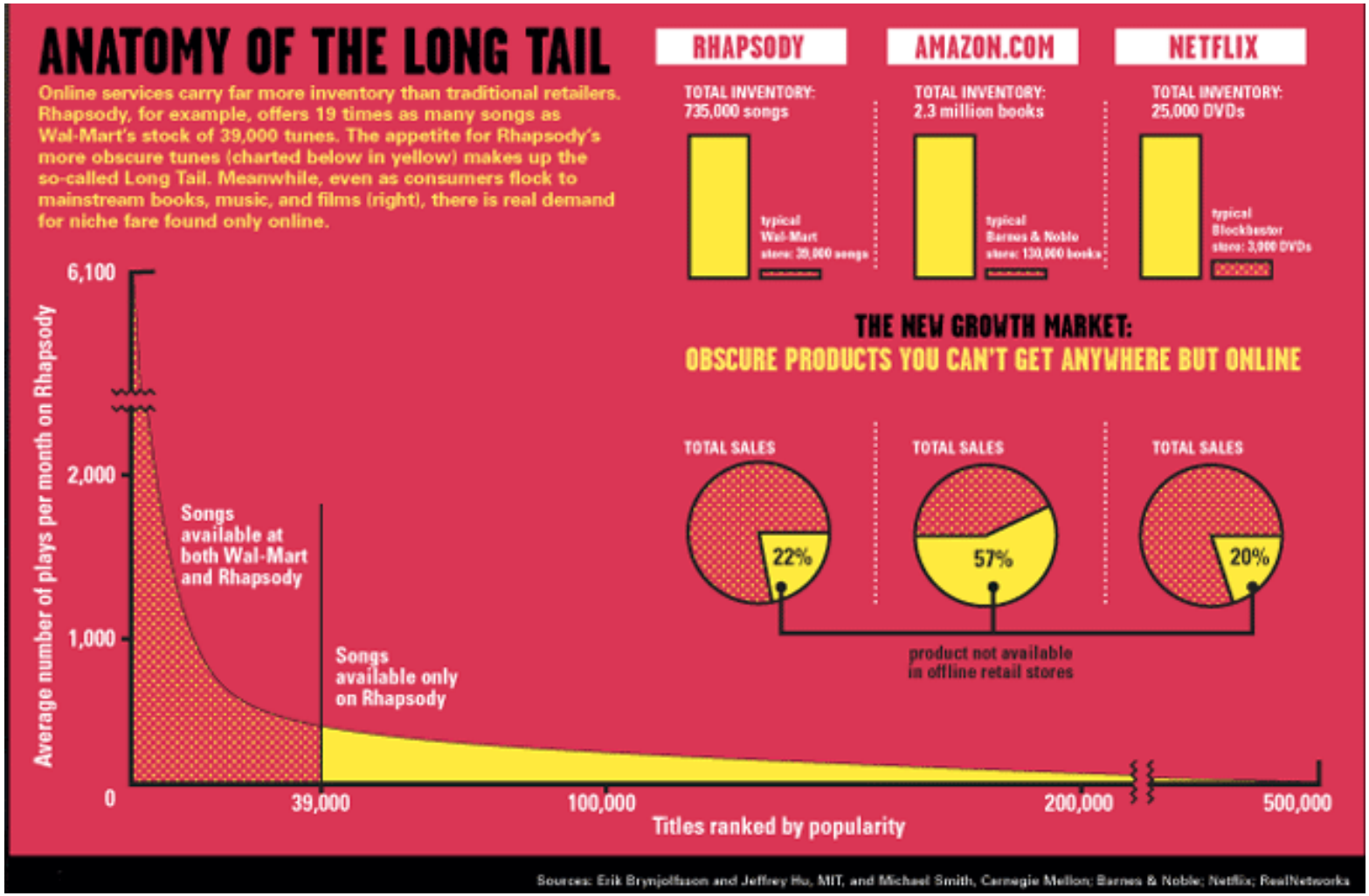
Recommender Systems

Lecture 6

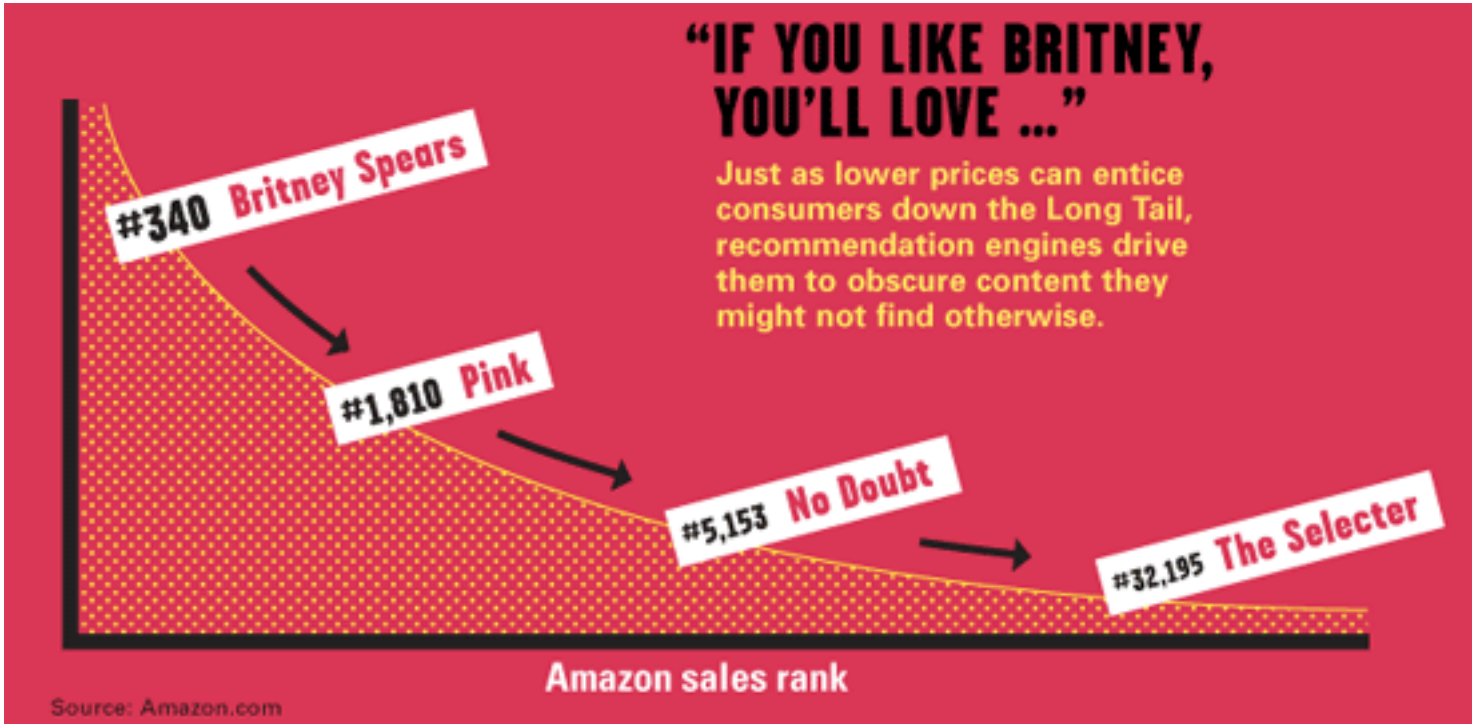
Outline

1. Problem Representation
   a) Issues: source, bias, long tail
   b) General approaches
      - Cold start
2. Content Filtering
3. Collaborative Filtering
   a) Neighborhood
   b) Matrix Factorization
4. Netflix Challenge
5. Optional Reading

The Long Tail (Wired)



The Long Tail (Wired)



The Recommendation Problem

- Given some subset of…
  - a single user's activity/ratings
  - information about users/items
  - ratings provided by other users

- Predict ratings for unrated items

Example Applications

- Movies
  - Netflix
- Songs
  - Pandora
- Products
  - Amazon
- Web Search
  - Google
- New Friends/Colleagues/Dates
  - Facebook/LinkedIn/OKCupid

…

User Activity

	Item 1	Item 2	Item 3	Item 4
Properties	🍎	🍏	📱	☎️
User A	☺			

Can we predict unknown user preferences based upon past ratings for a set of products?

