

The Netflix Prize and Production Machine Learning Systems: An Insider Look

Posted by **Loren Shure**, April 22, 2015

Do you watch movies on Netflix? Binge-watch TV series? Do you use their movie recommendations? Today's guest blogger, Toshi Takeuchi, shares an interesting blog post he saw about how Netflix uses machine learning for movie recommendations.



Contents

How Recommender Systems Work

What Netflix did with the winning solutions

So, Was It Worth \$1M?

Lessons Learned: New Metrics

Lessons Learned: System Architecture

MATLAB on Hadoop and MATLAB Production Server

Closing

Back in 2006 Netflix announced a famed machine learning and data mining competition "Netflix Prize" with a \$1 million award, finally claimed in 2009. It was a turning post that led to Kaggle and other data science competitions we see today.

With all the publicity and media attention it got, was it really worth \$1 million for Netflix? What did they do with the winning solutions in the end? I came across a very interesting insider blog post "Netflix Recommendations: Beyond the 5 stars" (<http://techblog.netflix.com/2012/04/netflix-recommendations-beyond-5-stars.html>) that reveals practical insights about what really matters not just for recommender systems but also generally for any real world commercial machine learning applications.

How Recommender Systems Work

The goal of the NetFlix Prize was to crowdsource a movie recommendation algorithm that delivers 10%+ improvement in prediction accuracy over the existing system. If you use Netflix, you see movies listed under "movies you may like" or "more movies like so-and-so", etc. These days such recommendations are a huge part of internet retail businesses.

It is probably useful to study a very simple example recommendation system based on a well known algorithm called Collaborative Filtering (http://en.wikipedia.org/wiki/Collaborative_filtering). Here is a toy dataset of movie ratings from 6 fictitious users (columns) for 6 movies released in 2014 (rows).

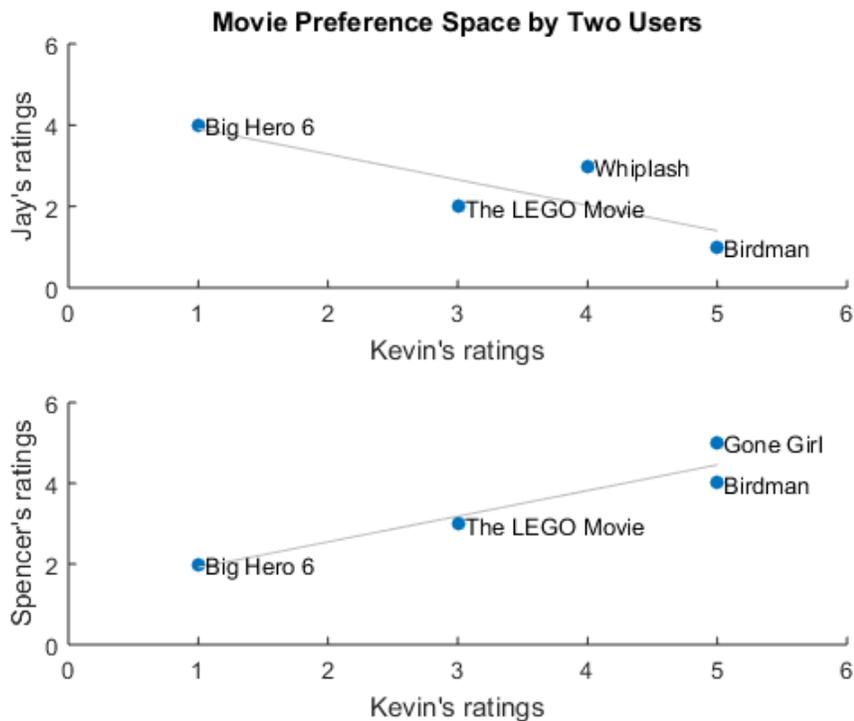
```
movies = {'Big Hero 6', 'Birdman', 'Boyhood', 'Gone Girl', 'The LEGO Movie', 'Whiplash'};
users = {'Kevin', 'Jay', 'Ross', 'Spencer', 'Megan', 'Scott'};
ratings = [1.0, 4.0, 3.0, 2.0, NaN, 1.0;
           5.0, 1.0, 1.0, 4.0, 5.0, NaN;
           NaN, 2.0, 2.0, 5.0, 4.0, 5.0;
           5.0, NaN, 3.0, 5.0, 4.0, 4.0;
           3.0, 2.0, NaN, 3.0, 3.0, 3.0;
           4.0, 3.0, 3.0, NaN, 4.0, 4.0];
```

