithms performed well on the competition dataset, we sought to put them to use. To do this, we had to overcome some limitations. For instance, the competition code the authors provided was built to handle 100 million ratings, however we needed to apply it to the more than 5 billion that we have. Also, the code was designed to run on a static dataset and thus not built to adapt as members added more ratings. Once we overcame those challenges, we deployed the two algorithms into production, where they are still used to predict our members ratings for videos.

One of the most interesting findings during the Netflix Prize came out of a blog post. Simon Funk introduced an incremental, iterative, and approximate way to compute a matrix factorization (referred to as SVD) using gradient descent [27]. This provided a practical way to scale matrix factorization methods to large datasets. Another enhancement to matrix factorization methods was Koren *et. al*'s SVD++ [42]. This asymmetric variation enables adding both implicit and explicit feedback, and removes the need for learning user-specific parameters for the implicit part.

The second model that proved successful in the Netflix Prize was the Restricted Boltzmann Machine (RBM). RBMs can be understood as the fourth generation of Artificial Neural Networks: the first being the Perceptron popularized in the 60s; the second being the backpropagation algorithm in the 80s; and the third being Belief Networks (BNs) from the 90s. RBMs are BNs that restrict the connectivity to make learning easier. RBMs can be stacked to form Deep Belief Networks (DBN), which is a form of deep learning. For the Netflix Prize, Salakhutditnov *et al.* proposed an RBM structure with binary hidden units and softmax visible units with 5 biases that are initialized with the movies that the user rated [71].

Many other learnings came out of the competition. For example, the matrix factorization methods mentioned above were combined with traditional neighborhood-based approaches [42]. Also, early in the competition, it became clear that it was important to take into account temporal dynamics of user feedback [43]. Another finding of the Netflix Prize was that there is a large amount of noise in the ratings

[1] The application of Matrix Factorization to the task of rating prediction closely resembles the technique known as Singular Value Decomposition used, for example, to identify latent factors in Information Retrieval. Therefore, it is common to see people referring to this MF solution as SVD.

provided by users. This was already known in the literature; Herlocker *et al.* [30] coined the term "magic barrier" to refer to the limit of accuracy in a recommender system due to the natural variability in ratings. This limit was relatively close to the actual Prize threshold [5], and might be a factor in the substantial effort needed to reduce RMSE by enough to cross the 10% line.

After almost three years, the final Grand Prize ensemble that won the $1M prize was a truly impressive compilation and culmination of work, blending hundreds of predictive models to finally cross the finish line [8]. The final solution was accomplished by combining many independent models developed by different teams that joined forces. It highlights the power of using ensembles to combine a heterogeneous set of models to achieve maximum accuracy.

At Netflix, we evaluated some of the new methods included in the final solution. The additional accuracy gains that we measured did not seem to justify the engineering effort needed to bring them into a production environment. In addition, our focus on improving Netflix personalization had expanded beyond rating prediction. In the next section, we will explain the different methods and components that produce a complete personalization approach such as the one used by Netflix.

4 Recommendation Beyond Rating Prediction

Netflix has discovered through the years that there is significant business value in incorporating recommendations to personalize as much of the user experience as possible. This realization motivated the Netflix Prize described in the previous section and has subsequently driven the effort to personalize the service in many other ways. In section 2 we introduced different industrial scenarios for recommender systems. In the following sections we will use Netflix as an example of a fully personalized industrial recommendation system.

Before we go into the details, first let us provide some context about the Netflix service as it relates to personalization. Netflix was originally known as a DVD-by-mail subscription service in the US, which it was at the time the Netflix Prize started. However, it has since grown into an international Internet video streaming subscription service. It allows members to instantly stream movies and TV shows from a catalog to a multitude of devices such as laptops, smart TVs, game consoles, tablets, and mobile phones. One of the key aspects of the Netflix service is that it allows members to watch any video we have available in our catalog at any time on any device. Since we have a large quantity of available videos that a member can watch, a key concern is how to help members find videos in our catalog that they want to watch and will enjoy enough to come back. This is the task where we rely primarily on our recommendation system to help. The videos we have are licensed from content providers or produced ourselves. The cost for serving each video is approximately the same, so we have no incentive for favoring one video over another when doing recommendation. Thus, we take a member-centric approach to recommendation. This is in contrast to other domains such as e-commerce, online

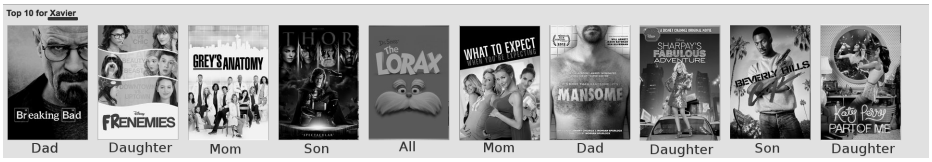

Fig. 1: Example of a Netflix Top 10 row. We promote personalization awareness and reflect on the diversity of a household. Note that the labels are an illustration, since the system does not explicitly know the true household composition.

advertising, or search where there can be very different amounts of revenue for different items.

4.1 Everything is a Recommendation

Netflix personalization starts on a member's homepage, which the application displays on any device after login. This page consists of groups of videos arranged in horizontal rows that create a two-dimensional grid of videos. Each row has a title that conveys the intended meaningful connection between the group of videos in that row. Most of our personalization is embodied in the way we generate rows, select rows, determine what videos to include in a row, determine the ordering of videos in a row, and determine the ordering rows on a page.

Take as a first example the Top 10 row (see Figure 1). This row is our best guess of the videos a user is most likely to watch and enjoy. Of course, when we say "a user', we really mean everyone in a household using that membership (or profile). It is important to keep in mind that Netflix personalization is intended to handle a household that is likely to have different people with different tastes. That is why with a Top 10 row, someone in a family that watches Netflix is likely to discover items for dad, mom, the children, or the family as a whole. Even for a single person household, we want the recommendations to appeal to a person's range of interests and moods. While there are specific techniques for group recommendations, such as the ones described in chapter **??**, those techniques usually rely on having captured each individual preferences rather than an aggregate. For this and other reasons, in most parts of our system we cater to different people and moods by not only optimizing for *accuracy*, but also for *diversity* [69, 87].

Another important element of Netflix personalization is *awareness*. We want members to be aware of how we are adapting to their tastes. This not only promotes trust in the system, but also encourages members to give feedback, which will result in better recommendations. A way of promoting trust with the personalization is to provide *explanations* for why we decide to recommend a given movie or show (see Figure 2). A video is not recommended because it suits a business needs, but because it matches the information we have from a user: viewing history, explicit

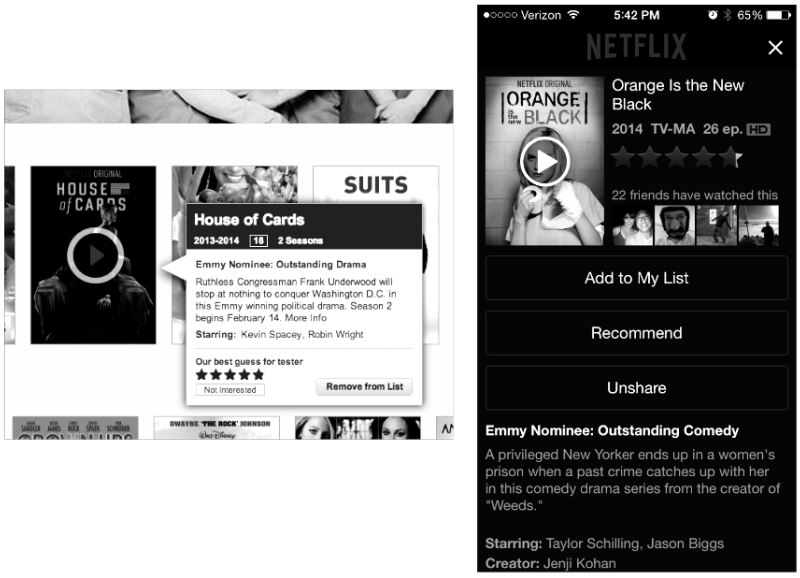

Fig. 2: Adding explanation and evidence for recommendations contributes to user satisfaction and requires specific algorithms. Evidence can include your predicted rating, related shows you have watched, or even friends who have interacted with the video.

feedback of ratings and taste preferences, or even a friends recommendations. See chapter **??** for more details on how to design good explanations for a recommender system.

On the topic of friends, we have a social feature that allows members to connect through Facebook to find friends who are also Netflix members. Knowing about someone's friends not only gives us another signal to use in our personalization algorithms, but also allows us to create rows based on their social circle to generate recommendations. See chapter **??** for some examples on how to use social network information to generate recommendations.

