# Recommending Netflix Movies

## Summary

The announcement and release of data for the "Netflix Challenge" in 2006 changed the way recommender systems research was done. Until this point research datasets of this size were simply not publicly available. The challenge itself also spawned a host of new ideas in recommender system design.

One idea that arose from the competition and had a lot of impact was to build factor models using SVD to create a low-rank approximation from which estimated ratings could be extracted. This in itself was not new, but the size of the data stopped this being practical until an alternative way of computing SVD was introduced by a blogger known as *Simon Funk* having a go at the challenge for himself. This new approach could not only work with a large amount of data in a time-efficient manner, but could also explicitly deal with the missing values.

By 2010, we had learned another thing from the data released for the Netflix Challenge; even though it had been anonymised before release, through the use of some relatively simple data mining techniques, researchers demonstrated that it was highly probable that specific individuals could be identified.

## Key points

### The Netflix Challenge

- Big data set released to the public
    - for training over 100M quadruplets
        - from just under 500K users and 18K movies
    - almost 3M triplets for performing testing
        - Online system would compute performance (using RMSE) of predicted ratings against the (hidden) actual ratings
- Lots of teams took part in the competition
    - Led to a lot of new approaches to recommendation
- Key findings:
    - RMSE is not a good success measure
        - Doesn't reflect user satisfaction
    - Time matters
        - Just because I liked something in the past doesn't mean I would like it now
    - Matrix Factorisation can be very powerful
        - SVD-like solutions played a very big part...
    - One method is not enough
        - Teams that did well blended predictions across models (i.e. used ensemble regression methods)
    - There are potentially better ways to improve recommendation
        - The Netflix competition data is both noisy and constrained; improving data quality and adding features could be way more powerful than attempting to find better models

### Latent Factor Models for Recommendation

- Performing LSA on the user-movie ratings matrix would seem like a natural approach to building a

recommender system
        - Noisy ratings would be smoothed-out
        - The model would be constrained to have fewer degrees of freedom
        - The concepts will represent different categories of movies (or users)
            - although won't necessarily be understandable as such
- Just need to compute a rank-k SVD of the original ratings matrix, then reconstruct the estimate and read out the predicted ratings...
    - Two problems:
        - The ratings matrix might be very big (in the Netflix data it would have over 8 Billion elements)
            - Standard SVD solvers will have problems with that!
        - The ratings matrix is sparse because of the unknowns
            - We don't really want to treat these as zeros
    - Solution proposed by Simon Funk (aka Brandyn Webb)
        - Just solve it directly using stochastic gradient decent (SGD)
            - Turns out to be a rather simple and eloquent solution
                - See the Appendix for the derivation
                - It's approximate in the sense that if there is missing data it isn't really and SVD; if every value in the input ratings matrix was know it is exactly equivalent to SVD
            - Also allows for easy addition of regularisers, e.g. to penalise the magnitude of predicted ratings (bear in mind original scale of 1-5)
                - Potentially also allows for non-linearities in the model

### Epilogue: The end of the Netflix Challenge

- At the end of the competition, two key methodologies stood-out, although it was blends of models using these approaches that actually won:
    - SVD/factorisation based latent factor models
    - Restricted Boltzmann Machine models
- The overall winning solution was never actually implemented and deployed at Netflix
    - too computationally expensive at scale
    - expected improvements not worth the engineering effort
        - there were better ways to make money!
- The death of the dataset:
    - Before they originally released the data set, Netflix went to some lengths to anonymise the data
        - Users only represented by sequential IDs
        - Only personal information about a user ID was the ratings of movies associated with it + the dates those ratings were created
            - Some amount of random noise was added to the ratings and dates
    - But, even with all this anonymisation it was possible to deanonymise some of the data and identify specific users, literally by comparing against public ratings on IMDB

## Further Reading

- Wikipedia has a good overview of how the Netflix prize played out:
  https://en.wikipedia.org/wiki/Netflix_Prize (https://en.wikipedia.org/wiki/Netflix_Prize)
  http://technocalifornia.blogspot.co.uk/2009/09/netflix-prize-lessons-learned.html
  (http://technocalifornia.blogspot.co.uk/2009/09/netflix-prize-lessons-learned.html)

