Building Industrial-scale Real-world Recommender Systems

Xavier Amatriain
Netflix
xamatriain@netflix.com

Categories and Subject Descriptors: H.3 [Information Search and Retreival]: Information Filtering

General Terms: Algorithms, Experimentation

Keywords: Recommender Systems, Applications

1. INTRODUCTION

In 2006, Netflix announced the Netflix Prize, a machine learning and data mining competition for movie rating prediction. We offered $1 million to whoever improved the accuracy of our existing system called Cinematch by 10%. 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.

A year into the 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 gave them this prize. And, they gave us the source code. We looked at the two underlying algorithms with the best performance in the ensemble. To put these algorithms to use, we had to work to overcome some limitations, for instance that they were built to handle 100 million ratings, instead of the more than 5 billion that we have, and that they were not built to adapt as members added more ratings. But once we overcame those challenges, we put the two algorithms into production, where they are still used as part of our recommendation engine.

You might be wondering what happened with the final Grand Prize ensemble that won the $1M two years later. This is a truly impressive compilation and culmination of years of work, blending hundreds of predictive models to finally cross the finish line. We evaluated some of the new methods offline but the additional accuracy gains that we measured did not seem to justify the engineering effort needed to bring them into a production environment.

This example highlights the fact that, besides improving offline metrics such as the RMSE, recommender systems need to take into account other practical issues such as scalability or deployment. In this tutorial, we go over some of those practical issues that many times are as important as the theory, if not more, in order to build an industrial-scale real-world recommender system.

Copyright is held by the author/owner(s).
RecSys'12, September 9–13, 2012, Dublin, Ireland.
ACM 978-1-4503-1270-7/12/09.

2. BEYOND RATING PREDICTION

We have discovered through the years that there is tremendous value in incorporating recommendations to personalize as much as possible. Personalization starts on our homepage, which consists of groups of videos arranged in horizontal rows. Each row has a title that conveys the intended meaningful connection between the videos in that group. Most of our personalization is based on the way we select rows, how we determine what items to include in them, and in what order to place those items.

Take as a first example the Top 10 row: this is our best guess at the ten titles you are most likely to enjoy. Of course, when we say "you", we really mean everyone in your household. 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 when you see your Top10, you are likely to discover items for dad, mom, the kids, or the whole family. Even for a single person household we want to appeal to your range of interests and moods. To achieve this, in many parts of our system we are not only optimizing for accuracy, but also for **diversity**.

Another important element in 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 encourages members to give feedback that will result in better recommendations. A different way of promoting trust with the personalization component is to provide **explanations** as to why we decide to recommend a given movie or show. We are not recommending it because it suits our business needs, but because it matches the information we have from you: your explicit taste preferences and ratings, your viewing history, or even your friends' recommendations.

Some of the most recognizable personalization in our service is the collection of "genre" rows. Each row represents 3 layers of personalization: the choice of genre itself, the subset of titles selected within that genre, and the ranking of those titles. As with other personalization elements, **freshness** and diversity is taken into account when deciding what genres to show from the thousands possible.

Similarity is also an important source of personalization in our service. Think of similarity in a very broad sense; it can be between movies or between members, and can be in multiple dimensions such as metadata, ratings, or viewing data. Furthermore, these similarities can be blended and used as features in other models. Similarity is used in multiple contexts, for example in response to a member's action such as searching or adding a title to the queue.

The goal of recommender systems 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, such as the various rows on Netflix, we need an appropriate **ranking model** that can use a wide variety of information to come up with an optimal ranking of the items for each of our members.

