Recommendation algorithms explained

Online stores suggest products, but how do they know what you like, asks **Julian Bucknall**

What's covered

▶ FILTERING PRODUCT RECOMMENDATIONS

Whether you buy books from Amazon or rent videos from Netflix, online stores want you to rate your transactions. They can then data mine your purchases and ratings to build up a statistical image of you. Using that data, they can make recommendations geared to your likes (and avoiding your dislikes) to encourage you to buy more. How do these algorithms work?

One of the biggest innovations in online shopping – first introduced by Amazon – was the automatically generated recommendation. You logged onto the site and, right there on the home page, the site would make suggestions for products you could purchase. For example, if you were a JavaScript developer like me, you'd see recommendations for programming books using that language, whereas if you were a mother with young kids, you'd see mentions of toys and children's books.

Personal touch

This personalisation of the home page has a big benefit for the online store compared to just displaying top 10 lists or banner ads: the click-through and conversion rates are far higher. Customers are more likely to view and buy the suggested products. The prediction algorithms then are of huge importance to online stores – the more accurate they are, the more the online store will sell. Consider though the problems that must be solved by such a recommendation algorithm. A large online store like Amazon may have millions of customers and millions of items in stock. New customers will have limited information about their preferences, while more established customers may have too much. The data on which these algorithms work is constantly updated and changed. Customers are browsing the site and the prediction algorithm should take the recently browsed items into consideration, for example – it doesn't help if I'm looking for a toy for my youngest niece and all I get are recommendations for jQuery.

The biggest and most important criterion for these systems (apart from accuracy) is speed. The recommendation algorithm must produce suggestions within a second or so. After all, the user is in the process of displaying the store's home page where the recommendations will appear.

Newspapers

Recommendation algorithms aren't just for online stores – they can be used anywhere you'd like to encourage your customers to remain on your site. The New York Times, for example, uses an algorithm to suggest articles that you might enjoy once you've finished reading the current one. The recommendations change once you've logged in and once you've built up some tracking history with their site so that the system can understand what you prefer reading. Staying on their site means that you are less likely to go elsewhere and therefore increases their impression and click ratios with their advertisers.

Other papers rely on 'most viewed' or 'most emailed' lists. These are not personalised suggestions at all, but rather the wisdom of the majority.

Traditionally, these recommendation algorithms have worked by finding similar customers in the database. In other words, they work by finding a set of customers who have bought or rated the same items that you have. Throw out the items you've already purchased or commented on, and then recommend the rest. For example, if I've already bought A and B, and the set of such customers also includes purchases for C, then C is recommended to me.

Collaborative

One of the earliest such algorithms is known as collaborative filtering. In essence, the algorithm represents each customer as a vector of all items on sale. Each entry in the vector is positive if the customer bought or rated the item, negative if the customer disliked the ▶

Spotlight on... Slope One predictors

Slope One is a family of predictive algorithms, all based on the same simple arithmetic on users' ratings. In essence, the predictors pre-calculate the average difference between the ratings for two items from customers who rated both. It turns out that Slope One algorithms are not only easy to implement (there are several open source libraries in different languages that implement them), they're free of patents – unlike the item-to-item collaborative filtering used by Amazon, for example – and they're reasonably accurate.

If you take a look at the ratings for Figure 2 on p74 as an example, we see that Brian rated Toy 1 point lower than Book, and Cory 2 points lower. So, on average, people who rate Toy rate it 1.5 points lower than Book. Hence, using Slope One we would assume that the lower rating applies to everyone, and therefore guess Alan's rating for Toy to be 1.5 points lower than his rating for Book, making 2.5.

Notice how this algorithm breaks down when considering the ratings for Book versus Jeans. The average increase in rating points is 1 from Book to Jeans, so we'd guess Cory's rating to be 6 for Jeans, which is out of range. There are therefore other improvements to and variants of the algorithms, including using weightings and limits to ensure guessed ratings fall into range.

item, or empty if the customer has not made his or her opinion known. Most of the entries are empty for most of the customers. Some variants factor in the popularity of the items to bump up the significance of items that are less popular or familiar.

The algorithm then creates its recommendations by calculating a similarity value between the current customer and everyone else. The most acceptable way to do this is to calculate the angle between the vectors – the simplest method being to calculate the cosine using the dot product divided by the product of the vector lengths. The larger the cosine, the smaller the angle, and therefore the more similar the customers.

