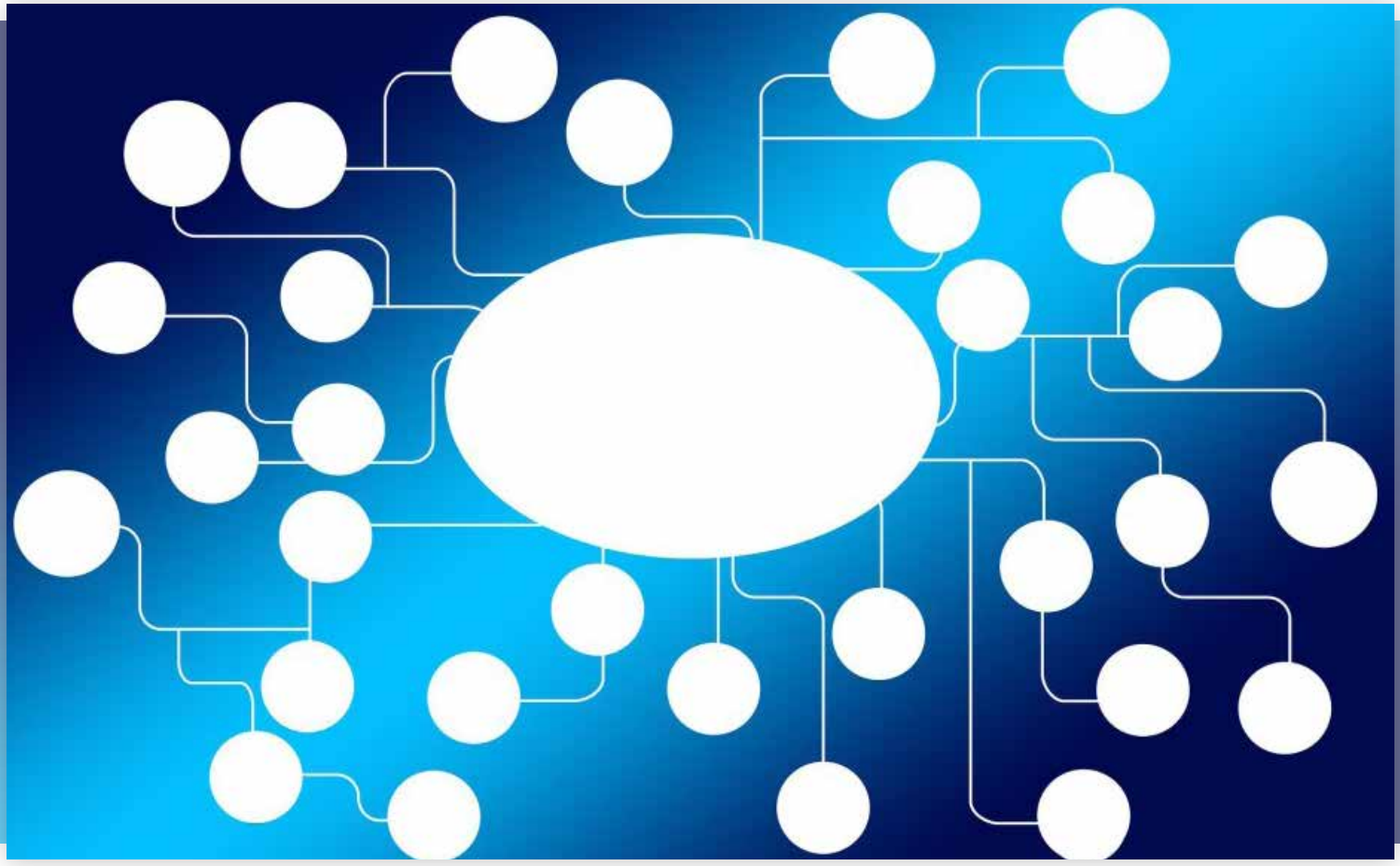
AdvancedRecommendationEngines

A White Paper Written by Bruce Ho, Chief Big Data Scientist, BigR.io, LLC



BigR.io
EMPOWERING DATA

AdvancedRecommendationEngines

A White Paper Written by Bruce Ho, Chief Big Data Scientist, BigR.io, LLC

About the Author

Bruce is a top rated problem solver, communicator, and data scientist. Bruce has a unique strength in bridging software engineering with mathematical algorithms. He has over 15 years of IT experience, engaging in high visibility, large scale projects in marketing automation,
cloud solutions, Big Data engineering and predictive analytics for top technology companies like Amazon, TeraData, and Life Technologies. He is a certified AWS solutions Architect, with a specialty in Big Data practices. Bruce published over 100 scientific papers during his academic career, and continues to draw the best ideas from start-of-the-art research to fuel his Data Science practice.