Similarity is also an important aspect of personalization. We think of similarity in a very broad sense; it can be between videos or between members, and can be along multiple dimensions such as metadata, ratings, or viewing data. While similarity itself can be used as the basis for a recommendation system, we tend to use various forms of similarity as features in other models or as navigational constructs for the user. For example, we generate rows of "adhoc genres" based on their similarity to videos that a member has interacted with recently, which we label "Because you watched". Also, we provide a list of similar videos that a user may be interested in on the information page for a video. Of course, there can be many different notions of

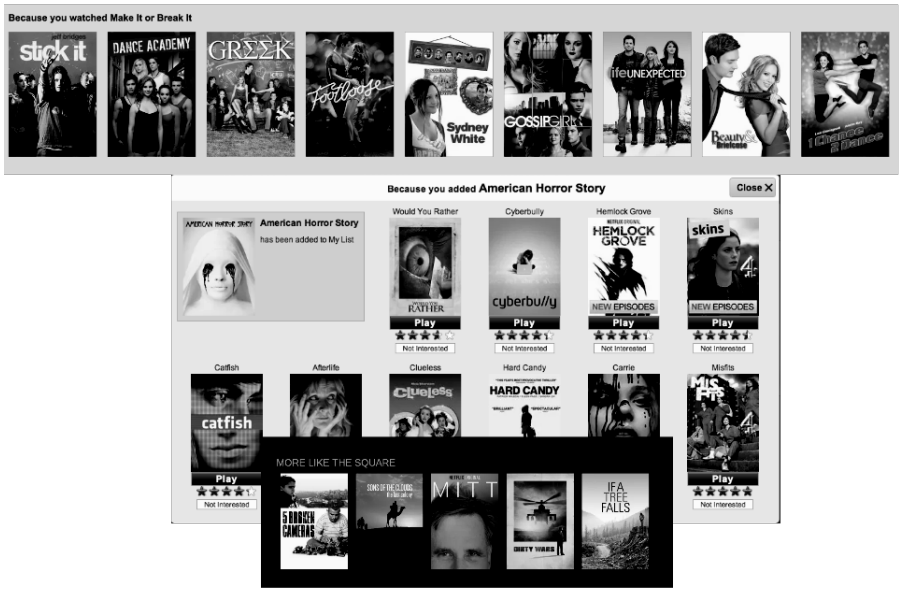

Fig. 3: Similarity can be used to present recommendations in different contexts and in response to certain user actions.

similarity between two items, each of which can be used for the basis of generating a list of similars. These can be combined by constructing independent similarity models and training an ensemble. Other more sophisticated graphical methods such as SimRank [34] accomplish a similar goal. On the other hand, similarity itself can be personalized using approaches such as Personalized Page Rank [26]

In many services like Netflix, even a situation where a user enters an explicit search query can be turned into a recommendation. An example of this might be the user entering a generic term (e.g. "summer" or "Italian") or the title of an item that is not available in the catalog. In these situations we need to come up with good related recommendations. Even the auto-complete suggestions when the user starts typing can be personalized and interpreted recommendation over a constrained set. LinkedIn's Metaphor system is another good example of a complete search recommendation system [64].

In most of the previous contexts, the goal of the recommender system is to present a number of attractive items for a person to choose from. This is usually accomplished by selecting some items and sorting them in the order of expected enjoyment (or *utility*). Since the most common way of presenting recommended items is in some form of list, we need an appropriate *ranking* model that can use a wide variety of information to come up with an optimal ordering of the items. In the next section, we will discuss how to design such a ranking model.

4.2 Ranking

The goal of a personalized ranking system is to find the best possible ordering of a set of items for a user within a specific context. At Netflix, we optimize ranking algorithms to put videos that a member is most likely to play and enjoy at the beginning of the list. We do this by learning a scoring function $f_{\texttt{rank}} : U \times V \rightarrow \mathbb{R}$ that maps from our (user, video) space to a score, which is a real number.

An obvious baseline for a ranking function that optimizes consumption is item popularity. The reason is clear: on average, a member is most likely to watch what most others are watching. However, popularity is the opposite of personalization; it will produce the same ordering of items for every user. Thus, the goal becomes to find a personalized ranking function that is better than item popularity, so we can better satisfy users with varying tastes.

Recall that our goal is to recommend the videos that each member is most likely to play and enjoy. One obvious way to approach this is to use the member's predicted rating of each item as an adjunct to item popularity. Using predicted ratings on their own as a ranking function can lead to items that are too niche or unfamiliar. This is because ratings on their own only indicate what someone who has watched a video will rate it, and ignores that most people would rate lowly most videos if they watched them [80]. It can also exclude items that the member may want to watch even though they may not rate them highly. To compensate for this, rather than using either popularity or predicted rating on their own, we would like to produce rankings that balance both of these aspects. One way to do this is to build a ranking prediction model using these two features.

Let us walk through an example of a very simple scoring approach by choosing our ranking function to be a linear combination of popularity and predicted rating. This gives an equation of the form $f_{\texttt{rank}}(u,v) = w_1 p(v) + w_2 r(u,v)$, where u represents the user, v is the video (item), $p(v)$ is the popularity of video v, and $r(u,v)$ is the predicted rating for user u of video v. The bias term that is typically learned as part of a linear model is omitted, since it is a constant and thus does not impact the final ranking. This equation defines a line in a two-dimensional space (see Figure 4).

Once we have such a function, we can pass a set of videos for a given user through it and sort them in descending order according to the score. However, first we need to determine the weights w_1 and w_2 in our model. We can formulate this as a machine learning problem: select positive and negative examples of (user, video) pairs from historical data and let a machine learning algorithm learn the weights that optimize our goal. Treating ranking as a classification or regression problem is known as a pointwise approach in the family of machine learning techniques known as *Learning to Rank*. In addition to recommendations, it is central to application scenarios such as search engines or advertisement targeting. A crucial difference in the case of ranked recommendations is the importance of personalization: we do not expect to optimize a global notion of relevance, but rather a personalized one.

It is interesting to note that in this model, the predicted rating has gone from being the final target variable we are trying to predict to generate a recommendation

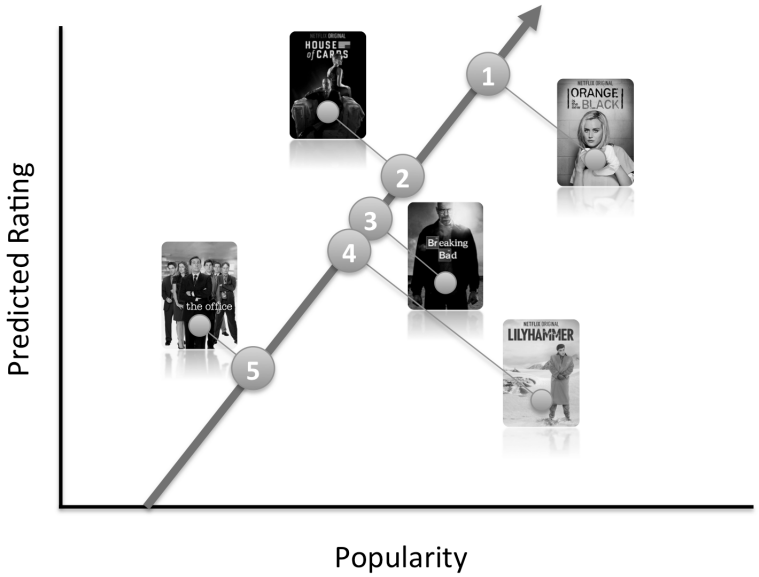

Fig. 4: Constructing a basic personalized two-dimensional ranking function based on popularity and predicted rating.

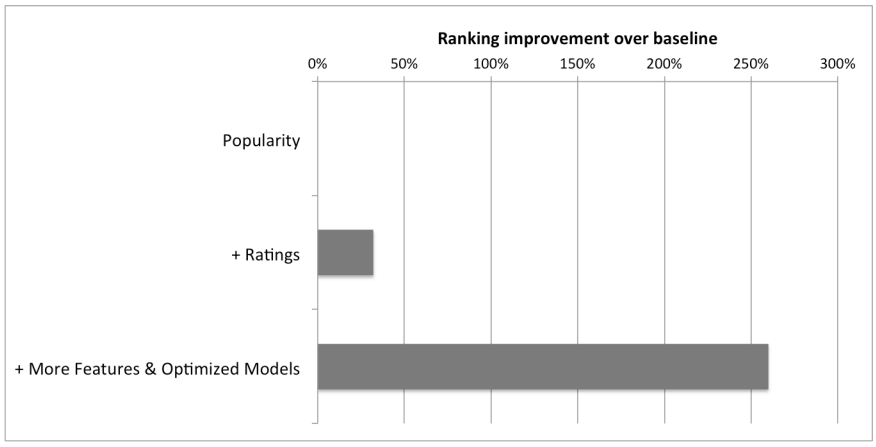

Fig. 5: Performance of Netflix ranking system when adding features and optimizing the learning to rank model. An example ranking metric is shown, but the results hold across a range of such metrics.

to being an input to another model that takes into account other features. A model like this that uses outputs of other models as inputs to produce a final prediction are also referred to as *weighted hybrid models* [14].

The previous two-dimensional model is a very basic example of a ranking function. Apart from popularity and predicted rating, we have tried many other features at Netflix related to many aspects of the video, user, and their interaction. Some have shown no positive effect while others have improved our ranking accuracy tremendously. Features can be simple information derived from metadata or be produced by other recommendation algorithms, as with the case of rating prediction. Also, many supervised classification methods beyond simple linear models can be used or adapted for ranking. In addition, algorithms that directly optimize ranking objectives can be used. Figure 5 shows an improvement in ranking that we obtained by adding different features and optimizing the machine learning approach, such as by using other learning-to-rank approaches like the ones described in section 8.2.

4.3 Page Optimization

Another recognizable aspect of personalization in our service is the selection of "genre" rows. These range from familiar high-level categories like "Comedies" and "Dramas" to highly tailored slices such as "Imaginative Time Travel Movies from the 1980s". Each row represents 3 layers of personalization: the choice of genre itself, the subset of videos selected within that genre, and the ranking of those videos. The set of potential genre rows is very large because they are created from combinations of individual aspects (represented as tags) associated with a video. Thus, the space of genres is much larger than the space of videos, which means selecting them is in itself a recommendation problem. To handle this, row candidates are generated using a member's implicit genre preferences (recent plays, ratings, and other interactions) or explicit feedback provided through our taste preference survey. These elements are used both as input to the selection algorithm as well as evidence to support the recommendations (see Figure 6).

The problem of generating a personalized and optimized page is complex. In the Netflix scenario, there are many thousands of row candidates that can be selected and ranked. As a matter of fact, the catalog of available candidate rows is much larger than the catalog of individual items since the same item can be included in many different rows. On the other hand, when optimizing the page, we are not only optimizing relevance. As with other personalization elements, *freshness* and diversity is taken into account to decide which of the thousands of possible genres to show.

Finally, it is important to note that when optimizing a full page layout, we need to incorporate a model of the user's browsing or attention behavior (see Figure 7) [45, 53]. For example, our model needs to consider whether the probability of the user seeing and clicking the third item in the second row is higher or lower than the probability of seeing and clicking the first item in the fourth row.

Fig. 6: Netflix genre rows can be generated from implicit and/or explicit feedback.