As you may have surmised, this process is computationally expensive. There are usually a lot of customers and a lot of calculations have to take place very quickly. There are techniques to reduce the computation (by sampling the customer base or ignoring unpopular items, for example), but in general it's always going to be expensive to calculate recommendations this way.

Customer clusters

Another traditional prediction algorithm involves the use of cluster models. Here the goal is to prepare the customer base by dividing it into clusters and then to assign the current customer to one of the clusters, in theory choosing the cluster with the most similarity. Once the cluster has been identified, the recommendations come from the purchases and ratings from other customers in that particular cluster.

Although the choice of cluster works in roughly the same way as the classification algorithm (we assume that we can calculate a characteristic vector that describes the cluster in much the same way that there is a vector per customer), the real meat of the algorithm is in the creation of the clusters. In general, clustering of customer data is done through a heuristic: start off with some number of empty clusters, assign a randomly selected customer to each, and then assign the other customers to the clusters according to similarity. Since the initial clusters are essentially randomly created, sub-algorithms must be used to merge or split clusters as they are being built up.

Using cluster models is less computationally intensive at the point where you need to make recommendations quickly for a customer. After all, there's less work to be done to find a similar cluster rather than a similar customer. If you like, most of the work is done up front in the creation of the clusters themselves.

Unfortunately, this particular method tends to result in low quality recommendations since the purchases/ratings are averaged out within a cluster. No longer is a particular customer matched to the most similar customer, but instead to the average of a large group of customers. Certainly the number of clusters can be increased to refine the matching, but then you run into the possibility of increasing computation time.

Simple search

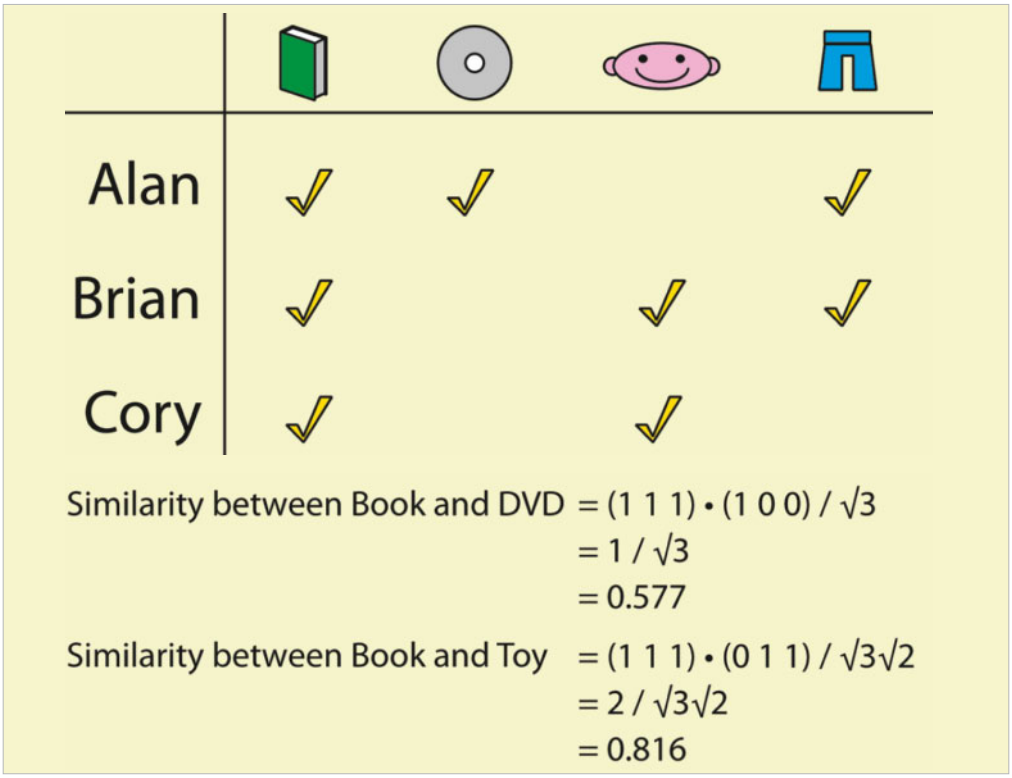
The next traditional algorithm is a fairly simple search algorithm. For example, if I buy Our Kind of Traitor by John Le Carré, the search algorithm would query the items database for other books by John Le Carré, spy books by other authors, DVDs of movies made from Le Carré books, and so on so forth. You can see targeted recommendations like these in banner ads as you surf the internet. I recently searched for a messenger bag (and bought one from Ona), and for a good two weeks afterwards, it was as if every website I visited wanted to recommend Timbuktu bags to me. I'm sure this kind of stalker effect has happened to you and, as you'll know, the recommendations offered tend to be low quality.