View Bruce's LinkedIn

Bruce has a deep rooted passion in using statistics modeling to improve the way business is done in the fashion of "Moneyball". His fascination with the use of mathematics to solve real world problems dates back to the early days of

Vector Calculus and Newtonian physics. After witnessing the growth of computing power in the mode of Moore's law, Bruce fully embraces the vision that data driven decision making will soon become a basic tenet of business practice. He loves playing the advocate-in-chief for predictive analysis when resolving client companies' challenges.

Bruce holds a BS in Physics from MIT, and an MS in Electrical Engineering & PhD in Applied Physics from Caltech.

Impact of Recommendation Engines on E-Commerce

When a shopper comes to an eCommerce site, how does she find the items she wants? Search is generally the default option when the site knows nothing about the shopper. Somewhere along the journey though, the shopper starts to feel that the site should know her preferences better than pulling up a few thousand randomly sorted options and dumping them on her. Certainly the corner store used to treat her with a more personal touch.

The key to making customers feel at home is getting to know their tastes, like the corner store used to and suggesting to them the newly arrived silver set that matches their placemats. This less charming but more consistent replacement for the store owner is called the recommendation engine. With Amazon's rise to preeminence, recommendation engine technology has gained some aura of mystique, often called a killer app by industry watchers. The company's continued rapid growth hasn't dampened that enthusiasm. The Netflix $1M challenge only added more fanfare.

The Netflix Challenge

Movies are a good example for where the recommendation engine makes or breaks the business. Viewer tastes and movie features vary widely, and the title list for movies is effectively infinite from the viewer's perspective. Search by title is equally fruitless; you cannot ask viewers to look for what they don't know. Making the right recommendation is a critical factor in determining whether the user comes back again. At Netflix, 75% of the movies watched came from their recommendation engine.

Netflix shook the geek world in 2006 with their announcement of a $1M prize to find the best movie recommendation algorithm. The winning team emerged almost three years later and generously published their approach, which is an ensemble of many algorithms including exotic neural network designs. There are many lessons to be learned from Netflix's investment, with one of the biggest being that Netflix never did deploy the winning algorithm into production. Reason number 1: the extra engineering complexity is not worth the benefit. This outcome very much highlights the importance of a system that's flexible and scalable alongside the quality of the recommendation. More on this at the end of this paper.

The Classic Recommendation Engine Design

Recommendation engine design is one area where common machine learning techniques such as linear regression or SVM (support vector machines) perform poorly due to extreme data sparsity. In 2015, Netflix data shows:

74 million subscribers [2]	13,300 titles [3]	Total cells (user-item pairs): 98 billion	Actual collected ratings: 5 billion ratings ~ 5%

Content Similarity

Content similarity takes the intrinsic characteristics of users or items and estimates their affinity using similarity measures such as Euclidean distance, cosine similarity, or Pearson's product moment coefficient. Someone who likes one comedy movie may like another. Mothers of teens may enjoy a film like "Mom's Night Out".

Content-based strategies require gathering and classifying product and user information that might not be available or easy to collect. The process of collecting features for all the products in the inventory is laborious and often manual. The algorithm is typically brand-specific, and therefore hard to commoditize.

On the user side, the user profile and preferences are often even harder to come by. Users generally won't volunteer their demographics and interests unless they see the return value, which isn't clear in the beginning. Although this strategy can work by relying on well-classified product content alone, at the very least the user must offer up some indication of their personal tastes in order to apply similarity matching.

Even when abundant features are collected, a content-based approach has the downside in that it cannot find hidden patterns. An article that contains "Operation Desert Storm" is probably of interest to readers of "Middle East foreign policy", but the association would be overlooked unless identical keyword tokens appear in both.