The idea behind Collaborative Filtering is that you can use the ratings from users who share similar tastes to predict ratings for unrated items. To get an intuition, let's compare the ratings by pairs of users over movies they both rated. The plot of ratings represents their preference space. The best-fit line should go up to the right if the relationship is positive, and it should go down if not.

```

figure
subplot(2,1,1)
scatter(ratings(:,1),ratings(:,2),'filled')
lsline
xlim([0 6]); ylim([0 6])
title('Movie Preference Space by Two Users')
xlabel('Kevin's ratings'); ylabel('Jay's ratings')
for i = 1:size(ratings,1)
    text(ratings(i,1)+0.05,ratings(i,2),movies{i})
end
subplot(2,1,2)
scatter(ratings(:,1),ratings(:,4),'filled')
lsline
xlim([0 6]); ylim([0 6])
xlabel('Kevin's ratings'); ylabel('Spencer's ratings')
for i = 1:size(ratings,1)
    text(ratings(i,1)+0.05,ratings(i,4),movies{i})
end

```



By looking at the slope of the best-fit lines, you can tell that Kevin and Jay don't share similar tastes because their ratings are negatively correlated. Kevin and Spencer, on the other hand, seem to like similar movies.

One popular measure of similarity in Collaborative Filtering is Pearson's correlation coefficient (cosine similarity (<http://blogs.mathworks.com/loren/2015/04/08/can-you-find-love-through-text-analytics/>) is another one). It ranges from 1 to -1 where 1 is positive correlation, 0 is no correlation, and -1 is negative correlation. We compute the pairwise correlation of users using rows with no missing values. What are the similarity scores between Kevin and Jay and Kevin and Spencer?

```

sims = corr(ratings, 'rows', 'pairwise');
fprintf('Similarity between Kevin and Jay:    %.2f\n',sims(1,2))
fprintf('Similarity between Kevin and Spencer: %.2f\n',sims(1,4))

```

```
Similarity between Kevin and Jay:    -0.83
Similarity between Kevin and Spencer: 0.94
```

Because Kevin and Jay have very different tastes, their similarity is negative. Kevin and Spencer, on the other hand, share highly similar tastes. Users who share similar tastes are called neighbors and we can predict ratings of unrated items by combining their existing ratings for other items. But we need to find those neighbors first. Let's find the neighbors for Kevin.

```
sims = sims - eye(length(users)); % set self-correlations to 0
kevin_corrs = sims(1,:);
[ng_h_corr, ng_h_idx] = sort(kevin_corrs,'descend');
ng_h_corr
```

```
ng_h_corr =
    0.9661    0.9439    0.8528         0   -0.4402   -0.8315
```

Kevin has three neighbors who have a high correlation with him. We can use their ratings and correlation scores to predict Kevin's ratings. The weighted average method is a basic approach to make predictions. Because the rating scale can be different among individuals, we need to use mean-centered ratings rather than raw ratings. Kevin hasn't rated 'Boyhood' yet. Would he like it?

```
kevin_mu = nanmean(ratings(:,1));           % Kevin's average rating
ng_h_corr(4:end) = [];                      % drop non-neighbors
ng_h_idx(4:end) = [];                       % drop non-neighbors
ng_h_mu = nanmean(ratings(:,ng_h_idx),1);   % neighbor average ratings
Predicted = nan(length(movies),1);         % initialize an accumulator

for i = 1:length(movies)                   % loop over movies
    ng_h_r = ratings(i,ng_h_idx);          % neighbor ratings for the movie
    isRated = ~isnan(ng_h_r);              % only use neighbors who rated
    meanCentered = ...                     % mean centered weighted average
        (ng_h_r(isRated) - ng_h_mu(isRated)) * ng_h_corr(isRated)'...
        / sum(ng_h_corr(isRated));
    Predicted(i) = kevin_mu + meanCentered; % add Kevin's average
end

Actual = ratings(:,1);                     % Kevin's actual ratings
table(Actual, Predicted, 'RowNames', movies) % compare them to predicted

fprintf('Predicted rating for "%s": %.d\n', movies{3}, round(Predicted(3)))
```

```
ans =
           Actual    Predicted
-----
Big Hero 6         1         1.4965
Birdman            5         4.1797
Boyhood           NaN         4.5695
Gone Girl         5         4.2198
The LEGO Movie    3         2.8781
Whiplash          4         3.9187
Predicted rating for "Boyhood": 5
```

Looks like Kevin would rate 'Boyhood' as a 5-star movie. Now, how accurate was our prediction? The metric used in the Netflix Prize was Root Mean Square Error (RMSE). Let's apply it to this case.

```
RMSE = sqrt(nanmean((Predicted - Actual).^2))
```

```
RMSE =  
0.5567
```

Now you saw how basic Collaborative Filtering worked, but an actual commercial system is naturally much more complex. For example, you don't usually see raw predicted ratings on the web pages. Recommendations are typically delivered as a top-N list. If so, is RMSE really a meaningful metric? For a competition, Netflix had to pick a single number metric to determine the winner. But choosing this metric had its consequences.