Item-to-item

What Amazon did to improve its recommendations was to turn collaborative filtering on its head. Instead of trying to find similar customers, it matched up items. Its version is called item-to-item collaborative filtering. This algorithm matches each of the current customer's purchased and rated items to similar items and then builds a list from those matched items.

First of all, then, the web site must build a 'similar items table' by analysing the items customers tend to purchase together. Here's how this works: for each item X in the catalog, find all customers C who purchased X. For each of those customers, find all items Y purchased by C and record that a customer bought X and Y. Then, for all pairs X and Y, calculate the similarity between X and Y in the same manner as for the collaborative filtering algorithm. Although this calculation is fairly computationally expensive, it can be done beforehand.

Once the similarity between every pair of items has been determined, the recommendation list follows

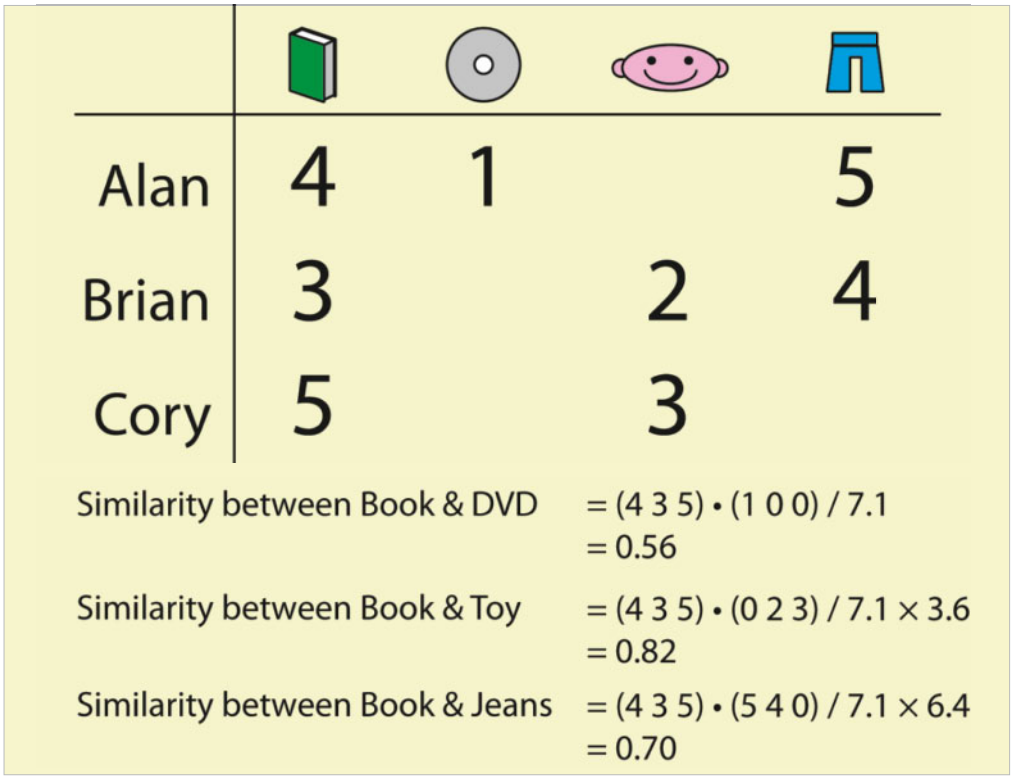


▲ **Figure1: An item-to-item recommendation algorithm based on customers' purchases of products**

Overfitting

As I mentioned in the main article, prediction algorithms are not an exact science. They are statistical in nature, where you are trying to fit various formulas to the data you have in order to predict the data you don't. As with any such process, you run into the danger of overfitting: producing a formula that describes the random noise rather than the underlying data and its relationships. Overfitting is predicated on discovering a large number of parameters for the model; the model is too complex rather than too simple..

As with the Netflix competition, your predictive model with be developed and trained on some set of data. However it will be used, not on that known information, but on some unknown dataset.



