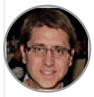
Recommender systems, Part 1

Introduction to approaches and algorithms

Learn about the concepts that underlie web recommendation engines



M. Jones

Published on December 12, 2013

💬 1

Content series:

This content is part 1 of 2 in the series: **Recommender systems**

> **Finding recommender systems**
>
> You can find recommender systems in many of the websites you use every day. including these well-known examples:
>
> **LinkedIn**, the business-oriented social networking site, forms recommendations for people you might know, jobs you might like, groups you might want to follow, or companies you might be interested in. LinkedIn uses Apache Hadoop to build its specialized collaborative-filtering capabilities.
>
> **Amazon**, the popular e-commerce site, uses content-based recommendation. When you select an item to purchase, Amazon recommends other items other users purchased based on that original item (as a matrix of item-to-likelihood-of-next-item purchase). Amazon patented this behavior, called *item-to-item collaborative filtering*.
>
> **Hulu**, a streaming-video website, uses a recommendation engine to identify content that might be of interest to users. It also uses (offline) item-based collaborative filtering with Hadoop to scale the processing of massive amounts of data.

> **Netflix**, the video rental and streaming service, is a famous example. In 2006, Netflix held a competition to improve its recommendation system, Cinematch. In 2009, three teams combined to build an ensemble of 107 recommendation algorithms that resulted in a single prediction. This ensemble proved to be the key to improving predictive accuracy, and the combined team won the prize.
>
> Other sites that incorporate recommendation engines include Facebook, Twitter, Google, MySpace, Last.fm, Del.icio.us, Pandora, Goodreads, and your favorite online news site. Use of a recommendation engine is becoming a standard element of a modern web presence.

Recommendation systems changed the way inanimate websites communicate with their users. Rather than providing a static experience in which users search for and potentially buy products, recommender systems increase interaction to provide a richer experience. Recommender systems identify recommendations autonomously for individual users based on past purchases and searches, and on other users' behavior. This article introduces you to recommender systems and the algorithms that they implement. In Part 2, learn about open source options for building a recommendation capability.

Basic approaches

Most recommender systems take either of two basic approaches: collaborative filtering or content-based filtering. Other approaches (such as hybrid approaches) also exist.

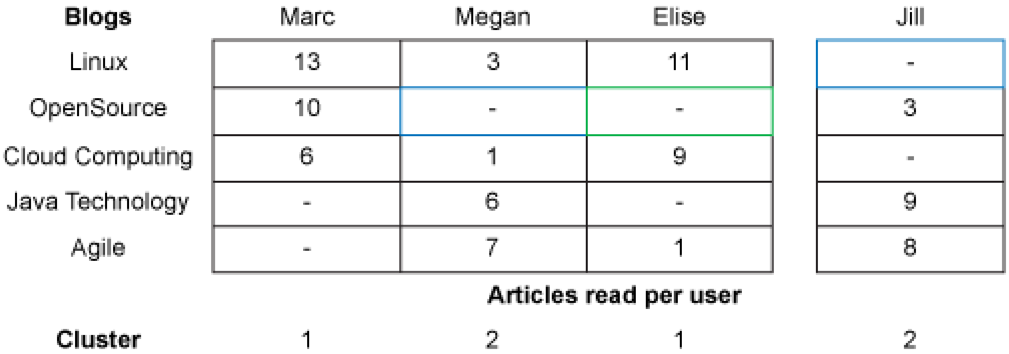
Collaborative filtering

Collaborative filtering arrives at a recommendation that's based on a model of prior user behavior. The model can be constructed solely from a single user's behavior or — more effectively — also from the behavior of other users who have similar traits. When it takes other users' behavior into account, collaborative filtering uses group knowledge to form a recommendation based on like users. In essence, recommendations are based on an automatic collaboration of multiple users and filtered on those who exhibit similar preferences or behaviors.

For example, suppose you're building a website to recommend blogs. By using the information from many users who subscribe to and read blogs, you can group those users based on their preferences. For example, you can group together users who read several of the same blogs. From this information, you identify the most popular blogs that are read by that group. Then — for a particular user in the group — you recommend the most popular blog that he or she neither reads nor subscribes to.