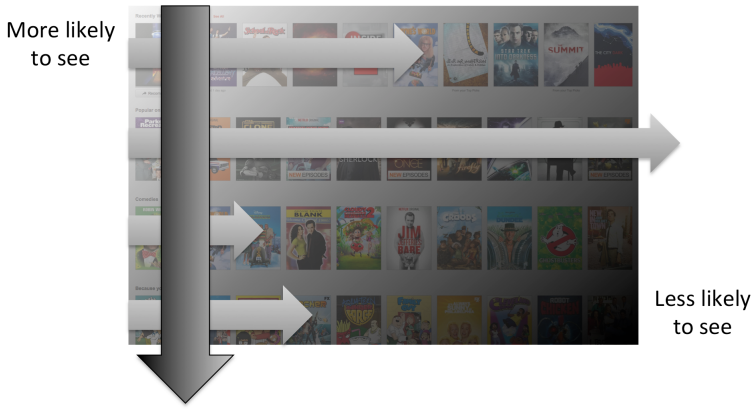

Fig. 7: Browsing and attention behavior of users needs to be taken into account when optimizing the whole page experience.

To conclude this section, it is worth highlighting how the recommendation approach has evolved at a company like Netflix. Starting from its formulation as a rating prediction problem in the Netflix Prize, it evolved to a one-dimensional ranking, and finally to a full-page personalized optimization problem. This evolution is illustrated in Figure 8.

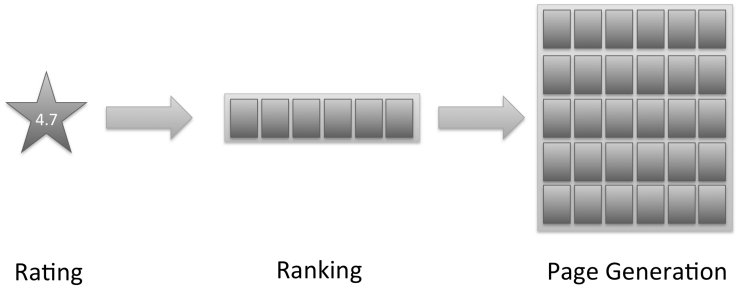

Fig. 8: The recommendation approach at Netflix has evolved from focusing on rating prediction to one-dimensional ranking and now to full page optimization.

5 Data and Models

5.1 Data

The discussion of ranking algorithms in the previous section highlights the importance of both data and models in creating an optimal recommendation experience. The availability of high volumes of high-quality user data allows for us to use some approaches that would have been unthinkable in the past. As an example, we will discuss next some of the data sources that we can use at Netflix to inform our recommendations.

We have large amounts of *play* data about what videos users watch, when, for how long, and on what device they watched it; as of 2013 we had around 50 million play events coming into the service every day. Given that helping our users find something to watch is one of our primary goals, this information about what and how they have watched in the past is very important. We still have several billion item *ratings* from users. We also receive millions of new ratings every day; 5 million per day in 2013. Our users also add millions of items to their *queues* each day. They also directly enter millions of *search queries*; 3 million per day in 2013. Our users can also give explicit feedback on their interests by completing a signup *onramp* or *taste survey* to express preferences.

On the item side, we already mentioned the use of item *popularity* for ranking. We have many ways of computing popularity such as over various time ranges, aggregating user actions in different ways, or grouping users by region or other similarity metrics. Each item in our catalog also has rich *metadata* such as synopsis, genres, actors, directors, subtitles, parental rating, and user reviews. Items also have associated *tag* data, which are human-provided annotations on each video that describe aspects such as mood (e.g. witty, dark, goofy), qualities (e.g. critically-acclaimed, visually-striking, classic), and storyline (e.g. marriage, time travel, talking animals).

Although a manual tagging approach would be unfeasible for other domains with a larger or faster changing catalog, it can be very efficient and practical in a domain like ours where the catalog is in the order of thousands of professionally-produced items. In this case, it would be hard to obtain such high-quality annotations from automatic methods. Finally, we can also tap into *external data* such as box office performance or critic reviews as a basis for additional features to describe an item.

We collect *presentation* and *impression* data that records what items we have recommended to a user, where we have shown them, and if they were rendered on a page in the user interface. We can also observe a user's interactions with the recommendations: scrolls, mouse-overs, clicks, or the time spent on a given page. Using this type of presentation and interaction data, we can look at the effect of showing a recommendation on a user's response. This is important for handling the presentation bias, where a user is more likely to watch a video simply because we put it in a location where they are likely to see it.

Some users choose to provide us with *social* data that we can also use for personalization. Social data may include the social network connections to other users, as well as interactions or activities of those connected users. It can also provide a source of interests (e.g. likes) beyond the scope of items in our catalog or movies and TV shows in general.

There are also many other data sources related to a user or context such as *demographics*, *language preference*, *device*, *location*, or *time* that can be used to derive features for our predictive models.

5.2 Models

Many different modeling approaches have been used for building recommendation systems. One thing we have found at Netflix is that with the great availability of data, both in quantity and types, a thoughtful approach is required to model selection, training, and testing. We use all sorts of machine learning approaches: from unsupervised methods such as *clustering* and *dimensionality reduction* algorithms to a number of supervised classifiers that have shown optimal results in various contexts. This is an incomplete list of methods that are useful to know when working in machine learning for personalization: *Linear regression*, *Logistic regression*, *Elastic nets*, *Singular Value Decomposition*, *Matrix Factorization*, *Restricted Boltzmann Machines*, *Markov Chains*, *Latent Dirichlet Allocation* [10], *Association Rules*, *Factorization Machines* [65], *Gradient Boosted Decision Trees* [25], *Random Forests* [12], and clustering techniques from the simple *k-means* to graphical approaches such as *Affinity Propagation* [24] or non-parametric such as *Hierarchical Dirichlet Processes* [85].

There is no easy answer to how to know which model will perform best for a given problem. In general, the simpler a feature space is, the simpler a model can be. But it is easy to get trapped in a situation where a new feature does not show value because the model cannot learn it. Or, the other way around, to conclude that

a more powerful model is not useful simply because one does not have the feature space that exploits its benefits. As such, it is important to understand how the problem definition, feature design, and model interact to find an optimal combination of them.

Many other chapters in this book (see Chapter **??** on Data Mining Methods for Recommender Systems for example) focus on describing these and other methods and their applicability to recommender systems.

6 Consumer Data Science

An abundance of source data, measurements, and associated experiments allow Netflix to operate as a data-driven organization. We have embedded this approach into our culture from when the company was founded and call it Consumer (Data) Science. Broadly speaking, the main goal of Consumer Science is to effectively innovate for users by using data to drive product decisions. That is accomplished by evaluating ideas rapidly, inexpensively, and objectively. We do this by running many experiments to test ideas. Once something is tested, we want to know the outcome and also understand why an approach succeeded or failed. This kind of approach guides not only how we improve the personalization algorithms or recommendation systems but also the majority of consumer-facing aspects of a service, from user interface design to streaming technology.

We do this in practice by employing the scientific method and conducting randomized controlled experiments, which are called *AB tests* (or bucket tests) [41]. A standard AB test randomly assigns each user to one of two groups: A and B. Typically group A would be the *control group* that would be given the current default experience. Group B that would have some new variation on that experience that is hypothesized to be better than the current experience. We use the following steps in running such an experiment:

1. **Start with a hypothesis**: Algorithm/feature/design X will increase user engagement with our service and ultimately user retention.
2. **Design a test**: Think about issues such as dependent & independent variables, control, and significance.
3. **Implement the test**: Set up the solution or prototype to run in a production environment where it can serve requests.
4. **Execute the test**: Assign users to the different groups and let them respond to the different experiences.
5. **Analyze the test**: Look for statistically significant changes in business metrics (e.g. retention) and try to explain them through variations in the behavioral metrics (e.g. increased selection of recommendations).

When we execute AB tests at Netflix, we track many different metrics (e.g. viewing hours). However, we ultimately trust user retention as our overall evaluation criteria (OEC) [40] because it is a long-term metric that ties directly to the success

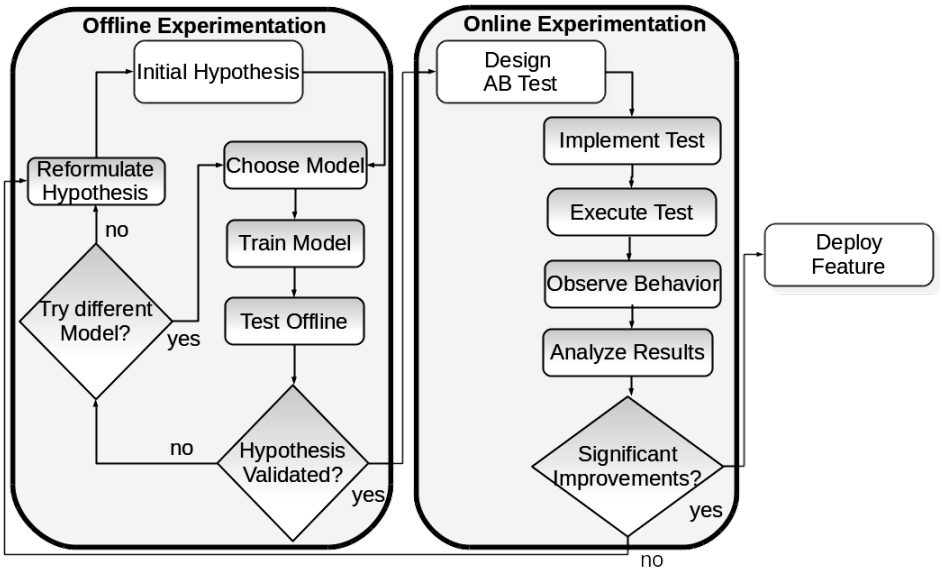

Fig. 9: Following an iterative and data-driven offline-online process for innovating in personalization

of the business as a whole. This is because as a montly subscription service, the longer that a member stays with us, the more revenue we can collect. Of course, if we want to run tests that measure retention, this means we have to let them run for months to measure the effect. Tests usually have thousands of users and anywhere from 2 to 20 experimental groups exploring variations of a base idea. We typically have many AB tests running in parallel, and can independently run tests on different components as long as there is no conflict between them. AB tests let us try radical ideas or test many approaches at the same time, but their key advantage is that they allow our decisions to be data-driven. It also helps us to only keep changes that objectively demonstrate a significant improvement, at least up to some level of statistical confidence, which helps reduce the complexity of our product, systems, and algorithms.