Utility Matrix

	Item 1	Item 2	Item 3	Item 4
User A	☺	☺		
User B			☺	
User C	☺	☺		☺
User D	☺			

Can we predict unknown user preferences based upon other users' preferences?

Utility Matrix (Numeric)

	Item 1	Item 2	Item 3	Item 4
User A	5	1		
User B			4	
User C	4	5		3
User D	3			

Can we predict unknown user preferences based upon other users' preferences?
Jaccard distance gets us back (i.e. rated or not)

Utility Matrix (Positive/Negative)

	Item 1	Item 2	Item 3	Item 4
User A	☺	☺		
User B			☺	
User C	☹	☺		☺
User D	☹			

FYI: some approaches may have to be adjusted in the presence of negative ratings

Utility Matrix + User/Item Properties

		Item 1	Item 2	Item 3	Item 4
Properties		🍎	🍏	📱	☎️
User A	👨🏽🇺🇸	☺	☺		
User B	👩🏾🇮🇳			☺	
User C	👨🏽🇨🇳	☺	☺		☺
User D	👩🏼🇺🇸	☺			

Content properties can be used for recommendation;
often in combination with preferences
(particularly useful to overcome **cold start** issues)

Where Do Ratings Come From?

- Users may be directly asked for ratings
  - Examples: stars, likes, 👍/👎
  - Depending upon scale/context, may introduce bias; only available for those willing to supply, representative?
- Ratings may be implicitly derived via behaviors
  - Examples: view count/duration, product/link clicks
  - Comes with its own set of challenges (e.g. watching = like/dislike?)
- Typically quite sparse

  # ratings / (# users * # items) << 100%

General Approaches

- Content-based Filtering
  - Model user/item features
- Collaborative Filtering
  - Extract patterns from the utility matrix
  - **Our focus**

Again: not necessarily exclusive

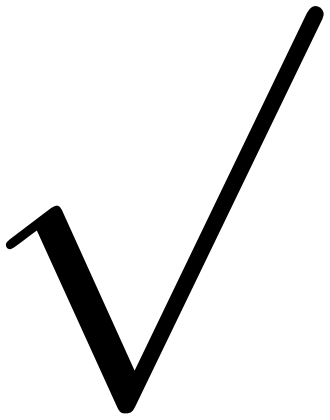
Evaluation

- Typically Root Mean Square Error (RMSE)
  - Others possible (e.g. MSE)

- Unless external/validation ratings supplied, operates on known ratings only

Evaluation

- Typically **Root** Mean Square Error (RMSE)



Evaluation

- Typically **Root Mean** Square Error (RMSE)

$$\sqrt{\frac{1}{|U| \cdot |I|} \sum_{u \in U, i \in I}}$$

Evaluation

- Typically **Root Mean Square Error** (RMSE)

$$\sqrt{\frac{1}{|U| \cdot |I|} \sum_{u \in U, i \in I} (\hat{r}_{ui} - r_{ui})^2}$$

Unless external data supplied, should split into train/test for validation (r_{ui})

Challenges

- Scalability
  - Typically # users >> # items, but not always
    - Netflix: 109M users, ~7K movies/shows
    - Amazon: 304M accounts, ~20K movies/shows
      - About 400M products sold
- Dynamics
  - Changing users, inventory, market
- Reviews
  - Power law
  - Not random

Content-Based Filtering

- Assume we only have available for a particular user: past ratings, per-item features (e.g. from profile)
  - Possibilities: TF-IDF (text), tags, user input
- Then best we can do is learn per-user classifier/regression model (per problem representation)
  - Rating values: OLS, …
  - Discrete: logistic (up/down), tree/NB (class), …