In the table in Figure 1, a set of blogs forms the rows, and the columns define the users. The intersection of blog and user contains the number of articles read by that user of that blog. By clustering the users based on their reading habits (for example, by using a *nearest-neighbor* algorithm), you can see two clusters of two users each. Note the similarities in the reading habits of the members of each cluster: Marc and Elise, who both read several articles about Linux® and cloud computing, form Cluster 1. In Cluster 2 reside Megan and Jill, who both read several articles about Java™ and agile.

Simple example of collaborative filtering



Blogs	Marc	Megan	Elise	Jill
Linux	13	3	11	-
OpenSource	10	-	-	3
Cloud Computing	6	1	9	-
Java Technology	-	6	-	9
Agile	-	7	1	8

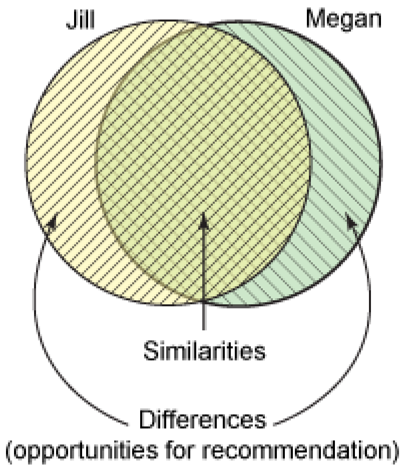
Articles read per user

Cluster	1	2	1	2

Now you can identify some differences within each cluster and make meaningful recommendations. In Cluster 1, Marc read 10 open source blog articles, and Elise read none; Elise read one agile blog, and Marc read none. In Figure 1, then, one recommendation for Elise is the open source blog. No recommendations can be made for Marc because the small difference between him and Elise in agile blog reads would likely be filtered away. In Cluster 2, Jill read three open source blogs, and Elise read none; Elise read 11 Linux blogs, and Jill read none. Cluster 2, then, carries a pair of recommendations: the Linux blog for Jill and the open source blog for Megan.

Another way to view these relationships is based on their similarities and differences, as illustrated in the Venn diagram in Figure 2. The similarities define (based on the particular algorithm used) how to group users who have similar interests. The differences are opportunities that can be used for recommendation — applied through a filter of popularity, for example.

Similarities and differences used in collaborative filtering



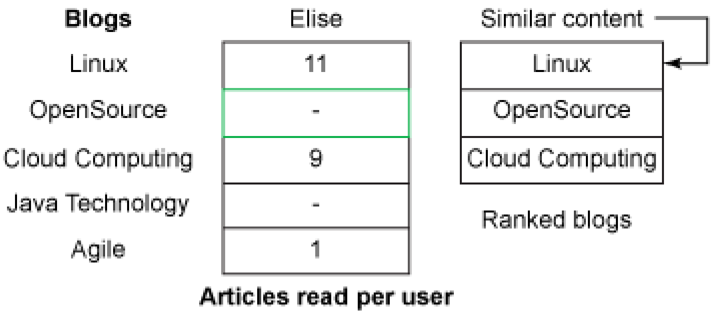
Although Figure 2 is a simplification (and suffers from a sparseness of data by using only two samples), it's a convenient representation.

Content-based filtering

Content-based filtering constructs a recommendation on the basis of a user's behavior. For example, this approach might use historical browsing information, such as which blogs the user reads and the characteristics of those blogs. If a user commonly reads articles about Linux or is likely to leave comments on blogs about software engineering, content-based filtering can use this history to identify and recommend similar content (articles on Linux or other blogs about software engineering). This content can be manually defined or automatically extracted based on other similarity methods.

Returning to Figure 1, focus on the user Elise. If you use a blog ranking that specifies that users who read about Linux might also enjoy reading about open source and cloud computing, you can easily recommend — on the basis of her current reading habits — that Elise read about open source. This approach, illustrated in Figure 3, relies solely on content that a single user accesses, not on the behavior of other users in the system.

Ranked differences used in content-based filtering



Blogs	Elise
Linux	11
OpenSource	-
Cloud Computing	9
Java Technology	-
Agile	1

Articles read per user

Similar content
Linux
OpenSource
Cloud Computing

Ranked blogs

The Venn diagram in Figure 2 also applies here: If one side is the user Elise and the other a ranked set of similar blogs, the similarities are ignored (because those blogs were already read by Elise), and the ranked differences are the opportunities for recommendation.