What Netflix did with the winning solutions

The goal of the competition was to crowdsource improved algorithms. \$1M for 3 years of R&D? One of the teams spent more than 2000 hours of work to deliver 8.43% improvement. Combined, a lot of people spent enormous amounts of time for a slim hope of the prize. Did Netflix use the solutions in the end?

They did adopt one solution with 8.43% improvement, but they had to overcome its limitations first.

The number of ratings in the competition dataset was 100 million, but the actual production system had over 5 billion

The competition dataset was static, but the number of ratings in the production system keeps growing (4 million ratings per day when the blog post was written)

They didn't adopt the grand prize solution that achieved 10% improvement.

Additional accuracy gains that we measured did not justify the engineering effort needed to bring them into a production environment

The Netflix business model changed from DVD rental to streaming and that changed the way data is collected and recommendations are delivered.

Why additional accuracy gains may not be worth the effort? For example, you could improve the RMSE by closing the prediction gaps in lower ratings - movies people would hate. Does that help end users? Does that increase revenue for Netflix? Probably not. What you can see here is that, in a production system, **scalability** and **adaptability** to changing business needs are bigger challenges than RMSE.

Had Netflix chosen for the competition some metrics more aligned with the needs of the production system, they might have had an easier time adopting the resulting solutions.

So, Was It Worth \$1M?

You often learn more from what didn't go well than what went well. Netflix was not able to take full advantage of the winning solutions, but they certainly appear to have learned good lessons based on how they now operate now. I would say they got well more than \$1M's worth from this bold experiment. Let's see how Netflix is taking advantage of lessons learned.

Lessons Learned: New Metrics

When Netflix talks about their current system, it is notable what they highlight.

"75% of what people watch is from some sort of recommendation"

"continuously optimizing the member experience and have measured significant gains in member satisfaction"

What they now care about is usage, user experience, user satisfaction, and user retention. Naturally they are also well aligned with the bottomline of Netflix as well. The second bullet point refers to A/B testing Netflix is conducting on the live production system. That means they are constantly changing the system...

Lessons Learned: System Architecture

"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." writes the Netflix blogger in "System Architectures for Personalization and Recommendation" (<http://techblog.netflix.com/2013/03/system-architectures-for.html>). You can also see more detail in this presentation (<http://www.slideshare.net/justinbasilico/lessons-learned-from-building-machine-learning-software-at-netflix>).

One of the techniques used in the winning solutions in the Netflix Prize was an ensemble method called Feature-Weighted Linear Stacking (<http://arxiv.org/abs/0911.0460>). Netflix apparently adopted a form of linear stacking technique to combine the predictions from multiple predictive models to produce final recommendations. The blog post gives an example: if u = user, v = video item, p = popularity, r = predicted rating, b = intercept, and w_1 and w_2 are respective weights, then a simple combination of popularity score and predicted ratings by Collaborative Filtering can be expressed as:

$$f_{rank}(u, m) = w_1 \cdot p(v) + w_2 \cdot r(u, v) + b$$

And you can just add more terms to this linear equation as you develop more predictive models, each running on its subsystem... This is a very flexible architecture and makes it easy to run an A/B test - it is just a matter of changing weights!

Netflix uses three layers of service to achieve this - offline, nearline and online to overcome this challenge.

Offline to process data - pre-computes the time-consuming steps in a batch process.

Online to process requests - responds to user action instantaneously by taking advantage of the Offline and Nearline outputs

Nearline to process events - bridges the two sub-systems by pre-computing frequently performed operations and caching the result in advance of active user action

As a simple example for an offline process, let's use the MovieLens (<http://grouplens.org/datasets/movielens/>) dataset with 100K ratings from 943 users on 1682 movies.