An interesting follow-up question that we have faced is how to integrate machine learning approaches into this data-driven culture of AB testing at Netflix. We have done this with an *offline-online testing procedure* that tries to combine the best of both approaches (see Figure 9). The *offline testing* cycle is a process where we test and optimize our algorithms on historical data prior to performing online AB testing. To measure model performance offline we track multiple metrics: from ranking measures such as normalized discounted cumulative gain and their page-level generalizations, to classification metrics such as accuracy, precision, and recall, to regression metrics such as RMSE, and other metrics to track different aspects of

recommendation like diversity and coverage (see Chapter **??** for more details on the use of offline metrics for evaluating recommender systems). We also keep track of how well these offline metrics correlate to measurable online metrics in our AB tests. However, since the mapping is not perfect, offline performance is only used as an indication to make informed decisions on follow up steps, not to directly deploy an algorithm without AB testing. Note that this correlation of offline and online metrics is an important practical issue that has just started to get some attention in the academic community [98].

Once offline testing has validated a hypothesis, we are ready to design and launch the online AB test that will demonstrate if an algorithmic change is an improvement from the perspective of user behavior. If it does, we will be ready to deploy the algorithmic improvement to the whole user-base. In fact, this is how we developed the personalization experience described in the previous sections: a sequence of AB tests demonstrating that each successive improvement in personalization was better than an unpersonalized method or the previous personalization approach.

We use this combination of offline and online testing for two primary reasons. The first is that setting up offline tests are typically easier in terms of the engineering involved because they do not need to serve millions of users in real-time. They can also be faster because they can look at one sub-problem such as ranking or rating prediction and rapidly evaluate changes at that level: on the scale of hours or days, versus the months it takes to run an AB test to determine impact on long-term metrics. This also leads to the second reason, which is that because we are interested in long-term improvements, not just short-term ones, the pool of users we can allocate to an AB test is a precious resource. This means that we want to make sure we are always keeping the AB test experimentation pipeline full for a feature and allocating users to tests that we have confidence will be better (or at the very least not worse) than the current default experience.

7 Architectures

So far, we have highlighted the importance of using both data and algorithms to create a good personalization experience. We also talked about enriching the interaction and engaging the user with the recommendation system. There is another important piece of the puzzle: how to create a software architecture that can deliver this experience and support rapid innovation. Coming up with a software architecture that handles large volumes of existing data, is responsive to user interactions, and makes it easy to experiment with new recommendation approaches is not a trivial task. In this section we will describe a generic three-layer architecture that addresses these challenges and its particular implementation at Netflix.

We will start by going through the general system architecture in Figure 10. It illustrates a blueprint for multiple personalization algorithm services such as ranking, row selection, and ratings prediction where each provide recommendations involving multi-layered machine learning. To start with, our users generate most of the

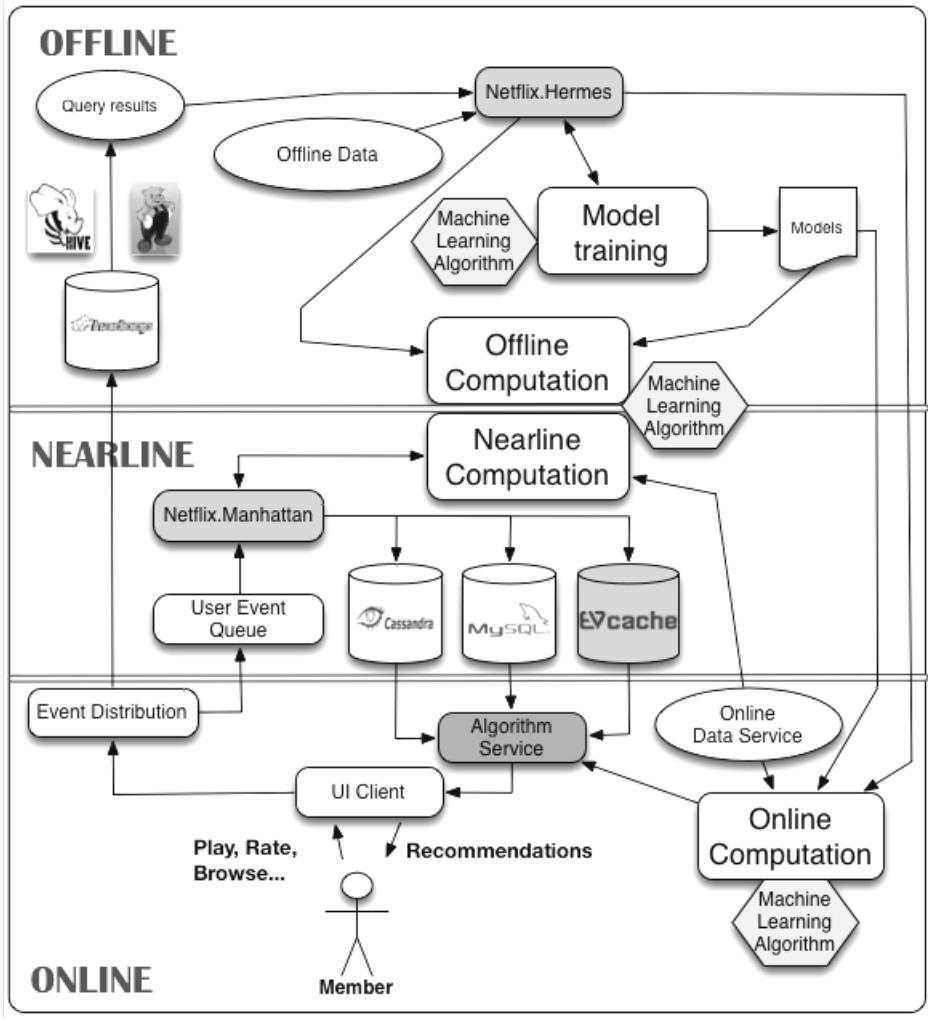

Fig. 10: System-level architecture diagram for a recommendation system. The main components of the architecture contains one or more machine learning algorithms.

events and data of interest to the system and at the end our system generates recommendations to show them. The simplest thing we can do with data is to store it for later offline processing, which provides input for *offline jobs*. However, computation can be done offline, nearline, or online. *Online computation* can respond better to recent events and user interaction, but has to respond to requests in real-time [2].

[2] For practical purposes we consider responses below a few hundred milliseconds (e.g. 200) to be real-time.

This can limit the computational complexity of algorithms deployed as well as the amount of data that can be processed. *Offline computation* has less limitations on the amount of data and the computational complexity of the algorithms since it runs in a batch manner with relaxed timing requirements. However, it can easily grow stale between updates because the most recent data is not incorporated. One of the key issues in a personalization architecture is how to combine and manage online and offline computation in a seamless manner. *Nearline computation* is an intermediate compromise between these two modes in which we can perform online-like computations, but do not require them to be served in real-time. *Model training* is another form of computation that uses existing data to generate a model that will later be used during the actual computation of recommendation results. Another part of the architecture describes how the different kinds of events and data need to be handled by the *event and data distribution* system. A related issue is how to combine the different *signals and models* that are needed across the offline, nearline, and online regimes. Finally, we also need to figure out how to combine intermediate *recommendation results* in a way that makes sense for the user [3]. This whole infrastructure runs across the public Amazon Web Services cloud. The rest of this section will detail the components of this architecture as well as their interactions. In order to do so, we will break the general diagram into different sub-systems and we will go into the details of each of them.

7.1 Event and Data Distribution

The goal of our system is to use past user interaction data to improve the user's future experience. For that reason, we would like the various user interface applications (Smart TVs, tablets, game consoles, etc.) to not only deliver a delightful user experience but also collect as many user actions as possible. These actions can be related to clicks, browsing, viewing, or even the content of the viewport at any time. They can be aggregated to provide base data for our algorithms. Here we try to make a distinction between data and events, although the boundary can be blurry. We think of events as small units of time-sensitive information that need to be processed with low latency. These events are routed to trigger a subsequent process, such as updating a nearline result set. On the other hand, we think of data as more dense information units that might need to be processed and stored for later use. Here the latency is not as important as the information quality and quantity. Of course, there are user actions that can be treated as both events and data and therefore sent to both flows.

At Netflix, our near-real-time event flow is managed through an internal framework called Manhattan. Manhattan is a distributed computation system that is central to our algorithmic architecture for recommendation. It is somewhat similar to

[3] Intermediate recommendations usually represent lists of items that have been pre-selected and even ranked in advanced but need to undergo further processing such as filtering or re-ranking before being presented to the user.

Twitter's Storm, but it addresses different concerns and responds to a different set of internal requirements. The data flow is managed mostly through logging through Chukwa[4] [62] to Hadoop[5] for the initial steps of the process. Later we use Hermes, described in the next section, as our publish-subscribe mechanism.

7.2 Offline, Nearline, and Online Computation

As mentioned above, our algorithmic results can be computed either online in real-time, offline in batch, or nearline in between. Each approach has its advantages and disadvantages, which need to be taken into account for each use case.

Online computation can respond quickly to events and use the most recent data. An example is to assemble a gallery of action movies sorted for a user given the current context. Online components are subject to availability and response time Service Level Agreements (SLA) that specify the maximum latency of the component in responding to requests from client applications while our user is waiting for recommendations to appear. For example, that recommendations need to be returned in at least 250ms for 99% of all requests. This can make it harder to fit complex and computationally costly algorithms in this approach. Also, a purely online computation may fail to meet its SLA in some circumstances, so it is always important to have a fast fallback mechanism such as reverting to a precomputed result. Computing online also means that the various data sources involved also need to be available online, which can require additional infrastructure to serve that data.