Collaborative Filtering

Collaborative filtering simply takes user rating data for each product without doing any intrinsic classification. The basis for matching is that a user who shares interest in some of the same products as another user may also like her other favorite products.

For example, if bike riders often buy sunglasses, the association may indicate interest in sunglasses by another biker who hasn't bought sunglasses, even though there is no direct match between bike and sunglasses features.

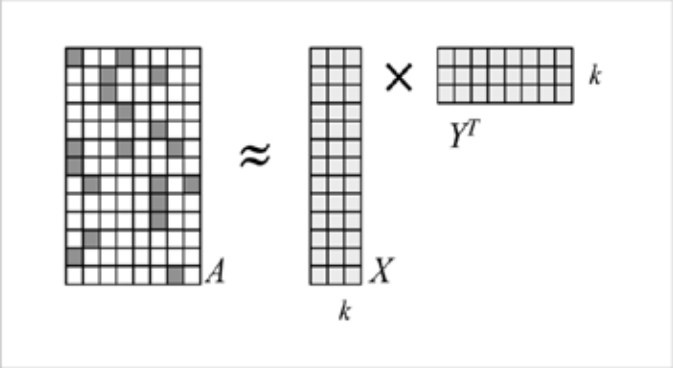


Figure 1. The input ratings matrix A is re-factored into a user vector and product vector, both expressed in terms of hidden f actors with k dimensions.

The input data of the rating matrix can be formidable in size – consider the data set example given above for Netflix. However, there is a machine learning technique known as Matrix Factorization that both reduces the computation complexity and discovers hidden rating similarity. Here, both user preferences and item features are expressed in terms of latent factors, vectors with only a fraction of the dimensions of the original ratings matrix. With the latent factors determined, the recommendation score for any unidentified user item pairs (missing from the original ratings matrix) can be derived using the dot product of the respective latent factor vectors. For example, the movie Troy would have a high values in latent factors "Action", and "Brad Pitt" (these are not likely latent factors, only used here for illustration). Someone who enjoys "Action" and "Brad Pitt" would also have high values for these two factors. This results in a high dot product for this user – movie combination and, therefore, a high recommendation score.

This technique works well with rather sparse data. As shown with the Netflix dataset, even a 5% initial population yields surprisingly useful results. However, this method does have trouble with a brand new user who has no recorded ratings history. This is known as the cold start problem. Only generic recommendations can be made in that case, until ratings build up over time.

For this reason, content similarity is sometimes used to supplement collaborative filtering to overcome the cold start problem, at the expense of doubling the setup cost.

As well as the combination of content similarity and collaboration filtering works, there are notable areas where they fail to reach new heights. For example, sometimes features interact. Someone can rate both comedy and romance very low, yet enjoy the subcategory of romantic comedy. Considering the possible pairs of all the features, a direct inclusion of all combinations would quickly overwhelm the system, and likely for very a small percent in return.

Another area for improvement is freshness of ratings. How can the engine assign more weight to newer ratings and yet not completely discard older ones? A single ratings matrix has no provision for this enhancement, yet shopper taste does shift over time. It is naturally desirable for the recommendations to keep up.

In time, the bright minds in the field start to dream of a better way, which:

- Accepts both user product rating and content features
- Can be computed with only linear complexity
- Takes account of features interactions in an efficient manner
- Includes timeliness as one of the influencers

By implication, this method would behave well with sparse data, yet mitigate the cold start problem. If the algorithm is based on linear regression, it would be cream on top in terms of computational efficiency, as well as opening up wider inclusion of additional characteristics that might indicate a user's preferences.

Advanced Techniques - Factorization Machine

Factorization Machine was invented by German researcher Steffen Rendle in 2010. This method lays out both user and product as unique features, among any other general features (such as freshness of ratings), at the discretion of the data scientist. The key innovation is where Rendle introduces cross-feature terms (equivalent to quadratic polynomial minus self-interacting terms in standard linear regression), but then applies the latent factor technique to reduce computation complexity. The cross-feature terms, on the one hand, express the user item pairs inherent in collaborative filtering, and on the other, capture any potential cross feature interactions.