If you are looking for a ranking function that optimizes consumption, an obvious baseline 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 member. Thus, the goal becomes to find a personalized ranking function that is better than item popularity, so we can better satisfy members with varying tastes. One obvious way to approach this is to use the member's predicted rating of each item as an adjunct to item popularity. At this point, we are ready to build a ranking prediction model using these two features. For example, we could use a simple linear function of the form form $rank(u, v) = w1p(v) + w2r(u, v) + b$, where u=user, v=video item, p=popularity and r=predicted rating

Once we have such a function, we can pass a set of videos through our function and sort them in descending order according to the score. You might be wondering how we can set the weights w1 and w2 in our model. In other words, in our simple two-dimensional model, how do we determine whether popularity is more or less important than predicted rating? There are at least two possible approaches to this. You could sample the space of possible weights and let the members decide what makes sense after many A/B tests. This procedure might be time consuming and not very cost effective. Another possible answer involves formulating this as a machine learning problem: select positive and negative examples from your historical data and let a machine learning algorithm learn the weights that optimize your goal. This family of machine learning problems is known as ”Learning to rank” and is central to application scenarios such as search engines or ad targeting.

As you might guess, apart from popularity and rating prediction, we have tried many other features. Some have shown no positive effect while others have improved our ranking accuracy tremendously. On the other hand, many supervised classification methods can be used for ranking. There is no easy answer to choose which model will perform best in a given ranking problem. The simpler your feature space is, the simpler your 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 you don't have the feature space that exploits its benefits.

3. SYSTEMS & ARCHITECTURE

When we design a Recommender System, we need to take into account under what conditions it will be operated and deployed. As we saw with the outcome of the Netflix Prize, issues such as scalability need to be considered. Another important factor that will determine the feasibility of an approach is the overall system **latency** measured as the time elapsed since the user gives us some feedback to the time the UI will present a different recommendation that is influenced by that input. The best algorithm will be useless if the system is unable to respond to user actions in a timely fashion.

In Netflix, we approach scalability and latency by optimizing our systems in several ways. We take advantage of the scalability of the cloud using Amazon Web Services. We also use Hadoop for distributed data processing, and Cassandra for efficient distributed storage. Our architecture follows an "offline-online" pattern to maximize data throughput while minimizing latency. Heavy data computation jobs that are not very sensitive to latency are processed offline. These jobs can be triggered periodically, or in response to user events. On the other hand, jobs that depend on real-time signals are processed online by taking advantage of the previously computed results. A simple an effective for some of these cases is to design filters that immediately filter out items such as a movie the user just watched. But, not everything can be solved with a simple filter. And, it is important to understand how things like filters can impact algorithms.

4. CONSUMER DATA SCIENCE

The abundance of source data, measurements and associated experiments allow us to operate a data-driven organization. Netflix has embedded this aproach into its culture since the company was founded, and we have come to call it Consumer (Data) Science. We strive for an innovation culture that allows us to evaluate ideas rapidly, inexpensively, and objectively. And, once we test something we want to understand why it failed or succeeded.

So, how does this work in practice? It is a slight variation over the traditional scientific process called A/B testing (or bucket testing): **(1)** Start with a hypothesis: Algorithm/feature/design X will increase member engagement with our service and ultimately member retention. **(2)** Design a test: Develop a solution or prototype. Think about dependent & independent variables, control, and significance. **(3)** Execute the test. **(4)** Let data speak for itself

When we execute A/B tests, we track many different metrics. But we ultimately trust member engagement (e.g. hours of play) and retention. Tests usually have thousands of members and anywhere from 2 to 20 cells exploring variations of a base idea. We typically have scores of A/B tests running in parallel. A/B tests let us try radical ideas or test many approaches at the same time, but the key advantage is that they allow our decisions to be data-driven.

An interesting follow-up question that we have faced is how to integrate our machine learning approaches into this data-driven A/B test culture at Netflix. We have done this with an offline-online testing process that tries to combine the best of both worlds. The offline testing cycle is a step where we test and optimize our algorithms prior to performing online A/B testing. To measure model performance offline we track multiple metrics used in the machine learning community: from ranking measures such as normalized discounted cumulative gain, mean reciprocal rank, or fraction of concordant pairs, to classification metrics such as accuracy, precision, recall, or F-score. We also use the famous RMSE from the Netflix Prize or other more exotic metrics to track different aspects like diversity. We keep track of how well those metrics correlate to measurable online gains in our A/B tests. However, since the mapping is not perfect, offline performance is used only as an indication to make informed decisions on follow up tests.