On the other end of the spectrum, offline computation enables more algorithmic approaches such as complex algorithms and less limitations on the amount of data that is used. A trivial example might be to periodically aggregate statistics from millions of video play events to compile baseline popularity metrics for recommendations. Offline systems also have simpler engineering requirements. For example, relaxed response time SLAs imposed by clients can be easily met. New algorithms can be deployed in production without the need to put too much effort into performance tuning. In the context of Consumer Science we take advantage of this to support rapid experimentation: if a new experimental algorithm is slower to execute, we can choose to simply deploy more cloud compute instances to achieve the throughput required to run an experiment, instead of spending valuable engineering time optimizing performance for an algorithm that may prove to be of little business value. However, because offline processing does not have strong latency requirements, it can not react quickly to changes in context or new data. Ultimately, this can lead to staleness that may degrade the usefulness of recommendations and thus the user experience. Offline computation also requires having infrastructure for storing, computing, and accessing large sets of precomputed results.

[4] Chukwa is a Hadoop subproject devoted to large-scale log collection and analysis.

[5] Hadoop is an open-source software framework for storage and large-scale processing of data-sets on clusters of commodity hardware.

Much of the computation we need for personalization involving machine learning algorithms can be done offline. This means that the jobs can be scheduled to be executed periodically and their execution does not need to be synchronous with the request or presentation of the results. There are two main kinds of tasks that fall in this category: model training and batch computation of intermediate or final results. In the model training jobs, we collect relevant existing data and apply a machine learning algorithm to produce a set of model parameters (which we will henceforth refer to as the model). The training process usually involves training several models with different hyper-parameters in order to select the optimal one. This final model will usually be encoded and stored in a file for later consumption. Although most of the models are trained offline in batch mode, we also have some incremental learning techniques where training updates are indeed performed online. Batch computation of results is the offline process defined above in which we use existing models and corresponding input data to compute results that will be used at a later time either for subsequent online processing or direct presentation to the user.

Both of these tasks need refined data to process, which usually is generated by running a database query. Since these queries run over large amounts of data, it can be beneficial to run them in a distributed fashion, which makes them very good candidates for running on Hadoop via either Hive[6] or Pig[7] jobs. Once the queries have completed, we use a mechanism for publishing the resulting data. We have several requirements for that mechanism: First, it should notify subscribers when the result of a query is ready. Second, it should support different repositories (not only HDFS[8], but also S3[9] or Cassandra, for instance). Finally, it should transparently handle errors, allow for monitoring, and alerting. At Netflix we use an internal tool named Hermes that provides all of these capabilities and integrates them into a coherent publish-subscribe framework. It allows data to be delivered to subscribers in near real-time. In some sense, it covers some of the same use cases as Apache Kafka [10], but it is not a message/event queue system.

Nearline computation can be seen as a compromise between the two previous modes. In this case, computation is performed exactly like in the online case. However, we remove the requirement to serve results as soon as they are computed and can instead store them, allowing processing to be asynchronous. The nearline computation is done in response to user events so that the system can be more responsive between requests. This opens the door for potentially more complex processing to be done per event. An example is to update recommendations to reflect that a video has been watched immediately after a user begins to watch it. Results can be stored

[6] Apache Hive is a data warehouse infrastructure built on top of Hadoop for providing data summarization, query, and analysis.

[7] Pig is a high-level platform for creating MapReduce programs used with Hadoop using a language called Pig Latin.

[8] The Hadoop Distributed File System (HDFS) is a distributed file system designed to run on commodity hardware.

[9] Amazon S3 (Simple Storage Service) is an online file storage web service offered by Amazon Web Services.

[10] Apache Kafka is publish-subscribe messaging rethought as a distributed commit log.

in an intermediate caching or storage backend. Nearline computation is also a natural setting for applying incremental learning algorithms.

In any case, the choice of online/nearline/offline processing is not an either/or question. All approaches can and should be combined. There are many ways to combine them. We already mentioned the idea of using offline computation as a fallback. Another option is to precompute part of a result with an offline process and leave the less costly or more context-sensitive parts of the algorithms for online computation.

Even the modeling part can be done in a hybrid offline/online manner. This is not a natural fit for traditional supervised classification applications where the classifier has to be trained in batch from labeled data and will only be applied online to classify new inputs. However, approaches such as Matrix Factorization are a more natural fit for hybrid online/offline modeling: some factors can be precomputed offline while others can be updated in real-time to create a more fresh result. Other unsupervised approaches such as clustering also allow for offline computation of the cluster centers and online assignment of clusters. These examples point to the possibility of separating our model training into a large-scale and potentially complex global model training and then alighter user-specific model training or updating phase that can be performed online or nearline.

Regardless of whether we are doing an online or offline computation, we need to think about how an algorithm will handle three kinds of inputs: models, data, and signals. Models are usually small files of parameters that have been previously trained offline. Data is previously processed information that has been stored in some sort of database, such as movie metadata or popularity. We use the term *signals* to refer to fresh information we input to algorithms. This data is obtained from live services and can be made of user-related information, such as what the user has watched recently, or context data such as session, device, or time.

7.3 Recommendation Results

The goal of our recommendation system is to come up with a personalized set of recommendations. These results can be serviced directly from lists that we have previously computed or they can be generated on the fly by online algorithms. Of course, we can think of using a combination of both where the bulk of the recommendations are computed offline and we add some freshness by post-processing the lists with online algorithms that use real-time signals.

At Netflix, we store offline and intermediate results in various repositories to be later consumed at request time: the primary data stores we use are Cassandra[11], EV-

[11] Apache Cassandra is an open source distributed database management system designed to handle large amounts of data across many commodity servers, providing high availability with no single point of failure.

Cache[12], and MySQL[13]. Each solution has advantages and disadvantages over the others. MySQL allows for storage of structured relational data that might be required for some future process through general-purpose querying. However, the generality comes at the cost of scalability issues in distributed environments. Cassandra and EVCache both offer the advantages of key-value stores. Cassandra is a well-known solution for a distributed and scalable NoSQL store. Cassandra works well in some situations, however in cases where we need intensive and constant write operations we find EVCache to be a better fit. The key issue, however, is not so much where to store the results but how to handle the requirements in a way that conflicting goals such as query complexity, read/write latency, and transactional consistency meet at an optimal point for each use case.

8 Research Directions with Industrial Applicability

The Netflix Prize spurred a lot of research advances, but the prize was a simplification of the full recommendation problem. In section 4, we illustrated the broader scope of the recommendation problem by presenting Netflix' comprehensive approach. In this section, we will describe some of the latest advances in Recommender Systems by highlighting some of the most promising research directions. Many of these directions are enabled by the availability of larger amounts of different data such as implicit user feedback, contextual information, or social network interaction data.

8.1 Beyond explicit ratings

Explicit ratings are neither the only feedback we can get from our users nor the best kind of feedback. As already described, explicit feedback is noisy. Another issue is that ratings are provided on an ordinal scale. However, traditional methods wrongly interpret ratings as being linear, for example by computing averages. This issue, however, has been addressed by some recent methods such as OrdRec [95] that treat rating prediction as ordinal regression.

In most real-world situations, implicit feedback is much more readily available than ratings and requires no extra effort on the user side. For instance, with a web page you can have users visiting a URL or clicking on an ad as a positive feedback. In a music service, a user can decide to listen to a song. We already described in section 5.1 that Netflix relies on many different kinds of data, the most important of which is user implicit feedback on the service about what a user watched. Also, many of these recommendation applications focus on helping a user choose

[12] EVCache is a distributed in-memory data store for the cloud.

[13] MySQL is one of the most popular open source relational databases.

an action (click, listen, watch), so it makes sense that information about previous such actions contain highly relevant information for predicting future actions. That is why, besides trying to address some of the issues with explicit ratings, there have been many recent approaches that use the more reliable and readily available data from implicit feedback. For example, Bayesian Personalized Ranking (BPR) [66], uses implicit feedback to compute a personalized ranking.

Implicit and explicit feedback can be combined in different ways [59]. Even the SVD++ approach explained in section 3.1 can combine explicit and implicit feedback. Another way is to use logistic ordinal regression [60] to provide a mapping. Taking a Bayesian approach like Matchbox [81], also offers a framework to integrate different kinds of feedback such as ordinal ratings or implicit like/don't like preferences.

8.2 Personalized Learning to Rank

In section 4 we highlighted the importance of ranking in an online recommendation scenario such as Netflix. The traditional pointwise approach to learning to rank described in section 4.2 treats ranking as a simple binary classification problem where the only input are positive and negative examples. Typical models used in this context include Logistic Regression, Support Vector Machines, or Gradient Boosted Decision Trees.

There is a growing research effort in finding better approaches to ranking. The pairwise approach to ranking, for instance, optimizes a loss function defined on pairwise preferences from the user. The goal is to minimize the number of preference inversions in the resulting ranking. Once we have reformulated the problem this way, we can transform it back into the previous binary classification problem. Examples of such an approach are RankSVM [17], RankBoost [23], RankNet [13], or BPR.

We can also try to directly optimize the ranking of the whole list by using a listwise approach. RankCosine [91], for example, uses similarity between the ranking list and the ground truth as a loss function. ListNet [15] uses KL-divergence as loss function by defining a probability distribution. RankALS [82] defines an objective function that directly includes the ranking optimization and then uses Alternating Least Squares (ALS) for optimizing.

Across these approaches, we use rank-specific information retrieval metrics to measure the performance of a ranking model. Some of those metrics include Normalized Discounted Cumulative Gain (NDCG), Mean Average Precision (MAP), Mean Reciprocal Rank (MRR), or Fraction of Concordant Pairs (FCP). Ideally, we would like to directly optimize our models those same metrics. However, it is hard to optimize machine-learned models directly on these measures since they are not differentiable and standard methods such as gradient descent or ALS cannot be directly applied.

In order to optimize those metrics, some methods find a smoothed version of the objective function to run gradient descent. CLiMF optimizes MRR [76], and TFMAP [75], optimizes MAP in a similar way. AdaRank [93] uses boosting to optimize NDCG. Another method to optimize NDCG is NDCG-Boost [86], which optimizes the expectation of NDCG over all possible permutations. SVM-MAP [94] relaxes the MAP metric by adding it to the SVM constraints. It is even possible to directly optimize the non-differentiable IR metrics by using techniques such as Coordinate Ascent [84], Genetic Programming, Simulated Annealing [36], or even Particle Swarming [21].