Example (Rating)

Movie	User 1	x_1	x_2
Love at Last	1	0.9	0.0
Romance Forever	**?**	1.0	0.1
Cute Puppies		0.9	0.0
Fast Forever!	4	0.1	1.0
Swords v Karate	5	0.0	0.9

Example (Like)

Movie	User 1	x_1	x_2
Love at Last	☹	0.9	0.0
Romance Forever	**?**	1.0	0.1
Cute Puppies		0.9	0.0
Fast Forever!	☺	0.1	1.0
Swords v Karate	☺	0.0	0.9

Sources of Bias

- Some movies are universally loved/hated
  - Source: external (e.g. IMDB), internal avg
- Some users are more picky than others
  - Source: running internal avg
- So for rating modeling via OLS…

$$\hat{r}_{ui} = b_u + b_i + x_i^\intercal w_u$$

Collaborative Filtering

- Takes as input the utility matrix, fills in unknown ratings

- Main classes of approaches
  - Neighborhood: user-user, item-item
  - Matrix Factorization
  - Others: clustering, graphs, …

Neighborhood-Based Methods

User-User

- Find top-k most similar users w.r.t. known ratings
  - Common: Pearson corr.
- Compute aggregated rating from their ratings
  - Such as average, weighted w.r.t. relative distance
- Better: few, rapidly changing items

Item-Item

- Find top-k most similar items w.r.t. known ratings
  - Common: Cosine distance
- Compute aggregated rating from own ratings
  - Such as average, weighted w.r.t. relative distance
- Better: few, rapidly changing users

Common Issues

- Bias
  - Typical to normalize w.r.t. average user/item
    - User/User: subtract each user's mean before
    - Item/Item: subtract each item's mean before
- Scaling
  - Compute similarity matrix in batch (e.g. nightly)
  - Data structures to index into neighborhood

Matrix Factorization

- Basic idea: summarize the correlations across rows and columns in the form of lower dimensional vectors
  - One per user (U_i), one per item (I_j)
  - Use these to impute missing ratings ($r_{ij}=U_i \cdot I_j$)

- If we impose consistent dimensionality, we can combine the vectors and thus …
  - $[r_{ij}]_{n \times d} = F_{user} F_{item}$
  - $F_{user} = n \times k$
  - $F_{item} = k \times d$

Factorization Approaches: SVD

- Decompose, choose *k* singular values
  - $[r_{ij}]_{n \times d} \approx Q_{n \times k} \Sigma_{k \times k} P_{d \times k}^T$
  - $F_{user} = Q\Sigma$
  - $F_{item} = P$

- Some issues…
  - Cannot apply to incomplete matrix, so have to provide some default value
    - Commonly 0, mean (e.g. per user)
  - Relatively high computational complexity

Factorization Approaches: Direct

- Pose directly as an optimization problem
  - Idea: difference between specified entries in the original ratings matrix and the projected matrix should be small

- Assume: $R = [r_{ij}]_{n \times d}$ $U = F_{user}$ $V = F_{item}$

$$J = ||R - UV||^2$$

- Frobenius norm: $||A||_F = \sqrt{\sum_{i=1}^{m} \sum_{j=1}^{n} |a_{ij}|^2}$

Factorization Regularization

To reduce overfitting, particularly when specified entries is small…

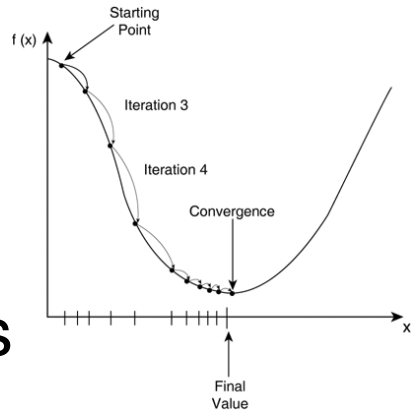
$$\arg\min_{U,V} ||R - UV||^2 + \lambda(||U||^2 + ||V||^2)$$

where λ is determined empirically (e.g. cross-validation)

