Winning the Netflix Prize: A Summary

by Edwin Chen on Mon 24 October 2011

How was the Netflix Prize won? I went through a lot of the Netflix Prize papers a couple years ago, so I'll try to give an overview of the techniques that went into the winning solution here.

Normalization of Global Effects

Suppose Alice rates Inception 4 stars. We can think of this rating as composed of several parts:

- A baseline rating (e.g., maybe the mean over all user-movie ratings is 3.1 stars).
- An Alice-specific effect (e.g., maybe Alice tends to rate movies lower than the average user, so her ratings are -0.5 stars lower than we normally expect).
- An Inception-specific effect (e.g., Inception is a pretty awesome movie, so its ratings are 0.7 stars higher than we normally expect).
- A less predictable effect based on the specific interactionbetween Alice and Inception that accounts for the remainder of the stars (e.g., Alice really liked Inception because of its particular combination of Leonardo DiCaprio and neuroscience, so this rating gets an additional 0.7 stars).

In other words, we've decomposed the 4-star rating into: 4 = [3.1 (the baseline rating) - 0.5 (the Alice effect) + 0.7 (the Inception effect)] + 0.7 (the specific interaction)

So instead of having our models predict the 4-star rating itself, we could first try to remove the effect of the baseline predictors (the first three components) and have them predict the specific 0.7 stars. (I guess you can also think of this as a simple kind of boosting.)

More generally, additional baseline predictors include:

- A factor that allows Alice's rating to (linearly) depend on the (square root of the) number of days since her first rating. (For example, have you ever noticed that you become a harsher critic over time?)
- A factor that allows Alice's rating to depend on the number of days since the movie's first rating by anyone. (If you're one of the first people to watch it, maybe it's because you're a huge fan and really excited to see it on DVD, so you'll tend to rate it higher.)
- A factor that allows Alice's rating to depend on the number of people who have rated Inception. (Maybe Alice is a hipster who hates being part of the crowd.)
- A factor that allows Alice's rating to depend on the movie's overall rating.
- (Plus a bunch of others.)

And, in fact, modeling these biases turned out to be fairly important: in their paper describing their final solution to the Netflix Prize, Bell and Koren write that

> Of the numerous new algorithmic contributions, I would like to highlight one – those humble baseline predictors (or biases), which capture main effects in the data. While the literature mostly concentrates on the more sophisticated algorithmic aspects, we have learned that an accurate treatment of main effects is probably at least as signficant as coming up with modeling breakthroughs.

(For a perhaps more concrete example of why removing these biases is useful, suppose you know that Bob likes the same kinds of movies that Alice does. To predict Bob's rating of Inception, instead of simply predicting the same 4 stars that Alice rated, if we know that Bob tends to rate movies 0.3 stars higher than average, then we could first remove Alice's bias and then add in Bob's: $4 + 0.5 + 0.3 = 4.8$.)

Neighborhood Models

Let's now look at some slightly more sophisticated models. As alluded to in the section above, one of the standard approaches to collaborative filtering is to use neighborhood models.

Briefly, a neighborhood model works as follows. To predict Alice's rating of Titanic, you could do two things:

- Item-item approach: find a set of items similar to Titanic that Alice has also rated, and take the (weighted) mean of Alice's ratings on them.
- User-user approach: find a set of users similar to Alice who rated Titanic, and again take the mean of their ratings of Titanic.

(See also my post on item-to-item collaborative filtering on Amazon.)

The main questions, then, are (let's stick to the item-item approach for simplicity):

- How do we find the set of similar items?
- How do we weight these items when taking their mean?

The standard approach is to take some similarity metric (e.g., correlation or a Jaccard index) to define similarities between pairs of movies, take the K most similar movies under this metric (where K is perhaps chosen via cross-validation), and then use the same similarity metric when computing the weighted mean.