8.3 Full Page Optimization

While one-dimensional ranking is already a step beyond rating prediction, we are most interested in optimizing the personalized experience over a complete "page". In order to do that we need to account for several things such as user navigational patterns, attention models and diversity [54]. While this is not a common theme in the literature, there are a few recent papers that are addressing the issue. Amr et. al, for example, present a complete approach to full page optimization in the context of news [3]. Their approach includes a sequential click model for the user and a relevance model that promotes diversity through the use of submodular functions.

8.4 Context-aware recommendations

Most of the work on recommender systems has traditionally focused on the two-dimensional user/item problem. But we know that in practice many other dimensions might effect a user's preference. In the case of Netflix, for example, the user's preference for shows might depend on variables such as time of the day, day of the week, or viewing device. All of teose other dimensions are referred to as context. Using contextual variables represents having to deal with more data and a higher dimensional problem. However, there is the potential for effective improvements in applications that make use of context [28].

Adomavicius and Tuzhilin do a thorough review of approaches to contextual recommendations in chapter **??** of this book. They categorize context-aware recommender systems (CARS) into three types: contextual pre-filtering, where context drives data selection; contextual post-filtering, where context is used to filter recommendations once they have been computed using a traditional approach; and contextual modeling, where context is integrated directly into the model. Although some standard approaches to recommendation could theoretically accept more dimensions, the only a few models have been adapted in this way. Oku et al.'s Context-aware Support Vector Machines (SVM) [58] extend SVMs with context dimensions to do recommendation. Xiong et al. present a Bayesian Probabilistic Tensor Factor-

ization model to capture the temporal evolution of online shopping preferences [92]. The authors show in their experiments that results using this third dimension in the form of a tensor does improve accuracy when compared to the non-temporal case. Multiverse is another multidimensional tensor factorization approach to contextual recommendations that has proved effective in different situations [35]. Another novel approach to contextual recommendations worth mentioning is the one based on the use of Sparse Linear Method (SLIM) [55].

A Factorization Machine [65] is a novel general-purpose regression model that models interactions between pairs of variables and the target by using factors. Factorization Machines have proved to be useful in different tasks and domains [67]. In particular, they can be efficiently used to model the interaction of contextual variables [68].

8.5 Metrics and Evaluation

Another important area of research for recommender systems is the development of metrics that accurately map to user satisfaction with the recommendations. This has been a concern for many years [30,37,51,80], but it is far from being solved. Chapter **??** of this book has a very good survey of the different approaches to evaluating recommender systems.

One of the issues with accuracy metrics is how much they are biased for popularity. Some recent research addresses this by trying to remove this popularity bias [79]. However, accuracy is not the only metric we should look at when evaluating recommendations [52]. Vargas *et al.*, for instance, propose a framework to evaluate also novelty and diversity. In general, we would like to optimize a recommender system to different metrics at the same time. To help with this, there are some recent attempts to introduce a multiple objective optimization function [69, 70]. These approaches deal with how to optimize a recommender system offline by using training data.

However, the ultimate objective should always be to evaluate the system on real users. This is best accomplished through the use of online AB tests. But the use of AB tests can be costly and challenging [40]. Thus, sometimes controlled user experiments might be a tool worth considering [38].

8.6 Class imbalance problems and presentation effects

In the traditional formulation of the recommendation problem, we have pairs of items and users but user feedback values for very few of those dyads. The problem is then formulated as finding a utility function or model to estimate values for missing dyads. However, in cases where we have implicit feedback, the recommendation problem becomes predicting the probability a user will interact with a given item.

There is a big shortcoming in using the standard recommendation formulation in such a setting: we do not have negative feedback. All the data we have is either positive or missing. The missing data includes both items that the user explicitly chose to ignore because they were not appealing and items that would have been perfect recommendations but were never presented to the user [78].

One way to address this class imblance problem is to convert missing examples into both a positive and a negative example, each with a different weight related to the probability that a random exemplar is positive or negative [22]. Another solution is to binarize the implicit feedback values: any feedback value greater than zero means positive preference, while any value equal to zero is converted to no preference [31]. A greater value in the implicit feedback value is used to measure the "confidence" in the fact the user liked the item. For example, in a music listening experience, playing a song would always be considered as positive feedback while the amount of repetitions (or for how long it was listened to) would be interpreted as support.

In many practical situations, we have more information than the simple binary implicit feedback from the user. In particular, we might know whether items not selected by the user were actually displayed to the user. This adds very valuable information, but slightly complicates the formulation of our recommendation problem. We now have three different kinds of values for items: positive, presented but not chosen, and not presented. This issue has been recently addressed by the so-called Collaborative Competitive Filtering (CCF) approach [96]. The goal of CCF is to model not only the collaboration between similar users and items, but also the competition between items for user attention. Another important issue related to how items are presented is the so-called position bias: An item that is presented in the first position of a list is more likely to be seen and chosen than one that is further down [63].

8.7 Social Recommendations

Many applications such as Netflix have access to social network data for some users. The use of this new source of data for recommendations is an active area of research, as highlighted in chapter **??**. Most of the initial approaches to social recommendation[14] relied on the so-called *trust-based model* where the trust (or influence) of others is transmitted through the social network connections [6, 57]. However, it is still unclear whether users prefer recommendations from friends to those coming from other users. For instance, in another study [11], the authors found that the selection of users where the recommendation came from did not make much difference, except if the recipients of the recommendation were made aware of it. In any case, it seems clear that social trust can be used in a positive way to generate explanations and support.

[14] It is important to note that the term "social recommendation" was originally used to describe collaborative filtering approaches [7, 74]

There are other uses of social information. For instance, social network data can be an efficient way to deal with user or item cold-start. Social information can, for instance, be used to select the most informative and relevant users or items for modeling [50]. In terms of selecting users, some recent methods propose using social information to select experts [73] in a similar way as collaborative filtering settings [4].

Social-based recommendations can also be combined with the more traditional content-based or collaborative filtering approaches [33]. Social network information can even be efficiently included in a pure collaborative filtering setting, for example by including it in the matrix factorization objective function [56, 97].

9 Conclusion

The Netflix Prize abstracted the recommendation problem to a simplified proxy of predicting ratings. It is now clear that this objective, accurate prediction of ratings, is just one of many components in an effective industrial recommendation system. These systems also need to take into account factors such as diversity, context, popularity, interest, evidence, freshness, and novelty. Trying to balance these often competing factors can be a daunting task, but we have found that it is best handled using a range of algorithmic approaches and many types of data.

Recommender systems deployed in the wild, such as those at Netflix, have the difficult goal of optimizing the probability a user chooses something and enjoys it enough to come back to the service. In order to do so, we need to figure out the best way employ all the available data: from user interactions to item metadata. We also need to have optimized approaches, appropriate metrics, rapid experimentation frameworks, solid algorithmic techniques, and scalable architectures embeded within a sound methology for figuring out what actually improves the user experience. When we put all of this together, we find ourselves continually making progress towards that goal of creating the best possible recommendation experience for our users.

References

1. Deepak Agarwal, Bee-Chung Chen, Pradheep Elango, and Raghu Ramakrishnan. Content recommendation on web portals. *Commun. ACM*, 56(6):92–101, June 2013.
2. Deepak Agarwal, Bee-Chung Chen, and Bo Pang. Personalized recommendation of user comments via factor models. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, EMNLP '11, pages 571–582, Stroudsburg, PA, USA, 2011. Association for Computational Linguistics.

3. Amr Ahmed, Choon Hui Teo, S.V.N. Vishwanathan, and Alex Smola. Fair and balanced: Learning to present news stories. In *Proceedings of the Fifth ACM International Conference on Web Search and Data Mining*, WSDM '12, pages 333–342, New York, NY, USA, 2012. ACM.
4. X. Amatriain, N. Lathia, J. M. Pujol, H. Kwak, and N. Oliver. The wisdom of the few: a collaborative filtering approach based on expert opinions from the web. In *Proc. of 32nd ACM SIGIR*, SIGIR '09, pages 532–539, New York, NY, USA, 2009. ACM.
5. X. Amatriain, J. M. Pujol, and N. Oliver. I Like It... I Like It Not: Evaluating User Ratings Noise in Recommender Systems. In Geert-Jan Houben, Gord McCalla, Fabio Pianesi, and Massimo Zancanaro, editors, *User Modeling, Adaptation, and Personalization*, volume 5535, chapter 24, pages 247–258. Springer Berlin, 2009.
6. R. Andersen, C. Borgs, J. Chayes, U. Feige, A. Flaxman, A. Kalai, V. Mirrokni, and M. Tennenholtz. Trust-based recommendation systems: an axiomatic approach. In *Proc. of the 17th WWW*, WWW '08, pages 199–208, New York, NY, USA, 2008. ACM.
7. C. Basu, H. Hirsh, and W. Cohen. Recommendation as classification: using social and content-based information in recommendation. In *Proc. of AAAI '98*, AAAI '98/IAAI '98, pages 714–720, Menlo Park, CA, USA, 1998. American Association for Artificial Intelligence.
8. R. M. Bell and Y. Koren. Lessons from the Netflix Prize Challenge. *SIGKDD Explor. Newsl.*, 9(2):75–79, December 2007.
9. Eric Berndhardsson. Music recommendations at spotify, January 2013.
10. David M. Blei, Andrew Y. Ng, and Michael I. Jordan. Latent dirichlet allocation. *J. Mach. Learn. Res.*, 3:993–1022, March 2003.
11. S. Bourke, K. McCarthy, and B. Smyth. Power to the people: exploring neighbourhood formations in social recommender system. In *Proc. of Recsys '11*, RecSys '11, pages 337–340, New York, NY, USA, 2011. ACM.
12. Leo Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.
13. C. Burges, T. Shaked, E. Renshaw, A. Lazier, M. Deeds, N. Hamilton, and G. Hullender. Learning to rank using gradient descent. In *Proceedings of the 22nd ICML*, ICML '05, pages 89–96, New York, NY, USA, 2005. ACM.
14. R. Burke. The adaptive web. chapter Hybrid Web Recommender Systems, pages 377–408. 2007.
15. Z. Cao and T. Liu. Learning to rank: From pairwise approach to listwise approach. In *In Proceedings of the 24th ICML*, pages 129–136, 2007.
16. Oscar Celma. *Music Recommendation and Discovery: The Long Tail, Long Fail, and Long Play in the Digital Music Space*. Springer, 2010.
17. O. Chapelle and S. S. Keerthi. Efficient algorithms for ranking with SVMs. *Information Retrieval*, 13:201–215, June 2010.
18. Wen-Yen Chen, Jon-Chyuan Chu, Junyi Luan, Hongjie Bai, Yi Wang, and Edward Y. Chang. Collaborative filtering for orkut communities: Discovery of user latent behavior. In *Proceedings of the 18th International Conference on World Wide Web*, WWW '09, pages 681–690, New York, NY, USA, 2009. ACM.
19. Abhinandan S. Das, Mayur Datar, Ashutosh Garg, and Shyam Rajaram. Google news personalization: Scalable online collaborative filtering. In *Proceedings of the 16th International Conference on World Wide Web*, WWW '07, pages 271–280, New York, NY, USA, 2007. ACM.
20. James Davidson, Benjamin Liebald, Junning Liu, Palash Nandy, Taylor Van Vleet, Ullas Gargi, Sujoy Gupta, Yu He, Mike Lambert, Blake Livingston, and Dasarathi Sampath. The youtube video recommendation system. In *Proceedings of the Fourth ACM Conference on Recommender Systems*, RecSys '10, pages 293–296, New York, NY, USA, 2010. ACM.
21. E. Diaz-Aviles, M. Georgescu, and W. Nejdl. Swarming to rank for recommender systems. In *Proc. of Recsys '12*, RecSys '12, pages 229–232, New York, NY, USA, 2012. ACM.
22. C. Elkan and K. Noto. Learning classifiers from only positive and unlabeled data. In *Proc. of the 14th ACM SIGKDD*, KDD '08, pages 213–220, New York, NY, USA, 2008. ACM.
23. Y. Freund, R. Iyer, R. E. Schapire, and Y. Singer. An efficient boosting algorithm for combining preferences. *J. Mach. Learn. Res.*, 4:933–969, December 2003.