Common Factorization Approaches

- Alternating Least Squares
  - Similar to EM: hold U fixed, optimize V; flip
  - Turns into OLS, and each row is independent
    - So parallelization!!!
  - Will only improve at each step, so converge to local minimum

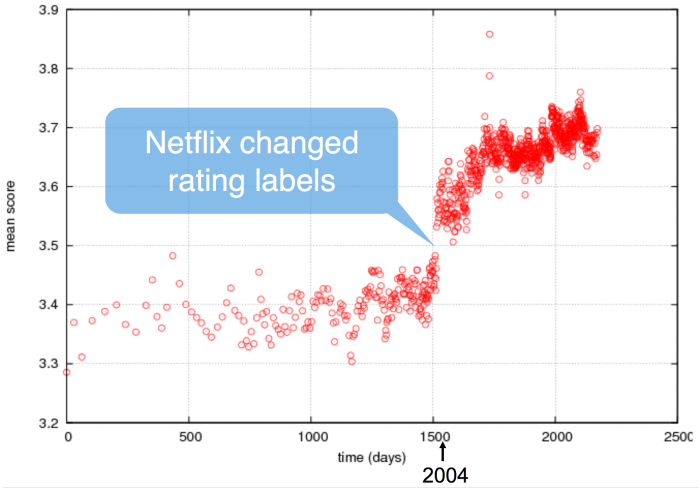
- Stochastic Gradient Descent
  - Converges to local minimum
  - Many parameters, so problematic for large ratings matrices



Addressing Bias

- Common to pre-normalize ratings

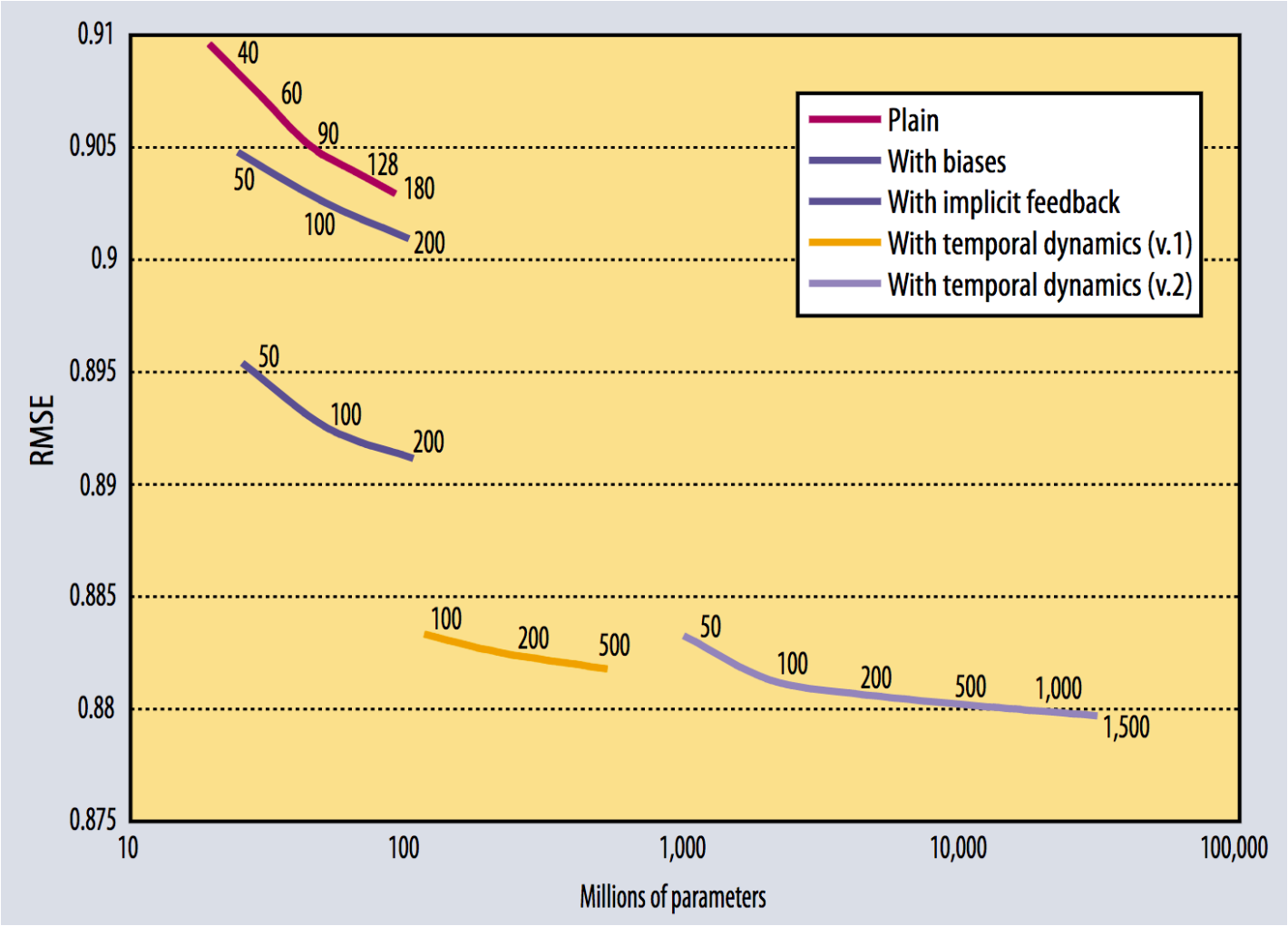
- Can add user/item biases explicitly
  - Can also model temporal effects