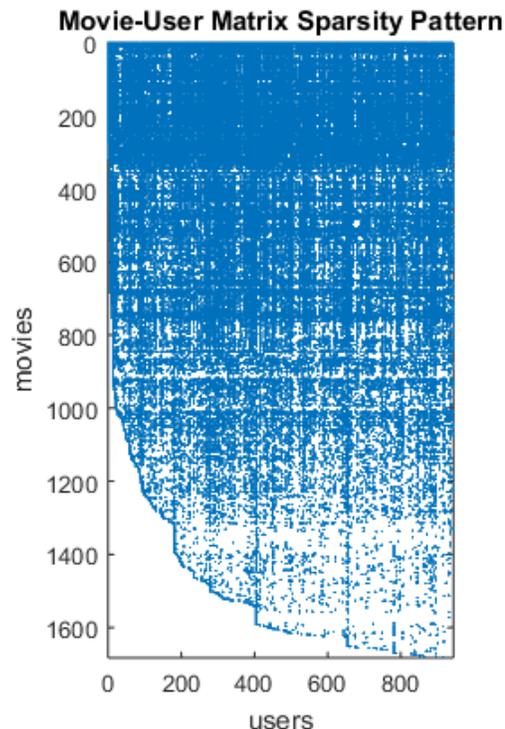
```
data_dir = 'ml-100k';

if exist(data_dir,'dir') ~= 7
    unzip('http://files.grouplens.org/datasets/movielens/ml-100k.zip')
end

data = readtable(fullfile(data_dir,'u.data'),'FileType','text',...
    'ReadVariableNames',false,'Format','%f%f%f%f');
data.Properties.VariableNames = {'user_id','movie_id','rating','timestamp'};

sp_mur = sparse(data.movie_id,data.user_id,data.rating);

[m,n] = size(sp_mur);
figure
spy(sp_mur)
title('Movie-User Matrix Sparsity Pattern')
xlabel('users')
ylabel('movies')
```



Older movies in lower row indices have fairly dense ratings but the newer movies with higher row indices are really sparse. You can also see that older

users in lower column indices don't rate newer movies at all – it seems they have stopped using the service.

You can take advantage of sparsity by computing directly on the sparse matrix. For example, you can pre-compute the Pearson correlation scores and mean centered ratings, and, once we have an active user, they can be used to compute the neighborhood and generate recommendations for that user.

It turns out you need to update the pre-computation of neighborhood and mean-centered ratings fairly frequently because most users don't rate many movies and one new rating can shift values a lot. For this reason, the item-based approach is used more commonly than user-based approach we studied earlier. Only big difference is that you compute similarity based on movies rather than users. It is just a matter of transposing the mean-centered matrix, but you need to update this less frequently because item-based scores are more stable.

```
meanRatings = sum(sp_mur,2)./sum(sp_mur~=0,2);  
[i,j,v] = find(sp_mur);  
meanCentered = sparse(i,j,v - meanRatings(i),m,n);  
sims = corr(meanCentered', 'rows', 'pairwise');
```

MATLAB on Hadoop and MATLAB Production Server

We can still do this in-memory with a 100K dataset, but for larger datasets, MATLAB can run MapReduce jobs with MATLAB Distributed Computing Server on Hadoop clusters for offline jobs. MATLAB Production Server enables rapid deployment of MATLAB code in Production environment and it may be a good choice for nearline or online uses where concurrent A/B testing is performed, because you avoid dual implementation and enable rapid system update based on the test result.

For an example of how you use MapReduce in MATLAB, check out "Scaling Market Basket Analysis with MapReduce" (<http://blogs.mathworks.com/loren/2015/02/25/scaling-market-basket-analysis-with-mapreduce/>). Incidentally, Collaborative Filtering and Association Rule Mining are related concepts. In Market Basket Analysis, we consider transactions as basic units. In Collaborative Filtering, we consider users as basic units. You may be able to repurpose the MapReduce code from that post with appropriate modifications.

To learn more about MATLAB capabilities, please consult the following resources.

MATLAB MapReduce and Hadoop (<http://www.mathworks.com/discovery/matlab-mapreduce-hadoop.html>)

MATLAB Production Server (<http://www.mathworks.com/products/matlab-production-server/>)

Data Analytics (<http://www.mathworks.com/solutions/data-analytics/>)

Closing

As noted earlier, Collaborative Filtering and Market Basket Analysis (<http://blogs.mathworks.com/loren/2015/01/29/introduction-to-market-basket-analysis/>) are closely related applications of data mining and machine learning. Both aim to learn from the customer behaviors, but Market Basket Analysis aims for high frequency transactions while Collaborative Filtering enables personalized recommendations. Naturally, you cannot restock your store shelves for individual shoppers, but you can in ecommerce. You can also think of Latent Semantic Analysis (<http://blogs.mathworks.com/loren/2015/04/08/can-you-find-love-through-text-analytics/>) as a related technique applied in text analytics domain.

Just as Market Basket Analysis found its way into web usage data mining, you can probably use Collaborative Filtering for the web data analysis. There may be applications in other fields as well. Any thoughts about your creative use of "Collaborative Filtering" with your data? Let us know here (<http://blogs.mathworks.com/loren/?p=1159#respond>).

Get the MATLAB code

Published with MATLAB® R2015a

Published with MATLAB® R2015a

These postings are the author's and don't necessarily represent the opinions of MathWorks.