24. Brendan J. Frey and Delbert Dueck. Clustering by passing messages between data points. *Science*, 315:2007, 2007.
25. Jerome H Friedman. Greedy function approximation: a gradient boosting machine. *Annals of Statistics*, pages 1189–1232, 2001.
26. Yasuhiro Fujiwara, Makoto Nakatsuji, Takeshi Yamamuro, Hiroaki Shiokawa, and Makoto Onizuka. Efficient personalized pagerank with accuracy assurance. In *Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '12, pages 15–23, New York, NY, USA, 2012. ACM.
27. S. Funk. Netflix update: Try this at home. http://sifter.org/ simon/journal/20061211.html, 2006.
28. M. Gorgoglione, U. Panniello, and A. Tuzhilin. The effect of context-aware recommendations on customer purchasing behavior and trust. In *Proc. of Recsys '11*, RecSys '11, pages 85–92, New York, NY, USA, 2011. ACM.
29. Pankaj Gupta, Ashish Goel, Jimmy Lin, Aneesh Sharma, Dong Wang, and Reza Zadeh. Wtf: The who to follow service at twitter. In *Proceedings of the 22Nd International Conference on World Wide Web*, WWW '13, pages 505–514, Republic and Canton of Geneva, Switzerland, 2013. International World Wide Web Conferences Steering Committee.
30. J. L. Herlocker, J. A. Konstan, L. G. Terveen, and J. T. Riedl. Evaluating collaborative filtering recommender systems. *ACM Trans. Inf. Syst.*, 22(1):5–53, 2004.
31. Yifan Hu, Y. Koren, and C. Volinsky. Collaborative Filtering for Implicit Feedback Datasets. In *Proc. of the 2008 Eighth ICDM*, volume 0 of *ICDM '08*, pages 263–272, Washington, DC, USA, December 2008. IEEE Computer Society.
32. Recommendation Systems in the Industry. Recommendation systems in the industry. Tutorial at Recsys 2009, October 2009.
33. M. Jamali and M. Ester. Trustwalker: a random walk model for combining trust-based and item-based recommendation. In *Proc. of KDD '09*, KDD '09, pages 397–406, New York, NY, USA, 2009. ACM.
34. Glen Jeh and Jennifer Widom. Simrank: A measure of structural-context similarity. In *Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '02, pages 538–543, New York, NY, USA, 2002. ACM.
35. A. Karatzoglou, X. Amatriain, L. Baltrunas, and N. Oliver. Multiverse recommendation: n-dimensional tensor factorization for context-aware collaborative filtering. In *Proc. of the fourth ACM Recsys*, RecSys '10, pages 79–86, New York, NY, USA, 2010. ACM.
36. M. Karimzadehgan, W. Li, R. Zhang, and J. Mao. A stochastic learning-to-rank algorithm and its application to contextual advertising. In *Proceedings of the 20th WWW*, WWW '11, pages 377–386, New York, NY, USA, 2011. ACM.
37. George Karypis. Evaluation of item-based top-n recommendation algorithms. In *CIKM '01: Proceedings of the tenth international conference on Information and knowledge management*, pages 247–254, New York, NY, USA, 2001. ACM.
38. Bart P. Knijnenburg. Conducting user experiments in recommender systems. In *Proceedings of the sixth ACM conference on Recommender systems*, RecSys '12, pages 3–4, New York, NY, USA, 2012. ACM.
39. Noam Koenigstein, Nir Nice, Ulrich Paquet, and Nir Schleyen. The xbox recommender system. In *Proceedings of the Sixth ACM Conference on Recommender Systems*, RecSys '12, pages 281–284, New York, NY, USA, 2012. ACM.
40. R. Kohavi, A. Deng, B. Frasca, R. Longbotham, T. Walker, and Y. Xu. Trustworthy online controlled experiments: five puzzling outcomes explained. In *Proceedings of KDD '12*, pages 786–794, New York, NY, USA, 2012. ACM.
41. Ron Kohavi, Randal M. Henne, and Dan Sommerfield. Practical guide to controlled experiments on the web: Listen to your customers not to the hippo. In *Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '07, pages 959–967, New York, NY, USA, 2007. ACM.
42. Y. Koren. Factorization meets the neighborhood: a multifaceted collaborative filtering model. In *Proceedings of the 14th ACM SIGKDD*, KDD '08, pages 426–434, New York, NY, USA, 2008. ACM.

43. Y. Koren. Collaborative filtering with temporal dynamics. In *Proceedings of the 15th ACM SIGKDD*, KDD '09, pages 447–456, New York, NY, USA, 2009. ACM.
44. Y. Koren, R. Bell, and C. Volinsky. Matrix Factorization Techniques for Recommender Systems. *Computer*, 42(8):30–37, August 2009.
45. Dmitry Lagun, Chih-Hung Hsieh, Dale Webster, and Vidhya Navalpakkam. Towards better measurement of attention and satisfaction in mobile search. In *Proceedings of the 37th International ACM SIGIR Conference on Research & Development in Information Retrieval*, SIGIR '14, pages 113–122, New York, NY, USA, 2014. ACM.
46. Paul B. Lamere. I've got 10 million songs in my pocket: Now what? In *Proceedings of the Sixth ACM Conference on Recommender Systems*, RecSys '12, pages 207–208, New York, NY, USA, 2012. ACM.
47. Lihong Li, Wei Chu, John Langford, and Robert E. Schapire. A contextual-bandit approach to personalized news article recommendation. In *Proceedings of the 19th International Conference on World Wide Web*, WWW '10, pages 661–670, New York, NY, USA, 2010. ACM.
48. Greg Linden, Brent Smith, and Jeremy York. Amazon.com recommendations: Item-to-item collaborative filtering. *IEEE Internet Computing*, 7(1):76–80, January 2003.
49. Jiahui Liu, Elin Pedersen, and Peter Dolan. Personalized news recommendation based on click behavior. In *2010 International Conference on Intelligent User Interfaces*, 2010.
50. N. N. Liu, X. Meng, C. Liu, and Q. Yang. Wisdom of the better few: cold start recommendation via representative based rating elicitation. In *Proc. of RecSys '11*, RecSys '11, New York, NY, USA, 2011. ACM.
51. M. R. McLaughlin and J. L. Herlocker. A collaborative filtering algorithm and evaluation metric that accurately model the user experience. In *Proc. of SIGIR '04*, 2004.
52. Sean M. Mcnee, John Riedl, and Joseph A. Konstan. Being accurate is not enough: how accuracy metrics have hurt recommender systems. In *CHI '06: CHI '06 extended abstracts on Human factors in computing systems*, pages 1097–1101, New York, NY, USA, 2006. ACM Press.
53. Vidhya Navalpakkam, LaDawn Jentzsch, Rory Sayres, Sujith Ravi, Amr Ahmed, and Alex Smola. Measurement and modeling of eye-mouse behavior in the presence of nonlinear page layouts. In *Proceedings of the 22Nd International Conference on World Wide Web*, WWW '13, pages 953–964, Republic and Canton of Geneva, Switzerland, 2013. International World Wide Web Conferences Steering Committee.
54. Vidhya Navalpakkam, LaDawn Jentzsch, Rory Sayres, Sujith Ravi, Amr Ahmed, and Alex Smola. Measurement and modeling of eye-mouse behavior in the presence of nonlinear page layouts. In *Proceedings of the 22Nd International Conference on World Wide Web*, WWW '13, pages 953–964, Republic and Canton of Geneva, Switzerland, 2013. International World Wide Web Conferences Steering Committee.
55. X. Ning and G. Karypis. Sparse linear methods with side information for top-n recommendations. In *Proc. of the 21st WWW*, WWW '12 Companion, pages 581–582, New York, NY, USA, 2012. ACM.
56. J. Noel, S. Sanner, K. Tran, P. Christen, L. Xie, E. V. Bonilla, E. Abbasnejad, and N. Della Penna. New objective functions for social collaborative filtering. In *Proc. of WWW '12*, WWW '12, pages 859–868, New York, NY, USA, 2012. ACM.
57. J. O'Donovan and B. Smyth. Trust in recommender systems. In *Proc. of IUI '05*, IUI '05, pages 167–174, New York, NY, USA, 2005. ACM.
58. K. Oku, S. Nakajima, J. Miyazaki, and S. Uemura. Context-aware SVM for context-dependent information recommendation. In *Proc. of the 7th Conference on Mobile Data Management*, 2006.
59. D. Parra and X. Amatriain. Walk the Talk: Analyzing the relation between implicit and explicit feedback for preference elicitation. In Joseph A. Konstan, Ricardo Conejo, José L. Marzo, and Nuria Oliver, editors, *User Modeling, Adaption and Personalization*, volume 6787 of *Lecture Notes in Computer Science*, chapter 22, pages 255–268. Springer, Berlin, Heidelberg, 2011.
60. D. Parra, A. Karatzoglou, X. Amatriain, and I. Yavuz. Implicit feedback recommendation via implicit-to-explicit ordinal logistic regression mapping. In *Proc. of the 2011 CARS Workshop*, 2011.