This has a couple problems:

- Neighbors aren't independent, so using a standard similarity metric to define a weighted mean overcounts information. For example, suppose you ask five friends where you should eat tonight. Three of them went to Mexico last week and are sick of burritos, so they strongly recommend against a taqueria. Thus, your friends' recommendations have a stronger bias than what you'd get if you asked five friends who didn't know each other at all. (Compare with the situation where all three Lord of the Rings Movies are neighbors of Harry Potter.)
- Different movies should perhaps be using different numbers of neighbors. Some movies may be predicted well by only one neighbor (e.g., Harry Potter 2 could be predicted well by Harry Potter 1 alone), some movies may require more, and some movies may have no good neighbors (so you should ignore your neighborhood algorithms entirely and let your other ratings models stand on their own).

So another approach is the following:

- You can still use a similarity metric like correlation or cosine similarity to choose the set of similar items.

- But instead of using the similarity metric to define the interpolation weights in the mean calculations, you essentially perform a (sparse) linear regression to find the weights that minimize the squared error between an item's rating and a linear combination of the ratings of its neighbors. Note that these weights are no longer constrained, so that if all neighbors are weak, then their weights will be close to zero and the neighborhood model will have a low effect.

(A slightly more complicated user-user approach, similar to this item-item neighborhood approach, is also useful.)

Implicit Data

Adding on to the neighborhood approach, we can also letimplicit data influence our predictions. The mere fact that a user rated lots of science fiction movies but no westerns, suggests that the user likes science fiction better than cowboys. So using a similar framework as in the neighborhood ratings model, we can learn for Inception a set of offset weightsassociated to Inception's movie neighbors.

Whenever we want to predict how Bob rates Inception, we look at whether Bob rated each of Inception's neighbors. If he did, we add in the corresponding offset; if not, then we add nothing (and, thus, Bob's rating is implicitly penalized by the missing weight).

Matrix Factorization

Complementing the neighborhood approach to collaborative filtering is the matrix factorization approach. Whereas the neighborhood approach takes a very local approach to ratings (if you liked Harry Potter 1, then you'll like Harry Potter 2!), the factorization approach takes a more global view (we know that you like fantasy movies and that Harry Potter has a strong fantasy element, so we think that you'll like Harry Potter) thatdecomposes users and movies into a set of latent factors(which we can think of as categories like "fantasy" or "violence").

In fact, matrix factorization methods were probably the most important class of techniques for winning the Netflix Prize. In their 2008 Progress Prize paper, Bell and Koren write

> It seems that models based on matrix-factorization were found to be most accurate (and thus popular), as evident by recent publications and discussions on the Netflix Prize forum. We definitely agree to that, and would like to add that those matrix-factorization models also offer the important flexibility needed for modeling temporal effects and the binary view. Nonetheless, neighborhood models, which have been dominating most of the collaborative filtering literature, are still expected to be popular due to their practical characteristics - being able to handle new users/ratings without re-training and offering direct explanations to the recommendations.

The typical way to perform matrix factorizations is to perform asingular value decomposition on the (sparse) ratings matrix (using stochastic gradient descent and regularizing the weights of the factors, possibly constraining the weights to be positive to get a type of non-negative matrix factorization). (Note that this "SVD" is a little different from the standard SVD learned in linear

algebra, since not every user has rated every movie and so the ratings matrix contains many missing elements that we don't want to simply treat as 0.)

Some SVD-inspired methods used in the Netflix Prize include:

- Standard SVD: Once you've represented users and movies as factor vectors, you can dot product Alice's vector with Inception's vector to get Alice's predicted rating of Inception.
- Asymmetric SVD: Instead of users having their own notion of factor vectors, we can represent users as a bag of items they have rated (or provided implicit feedback for). So Alice is now represented as a (possibly weighted) sum of the factor vectors of the items she has rated, and to get her predicted rating of Titanic, we can dot product this representation with the factor vector of Titanic. From a practical perspective, this model has an added benefit in that no user parameterizations are needed, so we can use this approach to generate recommendations as soon as a user provides some feedback (which could just be views or clicks on an item, and not necessarily ratings), without needing to retrain the model to factorize the user.
- SVD++: Incorporate both the standard SVD and the asymmetric SVD model by representing users both by their own factor representation and as a bag of item vectors.

