Evaluating Recommenders

Matthew Effect

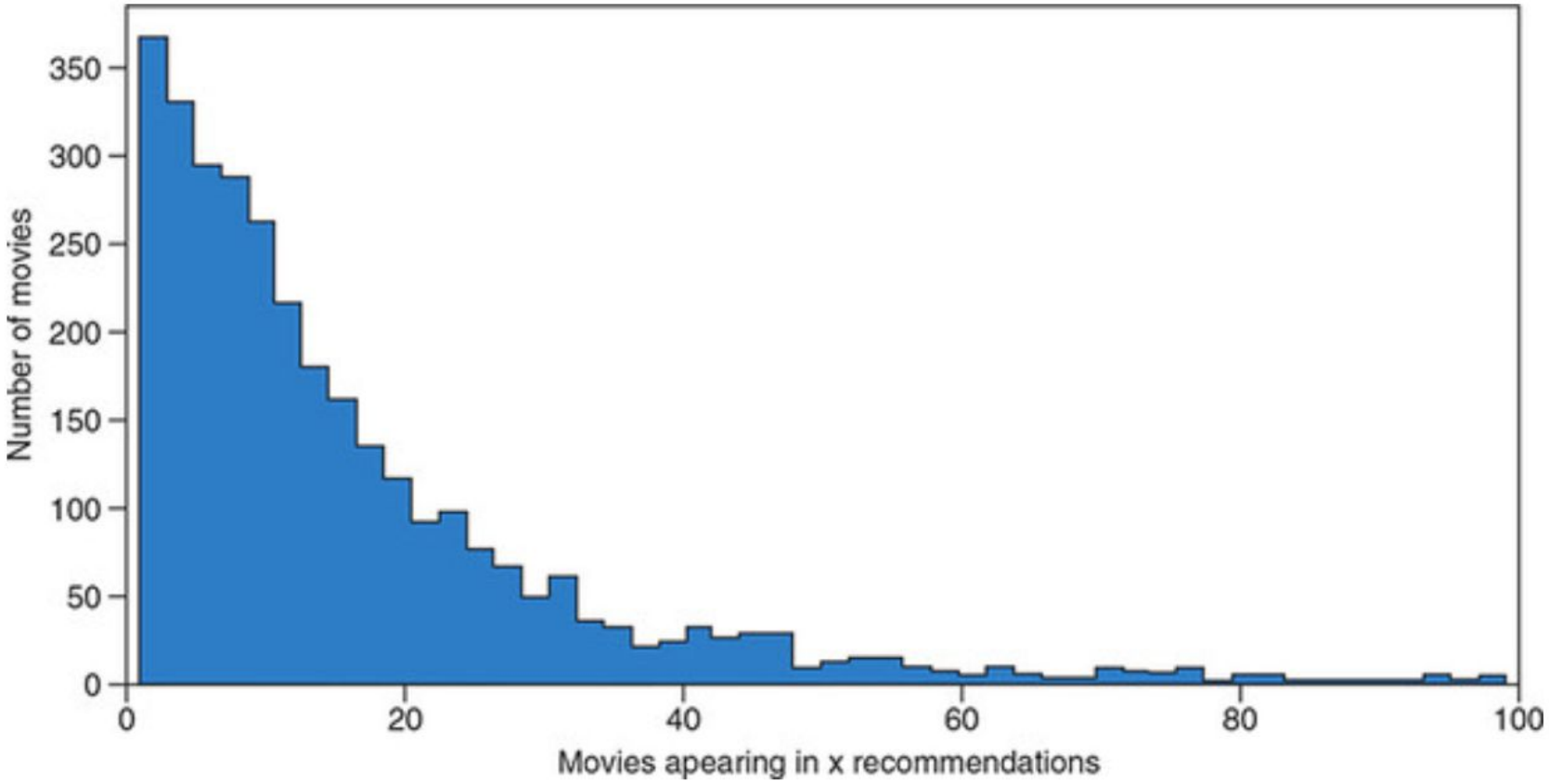
For unto everyone that hath shall be given, and he shall have abundance: but from him that hath not shall be taken even that which he hath.

