# How Does netflix recommend movies

Sarika Padmashali

10/04/2016

# Recommendation Engine

- Recommendation systems seek to predict the "rating" or "preference" that a user would give to an item.
- Content – based filtering recommends items based on a comparison between the content of the items and a user profile.
- Collaborative filtering filters information by using the recommendations of other people.

# Netflix Input

- Input - the history of star ratings across all the users and all the movies.
- Each data point consists of four numbers: (1) user ID- *u*, (2) movie title- *i*, (3) number of stars, 1–5, in the rating, denoted as $r_{ui}$, and (4) date of the rating, denoted as $t_{ui}$.
- Large dataset but only a fraction of users will have watched a given movie.
- The output is, a set of predictions , one for each movie *i* that user *u* has not watched yet. These can be real numbers, not just integers like an actual rating $r_{ui}$ The final output is a short, rank-ordered list of movies recommended to each user *u*, presumably those movies receiving ≥ 4 or the top five movies with the highest predicted .

# Root Mean Squared Error

- The real test of this mind-reading system is whether user *u* actually likes the recommended movies.
- The **Root Mean Squared Error** (RMSE), measured for those (*u, i*) pairs for which we have both the prediction and the actual rating. Let us say there are *C* such pairs.

$$\mathrm{RMSE} = \sqrt{\sum_{(u,i)} \frac{(r_{ui} - \hat{r}_{ui})^2}{C}}$$

- Smaller the RMSE better the recommender model

# Baseline Predictor Models

- Take the average of all the ratings $\bar{r}$ and use that as the predictor for all $\{\hat{r}_{ui}\}$.
- *Lets incorporate two more parameters say $b_i$* to model the quality of each movie *i* relative to the average and $b_u$ to model the bias of each user *u* relative to $\bar{r}$.
- Model of baseline predictor would look like:

$$\hat{r}_{ui} = \bar{r} + b_u + b_i.$$

- Where $b_u = (\sum_i r_{ui}/M_u) - \bar{r}$

$$b_i = (\sum_u r_{ui}/M_i) - \bar{r}$$

$M_u$ is the number of movies rated by user *u*,

$M_i$ is the number of users who rated movie *i*

# Neighborhood model

$$\hat{r}_{ui}^{N}$$

$$d_{ij} = \frac{\mathbf{r}_i^T \mathbf{r}_j}{\|\mathbf{r}_i\|_2 \|\mathbf{r}_j\|_2} = \frac{\sum_u \tilde{r}_{ui} \tilde{r}_{uj}}{\sqrt{\sum_u (\tilde{r}_{ui})^2 \sum_u (\tilde{r}_{uj})^2}}$$

$$\hat{r}_{ui}^{N} = (\bar{r} + b_u + b_i) + \frac{\sum_{j \in \mathcal{L}_i} d_{ij} \tilde{r}_{uj}}{\sum_{j \in \mathcal{L}_i} |d_{ij}|}$$

# Summary of the algorithm

- Train the baseline predictor by solving the least squares problem.
- Obtain the baseline predictor matrix
- Compute the movie- movie similarity matrix
- Pick a neighborhood size L to compute neighborhood movies L  for each movie i
- Compute sum of baseline predictor and the neighborhood predictor

# Regularization

- Learning is both an exercise of *hindsight* and one of *foresight*
- Perfect hindsight often means you are simply re-creating history
- Robust learning without overfitting
- Regularization is a common one: add a penalty term that reduces the sensitivity to model parameters by rewarding smaller parameters

$$\text{minimize}_{\{\text{model parameters}\}} \text{(Squared error term)} + \lambda \text{ (Parameter size squared)}.$$

# Latent Factor Model

- The latent-factor method relies on *global structures* underlying the table.
- One of the challenges in recommendation system design is that the table is both *large* and *sparse*
- We suspect there may be structures that can be captured by two, much smaller matrices.
- Build a low-dimensional model for these high-dimensional data.
- $K$-dimensional vector $\mathbf{p}_u$ to explain each user $u$'s movie taste. And for each movie $i$, we use a $K$-dimensional vector $\mathbf{q}_i$ explaining the movie's appeal. The inner product between these two vectors,  is the prediction .

# THE END