Regression

Some regression models were also used in the predictions. The models are fairly standard, I think, so I won't spend too long here. Basically, just as with the neighborhood models, we can take a user-centric approach and a movie-centric approach to regression:

- User-centric approach: We learn a regression model for each user, using all the movies that the user rated as the dataset. The response is the movie's rating, and the predictor variables are attributes associated to that movie (which can be derived from, say, PCA, MDS, or an SVD).
- Movie-centric approach: Similarly, we can learn a regression model for each movie, using all the users that rated the movie as the dataset.

Restricted Boltzmann Machines

Restricted Boltzmann Machines provide another kind of latent factor approach that can be used. See this paper for a description of how to apply them to the Netflix Prize. (In case the paper's a little difficult to read, I wrote an introduction to RBMs a little while ago.)

Temporal Effects

Many of the models incorporate temporal effects. For example, when describing the baseline predictors above, we used a few temporal predictors that allowed a user's rating to (linearly) depend on the time since the first rating he ever made and on the time since a movie's first rating. We can also get more fine-grained temporal effects by, say, binning items into a couple months' worth of ratings at a time, and allowing movie biases to change within each bin. (For example, maybe in May 2006, Time Magazine nominated Titanic as the best movie ever made, which caused a spurt in glowing ratings around that time.)

In the matrix factorization approach, user factors were also allowed to be time-dependent (e.g., maybe Bob comes to like comedy movies more and more over time). We can also give more weight to recent user actions.

Regularization

Regularization was also applied throughout pretty much all the models learned, to prevent overfitting on the dataset. Ridge regression was heavily used in the factorization models to penalize large weights, and lasso regression (though less effective) was useful as well. Many other parameters (e.g., the baseline predictors, similarity weights and interpolation weights in the neighborhood models) were also estimated using fairly standard shrinkage techniques.

Ensemble Methods

Finally, let's talk about how all of these different algorithms were combined to provide a single rating that exploits the strengths of each model. (Note that, as mentioned above, many of these models were not trained on the raw ratings data directly, but rather on the residuals of other models.)

In the paper detailing their final solution, the winners describe using gradient boosted decision trees to combine over 500 models; previous solutions used instead a linear regression to combine the predictors.

Briefly, gradient boosted decision trees work by sequentially fitting a series of decision trees to the data; each tree is asked to predict the error made by the previous trees, and is often trained on slightly perturbed versions of the data. (For a longer description of a similar technique, see my introduction to random forests.)

Since GBDTs have a built-in ability to apply different methods to different slices of the data, we can add in some predictors that help the trees make useful clusterings:

- Number of movies each user rated
- Number of users that rated each movie
- Factor vectors of users and movies
- Hidden units of a restricted Boltzmann Machine

(For example, one thing that Bell and Koren found (when using an earlier ensemble method) was that RBMs are more useful when the movie or the user has a low number of ratings, and that matrix factorization methods are more useful when the movie or user has a high number of ratings.)

Here's a graph of the effect of ensemble size from early on in the competition (in 2007), and the authors' take on it:

However, we would like to stress that it is not necessary to have such a large number of models to do well. The plot below shows RMSE as a function of the number of methods used. One can achieve our winning score (RMSE=0.8712) with less than 50 methods, using the best 3 methods can yield RMSE < 0.8800, which would land in the top 10. Even just using our single best method puts us on the leaderboard with an RMSE of 0.8890. The lesson here is that having lots of models is useful for the incremental results needed to win competitions, but practically, excellent systems can be built with just a few well-selected models.