- This blog post (http://technocalifornia.blogspot.co.uk/2009/09/netflix-prize-lessons-learned.html) has a good overview of the key things the challenge taught us.
- Simon Funk's Blog posts on using Matrix Factorisation on the Netflix Challenge data:
    - http://sifter.org/~simon/journal/20061027.2.html (http://sifter.org/~simon/journal/20061027.2.html)
    - http://sifter.org/~simon/journal/20061102.1.html (http://sifter.org/~simon/journal/20061102.1.html)
    - http://sifter.org/~simon/journal/20061211.html (http://sifter.org/~simon/journal/20061211.html)
    - http://sifter.org/~simon/journal/20070815.html (http://sifter.org/~simon/journal/20070815.html)
    - http://sifter.org/~simon/journal/20070817.html (http://sifter.org/~simon/journal/20070817.html)
- Papers by the winners with details of their approaches (three teams combined into one for the final prize, but they each wrote their own reflections):
    - http://www.netflixprize.com/assets/GrandPrize2009_BPC_BellKor.pdf (http://www.netflixprize.com/assets/GrandPrize2009_BPC_BellKor.pdf)
    - http://www.netflixprize.com/assets/GrandPrize2009_BPC_BigChaos.pdf (http://www.netflixprize.com/assets/GrandPrize2009_BPC_BigChaos.pdf)
    - http://www.netflixprize.com/assets/GrandPrize2009_BPC_PragmaticTheory.pdf (http://www.netflixprize.com/assets/GrandPrize2009_BPC_PragmaticTheory.pdf)
- To find out more about De-anonymisation of the dataset, read this:
    - Arvind Narayanan and Vitaly Shmatikov. (2007) Robust De-anonymization of Large Datasets (How to Break Anonymity of the Netflix Prize Dataset). http://arxiv.org/pdf/cs/0610105v2.pdf (http://arxiv.org/pdf/cs/0610105v2.pdf)

## Appendix 1: Derivation of computation of SVD via SGD

Assume we want to perform the decomposition $\mathbf{R} \approx \mathbf{UF}$ for a user-movie rating matrix $\mathbf{R}$ of size $m \times n$, $\mathbf{U}$ of size $m \times c$ and $\mathbf{F}$ of size $c \times n$, with $c << rank(\mathbf{R})$, such that $\|\mathbf{R} - \mathbf{UF}\|_{\mathrm{F}}$ is minimised.

The predicted rating, $p_{ij}$, for user $i$ and movie $j$ is:

$$p_{ij} = \sum_k \mathbf{U}_{ik}\mathbf{F}_{kj}$$

The squared error, $E_{ij}^2$ in the prediction is simply:

$$E_{ij}^2 = (\mathbf{R}_{ij} - p_{ij})^2 = (\mathbf{R}_{ij} - \sum_k \mathbf{U}_{ik}\mathbf{F}_{kj})^2$$

To compute the update rules for SGD we need the partial derivatives of the squared error w.r.t. the parameters we are estimating. Lets start by computing the partial derivatives for the error for one user $i = I$, one movie $j = J$ and one singular vector $k = K$:

$$\frac{\delta E^2}{\delta \mathbf{U}_{IK}} = 2E\frac{\delta E}{\delta \mathbf{U}_{IK}} = 2E(-\mathbf{F}_{KJ}) = -2(\mathbf{R}_{ij} - p_{ij})\mathbf{F}_{KJ}$$

Note: $\delta E / \delta \mathbf{U}_{IK}$ is the derivative of a constant ($\mathbf{R}_{IJ}$) minus a sum in which only one of the summands is a function of $\mathbf{U}_{IK}$, namely the term $\mathbf{U}_{IK}\mathbf{F}_{KJ}$. Clearly this sum differentiates to $\mathbf{F}_{JK}$ and the derivatives of all the other terms are zero.

We can follow the same procedure to compute the partial derivative with respect to $\mathbf{F}_{KJ}$:

$$\frac{\delta E^2}{\delta \mathbf{F}_{KJ}} = 2E\frac{\delta E}{\delta \mathbf{F}_{KJ}} = -2(\mathbf{R}_{ij} - p_{ij})\mathbf{U}_{IK}$$

SGD can be used to update estimates of $\mathbf{U}$ and $\mathbf{F}$ for a given rating $\mathbf{R}_{IJ}$ from user $I$ and movie $J$ for a given singular vector $K$ as follows:

$$\mathbf{U}_{IK} := \mathbf{U}_{IK} - \alpha \frac{\delta E^2}{\delta \mathbf{U}_{IK}} = \mathbf{U}_{IK} + 2\alpha(\mathbf{R}_{ij} - p_{ij})\mathbf{F}_{KJ}$$

$$\mathbf{F}_{KJ} := \mathbf{F}_{KJ} - \alpha \frac{\delta E^2}{\delta \mathbf{F}_{KJ}} = \mathbf{F}_{KJ} + 2\alpha(\mathbf{R}_{ij} - p_{ij})\mathbf{U}_{IK}$$

To use this in practice you would start by training the first singular vector using these update rules over multiple epochs (passes over the entire set of user-movie-rating triplets) and only move on to the next singular vector when the current vector has converged. If all the ratings in the original matrix $\mathbf{R}$ are known, then this decomposition is exactly equivalent to SVD; in the case of a sparse $\mathbf{R}$ with many unknowns, then this is an *approximate SVD*.