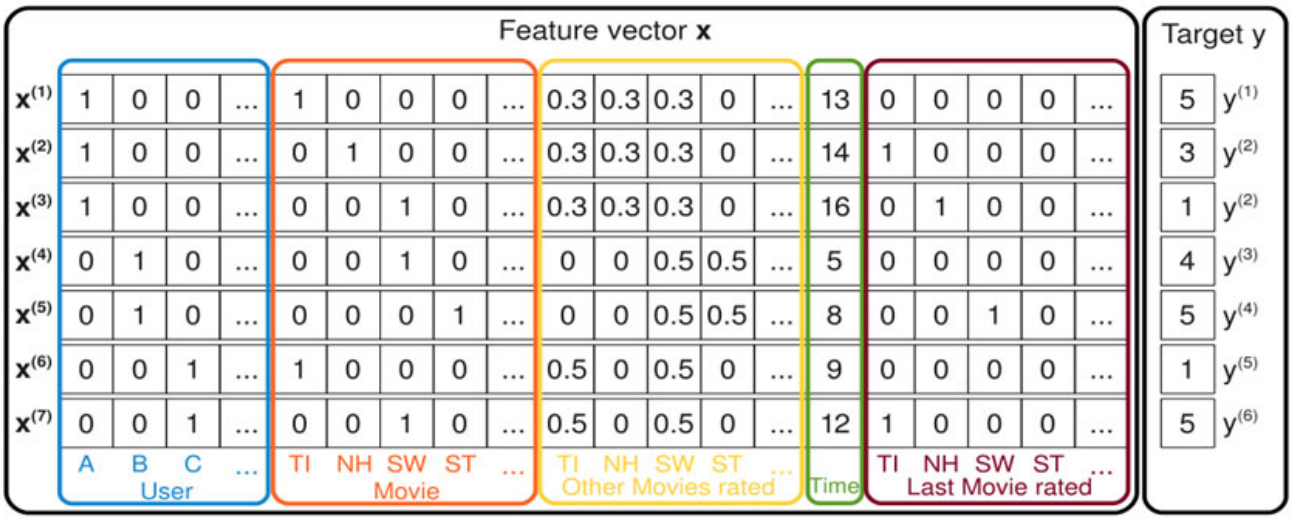


Figure 2. Factorization Machine accepts features which are a superset of content similarity and collaborative filtering. It includes both user item pairs and general content characteristics, while optimizing computation efficiency using the latent factor method of reduction.

Through clever formula reduction, Rendle brought the complexity back down to O (n).
No one ever got a job at Google or Amazon by proposing a O (n^2) solution! This reduction is critical in judging whether a solution is sound in practice.

Now that the problem is expressed as a system of linear equations, the direct implication is that parallelization techniques such as SGD (Stochastic Gradient Descent) can be applied to effect linear scalability. This leads to the system architectural considerations discussed in the next section.

Factorization machine is one of the many specialty approaches BigR.io's expertise can offer. It is an illustration of our philosophy to help our clients design the highest performing and most advanced solutions in any given situation, while balancing budget and other operational concerns.

Deployment System Architecture

What did Netflix learn for its $1M investment in the ultimate recommendation algorithm? According to Netflix: “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.” Clearly, if production viability was included in the competition criteria, they may have made better use of the prize money. Case in point: 100 million ratings were published for the competition, but the actual production load was 5 billion.

The engineering disciplines necessary to bring advanced algorithms to deployment cannot be overlooked. The algorithm should be adaptable to a scalable implementation (such as Apache Spark), and not simply dropped into a faster box in its original “data science” form.

From a system architecture perspective, model training and on-demand predictions for online users have very different requirements in data volume, computational load, and response time, and should always be launched in separate subsystems.

Often, it is also useful to introduce a third caching layer, in which the system predicts the next set of predictions in response to user events. While users are digesting the first prediction or consuming the recommended item, the system pre-computes the likely follow-up and caches the results awaiting future user requests. The potentials for performance gain are numerous and only await diligent discoveries.

Conclusion

Recommendation technology has come a long way since the dawn of eCommerce. Advances in matching algorithms have been nothing short of stunning. Factorization Machine not only eliminates the cold start barrier, it incorporates feature interactions and rating freshness; important functions that were missing from previous generation technology. At the same time, it fully embraces parallel processing and therefore is well suited to Big Data platforms.

BigR.io is uniquely qualified to not only fine tune the recommendation algorithm for your business model, but to deliver a complete solution which meets the scalability and operational requirements for a high-volume, fast-response production system.