Hybrids

Hybrid approaches that combine collaborative and content-based filtering are also increasing the efficiency (and complexity) of recommender systems. A simple example of a hybrid system could use the approaches shown in Figure 1 and Figure 3. Incorporating the results of collaborative and content-based filtering creates the potential for a more accurate recommendation. The hybrid approach could also be used to address collaborative filtering that starts with sparse data — known as *cold start*— by enabling the results to be weighted initially toward content-based filtering, then shifting the weight toward collaborative filtering as the available user data set matures.

Algorithms that recommender systems use

As demonstrated by the winning approach for the Netflix prize, many algorithmic approaches are available for recommendation engines. Results can differ based on the problem the algorithm is designed to solve or the relationships that are present in the data. Many of the algorithms come from the field of machine learning, a subfield of artificial intelligence that produces algorithms for learning, prediction, and decision-making.

Pearson correlation

Similarity between two users (and their attributes, such as articles read from a collection of blogs) can be accurately calculated with the *Pearson correlation.* This algorithm measures the linear dependence between two variables (or users) as a function of their attributes. But it doesn't calculate this measure over the entire population of users. Instead, the population must be filtered down to *neighborhoods* based on a higher-level similarity metric, such as reading similar blogs.

The Pearson correlation, which is widely used in research, is a popular algorithm for collaborative filtering.

Clustering algorithms

Clustering algorithms are a form of unsupervised learning that can find structure in a set of seemingly random (or unlabeled) data. In general, they work by identifying similarities among items, such as blog readers, by calculating their distance from other items in a *feature space.* (Features in a feature space could represent the number of articles read in a set of blogs.) The number of independent features defines the dimensionality of the space. If items are "close" together, they can be joined in a cluster.

Many clustering algorithms exist. The simplest one is *k*-means, which partitions items into *k* clusters. Initially, the items are randomly placed into clusters. Then, a *centroid* (or *center*) is calculated for each cluster as a function of its members. Each item's distance from the centroids is then checked. If an item is found to be closer to another cluster, it's moved to that cluster. Centroids are recalculated each time all item distances are checked. When stability is reached (that is, when no items move during an iteration), the set is properly clustered, and the algorithm ends.

Calculating the distance between two objects can be difficult to visualize. One common method is to treat each item as a multidimensional vector and calculate the distance by using the Euclidean algorithm.

Other clustering variants include the Adaptive Resonance Theory (ART) family, Fuzzy C-means, and Expectation-Maximization (probabilistic clustering), to name a few.

Other algorithms

Many algorithms — and an even larger set of variations of those algorithms — exist for recommendation engines. Some that have been used successfully include:

Bayesian Belief Nets, which can be visualized as a directed acyclic graph, with arcs representing the associated probabilities among the variables.

Markov chains, which take a similar approach to Bayesian Belief Nets but treat the recommendation problem as sequential optimization instead of simply prediction.

Rocchio classification (developed with the Vector Space Model), which exploits feedback of the item relevance to improve recommendation accuracy.

Challenges with recommender systems

Taking advantage of the "wisdom of crowds" (with collaborative filtering) has been made simpler with the data-collection opportunities the web affords. But the massive amounts of available data also complicate this opportunity. For example, although some users' behavior can be modeled, other users do not exhibit typical behavior. These users can skew the results of a recommender system and decrease its efficiency. Further, users can exploit a recommender system to favor one product over another — based on positive feedback on a product and negative feedback on competitive products, for example. A good recommender system must manage these issues.

One problem that's endemic to large-scale recommendation systems is scalability. Traditional algorithms work well with smaller amounts of data, but when the data sets grow, the traditional algorithms can have difficulty keeping up. Although this might not be a problem for offline processing, more-specialized approaches are needed for real-time scenarios.

Finally, privacy-protection considerations are also a challenge. Recommender algorithms can identify patterns individuals might not even know exist. A recent example is the case of a large company that could calculate a pregnancy-prediction score based on purchasing habits. Through the use of targeted ads, a father was surprised to learn that his teenage daughter was pregnant. The company's predictor was so accurate that it could predict a prospective mother's due date based on products she purchased.

Going further

Recommendation engines now power most of the popular social and commerce websites. They provide tremendous value to the site's owners and to its users but also have some downsides. This article explored some of the ideas behind recommendation systems and the algorithms that power them. Part 2 of this series introduces you to some of the open source options available for building a recommendation capability.