61. Luiz Pizzato, Tomek Rej, Thomas Chung, Irena Koprinska, and Judy Kay. Recon: A reciprocal recommender for online dating. In *Proceedings of the Fourth ACM Conference on Recommender Systems*, RecSys '10, pages 207–214, New York, NY, USA, 2010. ACM.
62. Ariel Rabkin and Randy Katz. Chukwa: A system for reliable large-scale log collection. In *Proceedings of the 24th International Conference on Large Installation System Administration*, LISA'10, pages 1–15, Berkeley, CA, USA, 2010. USENIX Association.
63. F. Radlinski, M. Kurup, and T. Joachims. How does clickthrough data reflect retrieval quality? In *Proc. of the 17th CIKM*, CIKM '08, pages 43–52, New York, NY, USA, 2008. ACM.
64. Azarias Reda, Yubin Park, Mitul Tiwari, Christian Posse, and Sam Shah. Metaphor: A system for related search recommendations. In *Proceedings of the 21st ACM International Conference on Information and Knowledge Management*, CIKM '12, pages 664–673, New York, NY, USA, 2012. ACM.
65. S. Rendle. Factorization Machines. In *Proc. of 2010 IEEE ICDM*, pages 995–1000. IEEE, December 2010.
66. S. Rendle, C. Freudenthaler, Z. Gantner, and L. S. Thieme. BPR: Bayesian personalized ranking from implicit feedback. In *Proceedings of the 25th UAI*, UAI '09, pages 452–461, Arlington, Virginia, United States, 2009. AUAI Press.
67. S. Rendle, C. Freudenthaler, and L. S. Thieme. Factorizing personalized Markov chains for next-basket recommendation. In *Proc. of the 19th WWW*, WWW '10, pages 811–820, New York, NY, USA, 2010. ACM.
68. S. Rendle, Z. Gantner, C. Freudenthaler, and L. Schmidt-Thieme. Fast context-aware recommendations with factorization machines. In *Proc. of the 34th ACM SIGIR*, SIGIR '11, pages 635–644, New York, NY, USA, 2011. ACM.
69. Marco Tulio Ribeiro, Anisio Lacerda, Adriano Veloso, and Nivio Ziviani. Pareto-efficient hybridization for multi-objective recommender systems. In *Proceedings of the sixth ACM conference on Recommender systems*, RecSys '12, pages 19–26, New York, NY, USA, 2012. ACM.
70. Mario Rodriguez, Christian Posse, and Ethan Zhang. Multiple objective optimization in recommender systems. In *Proceedings of the sixth ACM conference on Recommender systems*, RecSys '12, pages 11–18, New York, NY, USA, 2012. ACM.
71. R. Salakhutdinov, A. Mnih, and G. E. Hinton. Restricted Boltzmann machines for collaborative filtering. In *Proc of ICML '07*, New York, NY, USA, 2007. ACM.
72. Science. Rockin' to the Music Genome. *Science*, 311(5765):1223d–, 2006.
73. X. Sha, D. Quercia, P. Michiardi, and M. Dell'Amico. Spotting trends: the wisdom of the few. In *Proc. of the Recsys '12*, RecSys '12, pages 51–58, New York, NY, USA, 2012. ACM.
74. U. Shardanand and P. Maes. Social information filtering: algorithms for automating word of mouth. In *Proc. of SIGCHI '95*, CHI '95, pages 210–217, New York, NY, USA, 1995. ACM Press/Addison-Wesley Publishing Co.
75. Y. Shi, A. Karatzoglou, L. Baltrunas, M. Larson, A. Hanjalic, and N. Oliver. TFMAP: optimizing MAP for top-n context-aware recommendation. In *Proc. of the 35th SIGIR*, SIGIR '12, pages 155–164, New York, NY, USA, 2012. ACM.
76. Y. Shi, A. Karatzoglou, L. Baltrunas, M. Larson, N. Oliver, and A. Hanjalic. CLiMF: learning to maximize reciprocal rank with collaborative less-is-more filtering. In *Proc. of the sixth Recsys*, RecSys '12, pages 139–146, New York, NY, USA, 2012. ACM.
77. Börkur Sigurbjörnsson and Roelof van Zwol. Flickr tag recommendation based on collective knowledge. In *Proceedings of the 17th International Conference on World Wide Web*, WWW '08, pages 327–336, New York, NY, USA, 2008. ACM.
78. H. Steck. Training and testing of recommender systems on data missing not at random. In *Proc. of the 16th ACM SIGKDD*, KDD '10, pages 713–722, New York, NY, USA, 2010. ACM.
79. Harald Steck. Item popularity and recommendation accuracy. In *Proceedings of the fifth ACM conference on Recommender systems*, RecSys '11, pages 125–132, New York, NY, USA, 2011. ACM.
80. Harald Steck. Evaluation of recommendations: Rating-prediction and ranking. In *Proceedings of the 7th ACM Conference on Recommender Systems*, RecSys '13, pages 213–220, New York, NY, USA, 2013. ACM.

81. D. H. Stern, R. Herbrich, and T. Graepel. Matchbox: large scale online bayesian recommendations. In *Proc. of the 18th WWW*, WWW '09, pages 111–120, New York, NY, USA, 2009. ACM.
82. G. Takács and D. Tikk. Alternating least squares for personalized ranking. In *Proc. of Recsys '12*, RecSys '12, pages 83–90, New York, NY, USA, 2012. ACM.
83. Gábor Takács, István Pilászy, Bottyán Németh, and Domonkos Tikk. Major components of the gravity recommendation system. *SIGKDD Explor. Newsl.*, 9(2):80–83, December 2007.
84. Ming Tan, Tian Xia, Lily Guo, and Shaojun Wang. Direct optimization of ranking measures for learning to rank models. In *Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '13, pages 856–864, New York, NY, USA, 2013. ACM.
85. Yee Whye Teh, Michael I. Jordan, Matthew J. Beal, and David M. Blei. Hierarchical dirichlet processes. *Journal of the American Statistical Association*, 101, 2004.
86. H. Valizadegan, R. Jin, R. Zhang, and J. Mao. Learning to Rank by Optimizing NDCG Measure. In *Proc. of SIGIR '00*, pages 41–48, 2000.
87. S. Vargas and P. Castells. Rank and relevance in novelty and diversity metrics for recommender systems. In *Proceedings of the fifth ACM conference on Recommender systems*, RecSys '11, pages 109–116, New York, NY, USA, 2011. ACM.
88. Jian Wang, Badrul Sarwar, and Neel Sundaresan. Utilizing related products for post-purchase recommendation in e-commerce. In *Proceedings of the Fifth ACM Conference on Recommender Systems*, RecSys '11, pages 329–332, New York, NY, USA, 2011. ACM.
89. Jian Wang, Yi Zhang, Christian Posse, and Anmol Bhasin. Is it time for a career switch? In *Proceedings of the 22Nd International Conference on World Wide Web*, WWW '13, pages 1377–1388, Republic and Canton of Geneva, Switzerland, 2013. International World Wide Web Conferences Steering Committee.
90. Jason Weston, Hector Yee, and Ron Weiss. Learning to rank recommendations with the k-order statistic loss. In *ACM International Conference on Recommender Systems (RecSys)*, 2013.
91. F. Xia, T. Y. Liu, W. Wang, J.and Zhang, and H. Li. Listwise approach to learning to rank: theory and algorithm. In *Proc. of the 25th ICML*, ICML '08, pages 1192–1199, New York, NY, USA, 2008. ACM.
92. L. Xiong, X. Chen, T. Huang, and J. G. Carbonell J. Schneider. Temporal collaborative filtering with bayesian probabilistic tensor factorization. In *Proceedings of SIAM Data Mining*, 2010.
93. J. Xu and H. Li. AdaRank: a boosting algorithm for information retrieval. In *Proc. of SIGIR '07*, SIGIR '07, pages 391–398, New York, NY, USA, 2007. ACM.
94. J. Xu, T. Y. Liu, M. Lu, H. Li, and W. Y. Ma. Directly optimizing evaluation measures in learning to rank. In *Proc. of SIGIR '08*, pages 107–114, New York, NY, USA, 2008. ACM.
95. Koren Y and J. Sill. OrdRec: an ordinal model for predicting personalized item rating distributions. In *RecSys '11*, pages 117–124, 2011.
96. S.H. Yang, B. Long, A.J. Smola, H. Zha, and Z. Zheng. Collaborative competitive filtering: learning recommender using context of user choice. In *Proc. of the 34th ACM SIGIR*, SIGIR '11, pages 295–304, New York, NY, USA, 2011. ACM.
97. X. Yang, H. Steck, Y. Guo, and Y. Liu. On top-k recommendation using social networks. In *Proc. of RecSys '12*, RecSys '12, pages 67–74, New York, NY, USA, 2012. ACM.
98. Jeonghee Yi, Ye Chen, Jie Li, Swaraj Sett, and Tak W. Yan. Predictive model performance: Offline and online evaluations. In *Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '13, pages 1294–1302, New York, NY, USA, 2013. ACM.