Netflix Competition (2006)

- Training: 100M ratings
  - 500K customers, 17K movies
  - 1-5 stars
- Test: ~3M (RMSE)
  - $1M to 10% improvement over Netflix
    - Awarded in 2009
  - $50K to first place each year until finish

Initial Progress: MF



Down to the Buzzer

- On June 26, 2009 the team "BellKor's Pragmatic Chaos", a merger of teams "Bellkor in BigChaos" and "Pragmatic Theory", achieved a 10.05% improvement over Cinematch (a Quiz RMSE of 0.8558)
- On July 25, 2009 the team "The Ensemble", a merger of the teams "Grand Prize Team" and "Opera Solutions and Vandelay United", achieved a 10.09% improvement over Cinematch (a Quiz RMSE of 0.8554)
- At the end, BellKor was deemed the winner for having submitted 20 minutes earlier :)

Final Leaderboard

NETFLIX

Netflix Prize COMPLETED

Home | Rules | Leaderboard | Update

Leaderboard

Showing Test Score. Click here to show quiz score

Rank	Team Name	Best Test Score	% Improvement	Best Submit Time
Grand Prize - RMSE = 0.8567 - Winning Team: BellKor's Pragmatic Chaos				
1	BellKor's Pragmatic Chaos	0.8567	10.06	2009-07-26 18:18:28
2	The Ensemble	0.8567	10.06	2009-07-26 18:38:22
3	Grand Prize Team	0.8582	9.90	2009-07-10 21:24:40
4	Opera Solutions and Vandelay United	0.8588	9.84	2009-07-10 01:12:31
5	Vandelay Industries !	0.8591	9.81	2009-07-10 00:32:20
6	PragmaticTheory	0.8594	9.77	2009-06-24 12:06:56
7	BellKor in BigChaos	0.8601	9.70	2009-05-13 08:14:09
8	Dace	0.8612	9.59	2009-07-24 17:18:43
9	Feeds2	0.8622	9.48	2009-07-12 13:11:51
10	BigChaos	0.8623	9.47	2009-04-07 12:33:59
11	Opera Solutions	0.8623	9.47	2009-07-24 00:34:07
12	BellKor	0.8624	9.46	2009-07-26 17:19:11