Diversity

- Rich will get richer and the poor get poorer
- Popular items get recommended often
- Filter bubble
- Hard to measure diversity
- Attempt to calculate diversity by calculating the average dissimilarity between all pairs of recommended items– *Matthew Effect*

Coverage

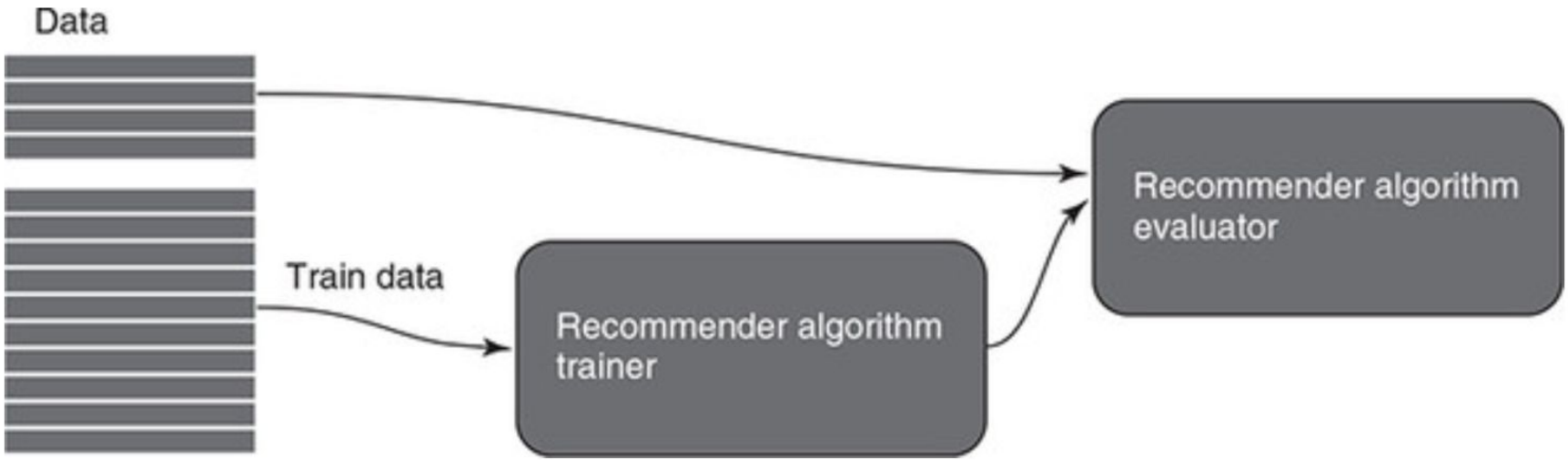
- Content coverage - Coverage of the catalog
- User coverage - Recommend assets to all users
  - Iterate over all users, call the recommender and see if it returns anything


Number of movies
350
300
250
200
150
100
50
0
0
20
40
60
80
100
Movies apearing in x recommendations