▲ **Figure 2: An item-to-item recommendation algorithm based on customers' ratings of products**

easily. Figure 1 shows a simple example where we just have three customers purchasing four products: Alan, Brian, Cory, and Book, DVD, Toy, Jeans. Taking Book first, we see that all the customers purchased it. (We'll assume that 'purchased it' equals 1, and 'didn't purchase it' equals 0.) The similarity between Book and DVD is 0.58, between Book and Toy or Jeans is 0.82. For DVD, only Alan bought it, and he also bought Book and Jeans. The similarity between DVD and Book or Jeans is therefore 1.0. Notice that similarity is not necessarily associative, but in general, over many products and customers, it will be close. From this, if David lands on the page for Book, we can recommend the product Toy or Jeans to him equally, then DVD. If he lands on DVD, we'll suggest Book and Jeans equally, and so on.

Figure 2 shows the situation if, instead of simply buy/didn't buy, we use the customers' ratings instead. (If a customer bought but didn't rate the product, we could assign a neutral rating like 3 to the purchase.) I calculate that the similarity between Book and DVD is 0.56, between Book and Toy 0.82, between Book and Jeans 0.70. Therefore David, on visiting Book, would be recommended Toy, Jeans and DVD in that order.

BellKor

Netflix uses a home-brewed recommendation algorithm called CineMatch. Back in 2006, the film rental company announced a competition with a million dollar prize to see if anyone could improve on the recommendations provided by CineMatch. The goal was a 10 per cent improvement on CineMatch's score on a test subset of the vast Netflix database. The winner, after nearly three years, was a group that called itself BellKor.

The competitors who accepted the challenge were given a huge dataset of 100 million ratings, with each rating (between one and five stars) containing the date of the rating, the title and release year of the movie, and an anonymous ID representing the user. They were also provided a qualifying dataset of a selection of 2.8 million ratings with the same information, but without the actual rating. The object was to devise an algorithm from the large dataset, apply it to the qualifying dataset to guess the ratings, and then Netflix would see how close the guessed ratings were to the actual ratings.

It's fascinating to see the strategies BellKor used in order to tune its algorithm. It must be stressed that BellKor's solution is not a single algorithm per se, but can be viewed instead as a series of algorithmic knobs that can be twiddled to produce the best answer.

Predictors

The first strategy was to create a set of baseline predictors. These describe a user's ratings on the average. Suppose that the average rating over all movies is 3.5. As an example of a specific film, Star Wars might have an overall rating of 4, which would be 0.5 better than the average movie. Our hypothetical user though tends to rate movies lower than the mean: say his average over all movies was 3.2. Since he averages 0.3 lower than the mean, our initial guess for how he might rate Star Wars would be 4 – 0.3, or 3.7.

The second strategy used was the realisation that time plays a big part in people's ratings. Firstly, a movie's popularity will change over time. For example, a movie may start big and then be forgotten, whereas others may start small and then become cult films. Popularity is also affected by the star or director when they release a better (or worse) movie subsequently, and by their appearance, good or bad, in the media.

Time

A user's overall ratings tend to drift over time. This could be because the 'user' is actually a whole household, and so the person making the ratings may change, or it could be because of the psychological effect that after the user has made a run of good ratings, their next rating may be better than would otherwise be warranted (or vice versa: after a run of bad movies, the next good film may be rated lower than expected).

The next strategy could also be described as partially temporal: the user may 'batch' up and enter ratings for a set of recently-viewed movies on one day. The user would tend to let the ratings influence each other unconsciously (if most movies were good, the bad movies would tend to get better than expected ratings, or vice versa), rather than considering them all independently. This strategy is known as frequency.

The BellKor development team then described these strategies mathematically and statistically to provide them parameters to the model that could be tweaked. Taking a large subset of the training data, it repeatedly ran the model, changing the parameters bit by bit, until it predicted the remaining smaller subset's ratings. At that point it was able to submit its guesses for the qualifying subset.

From all this, I'm sure that you can see that prediction algorithms are certainly not exact. But then again, providing they are fast and generally accurate enough, it doesn't matter. For Amazon, the recommendation engine is a differentiating factor, and for Netflix it's a primary reason for keeping customers in their subscriptions – after all, once a user has watched Star Wars and its collection of sequels/prequels, they're going to want suggestions for other things or they'll cancel. ■