Serendipity

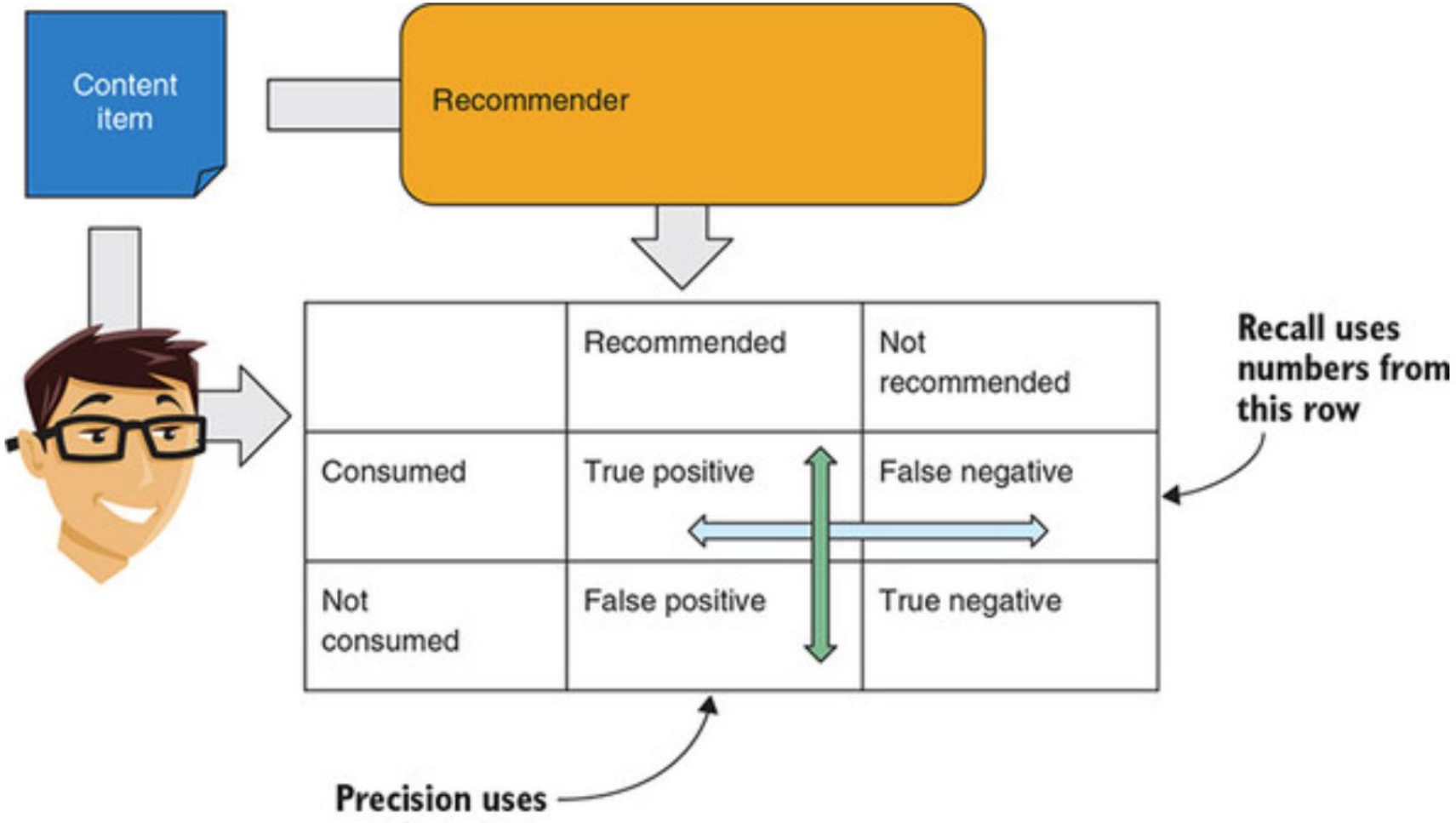
- Surprising the user
- Less constraint on the algorithm
- Difficult to measure

Offline Evaluation



Error Metrics

- RMSE
- MAE
- If items have a long tail then divide the data into a set with popular items and evaluate that, then separately evaluate the other data set with long-tail items


Content item
Recommender
Recommended
Not recommended
Consumed
True positive
False negative
Not consumed
False positive
True negative
Recall uses numbers from this row
Precision uses

- ***Precision***— What fraction of the recommended items the user consumed:

$$Precision = \frac{True\ Positive}{True\ Positive + False\ Positive}$$

- ***Recall***— What, out of all the items that the user consumed, was recommended:

$$Recall = \frac{True\ Positive}{True\ Positive + False\ Negative}$$

Rank Aware Metrics

- We are interested in having good recommendations at the top but not care so much about what happens further down
- It’s often considered more important to optimize precision, while not putting too much importance on whether the user gets all the possible relevant items (the recall)

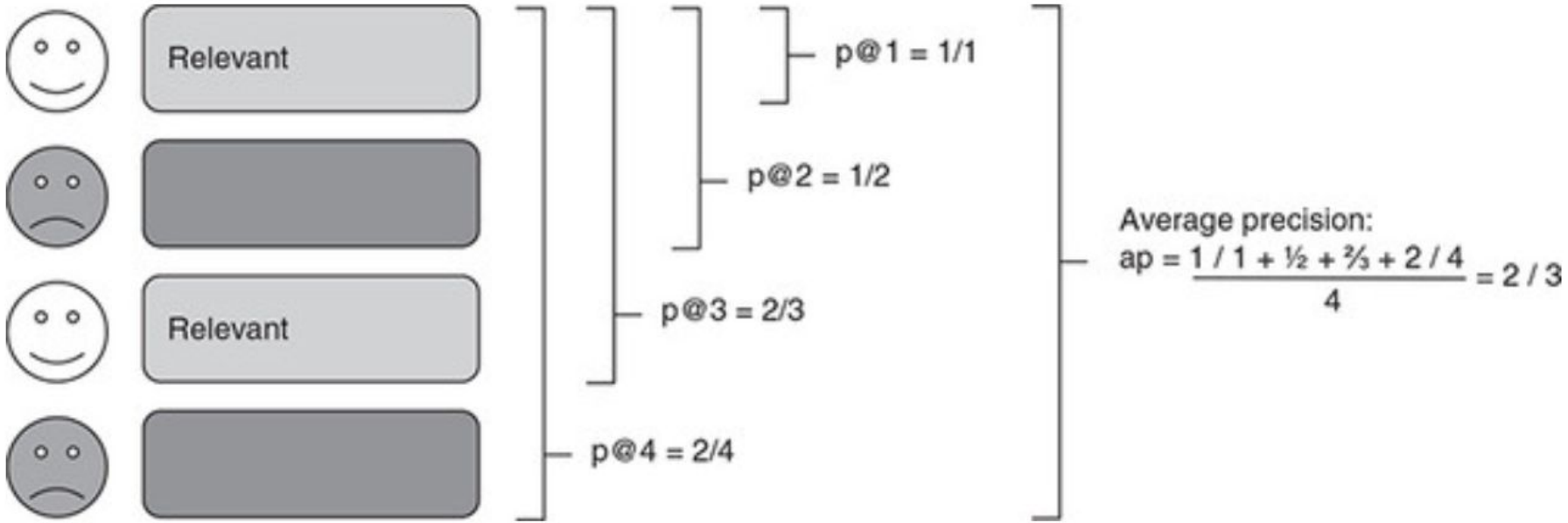
Precision at K

- Calculate average over all users and dividing by the number of users

$$P@k(u) = \frac{\#\{\textit{relevant content in the top k postitions}\}}{k}$$

Mean Average Precision (MAP)

- Take the mean of the average over all recommendations



Discounted Cumulative Gain

- It is finding each relevant item and then penalizing the items the farther down it is in the list

Splitting the Dataset

Random Splitting

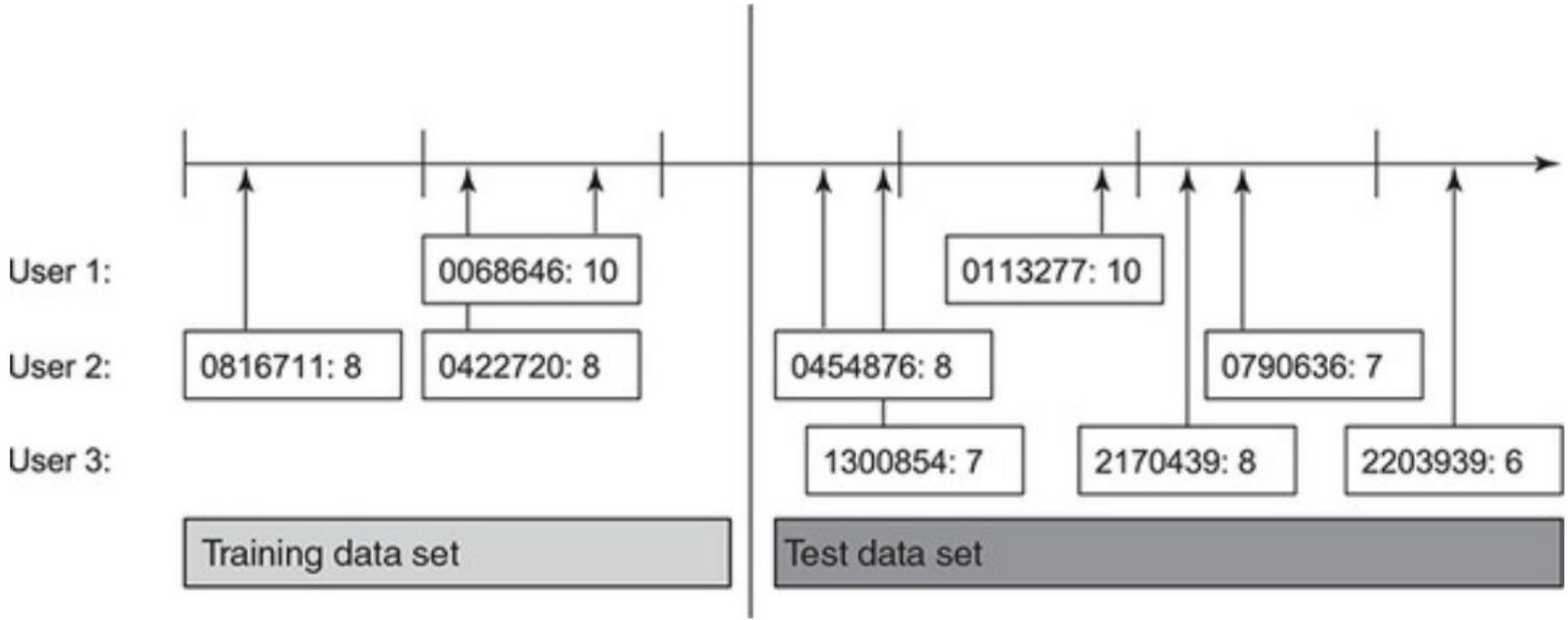
Random splitting

Recommenders will be trained with ratings that are added after the ones that it needs to predict

User in validation set but not seen in Train set

Item in validation set but not in the Training set

Split by Time



Split by Time

- Users who only appear in the test set
- Clean the test data set for users who don’t appear in the training set

Split by Users

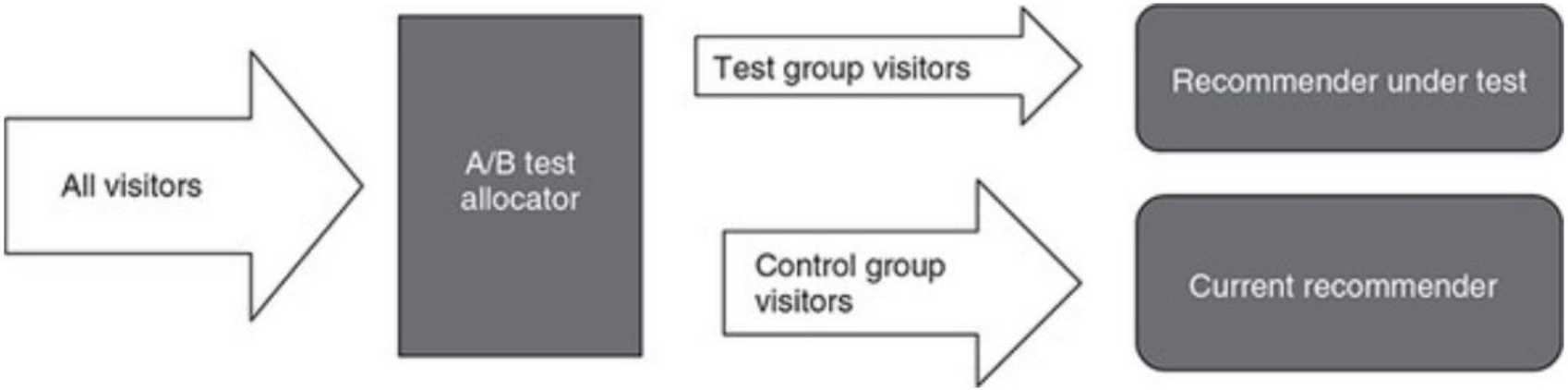
Divide each user’s ratings between a training and test set

This way all users will be in the training set and we won't find any users in the test set that haven’t been seen before

More demanding to split data this way

We need to order data by timestamps

Online testing



Summary

- You should consider testing before implementing a recommender system.
- Regression tests guard your code against mistakes added unknowingly.
- Serendipity, the users finding things in your recommendations that they love but never knew they would, is hard to measure, but it’s important.
- Different metrics are used to calculate whether the recommender is good or, at least, if it compares to a baseline.

Summary

- A/B testing (where visitors are split into two groups) is something you need to consider if you want to fine-tune your recommender system. An A/B test can also be done to test which parameters work better; for example, the size of the neighborhood in collaborative filtering or the number of latent factors in matrix factorization.
- Watch out for feedback loops.