The Spoils


NETFLIX
2009
DATE 09.21.09
PAY TO THE ORDER OF BellKor's Pragmatic Chaos
$ 1,000,000 00
AMOUNT ONE MILLION
00/100
FOR The Netflix Prize
Reed Hastings
The Netflix Prize

Optional Reading (1)

COVER FEATURE

MATRIX FACTORIZATION TECHNIQUES FOR RECOMMENDER SYSTEMS

Yehuda Koren, *Yahoo Research*
Robert Bell and Chris Volinsky, *AT&T Labs—Research*

As the Netflix Prize competition has demonstrated, matrix factorization models are superior to classic nearest-neighbor techniques for producing product recommendations, allowing the incorporation of additional information such as implicit feedback, temporal effects, and confidence levels.

Modern consumers are inundated with choices. Electronic retailers and content providers offer a huge selection of products, with unprecedented opportunities to meet a variety of special needs and tastes. Matching consumers with the most appropriate products is key to enhancing user satisfaction and loyalty. Therefore, more retailers have become interested in recommender systems, which analyze patterns of user interest in products to provide personalized recommendations that suit a user's taste. Because good personalized recommendations can add another dimension to the user experience, e-commerce leaders like Amazon.com and Netflix have made recommender systems a salient part of their websites.

Such systems are particularly useful for entertainment products such as movies, music, and TV shows. Many customers will view the same movie, and each customer is likely to view numerous different movies. Customers have proven willing to indicate their level of satisfaction with particular movies, so a huge volume of data is available about which movies appeal to which customers. Companies can analyze this data to recommend movies to particular customers.

RECOMMENDER SYSTEM STRATEGIES

Broadly speaking, recommender systems are based on one of two strategies. The *content filtering* approach creates a profile for each user or product to characterize its nature. For example, a movie profile could include attributes regarding its genre, the participating actors, its box office popularity, and so forth. User profiles might include demographic information or answers provided on a suitable questionnaire. The profiles allow programs to associate users with matching products. Of course, content-based strategies require gathering external information that might not be available or easy to collect.

A known successful realization of content filtering is the Music Genome Project, which is used for the Internet radio service Pandora.com. A trained music analyst scores

42 COMPUTER Published by the IEEE Computer Society 0018-9162/09/$26.00 © 2009 IEEE

Optional Reading (2)

Factorization Meets the Neighborhood: a Multifaceted Collaborative Filtering Model

Yehuda Koren
AT&T Labs – Research
180 Park Ave, Florham Park, NJ 07932
yehuda@research.att.com

ABSTRACT

Recommender systems provide users with personalized suggestions for products or services. These systems often rely on Collaborating Filtering (CF), where past transactions are analyzed in order to establish connections between users and products. The two more successful approaches to CF are latent factor models, which directly profile both users and products, and neighborhood models, which analyze similarities between products or users. In this work we introduce some innovations to both approaches. The factor and neighborhood models can now be smoothly merged, thereby building a more accurate combined model. Further accuracy improvements are achieved by extending the models to exploit both explicit and implicit feedback by the users. The methods are tested on the Netflix data. Results are better than those previously published on that dataset. In addition, we suggest a new evaluation metric, which highlights the differences among methods, based on their performance at a top-K recommendation task.

Categories and Subject Descriptors

H.2.8 [**Database Management**]: Database Applications—*Data Mining*

General Terms

Algorithms

Keywords

collaborative filtering, recommender systems

1. INTRODUCTION

Modern consumers are inundated with choices. Electronic retailers and content providers offer a huge selection of products, with unprecedented opportunities to meet a variety of special needs and tastes. Matching consumers with most appropriate products is not trivial, yet it is a key in enhancing user satisfaction and loyalty. This emphasizes the prominence of *recommender systems*, which provide personalized recommendations for products that suit a user's taste [1]. Internet leaders like Amazon, Google, Netflix, TiVo and Yahoo are increasingly adopting such recommenders.

Recommender systems are often based on *Collaborative Filtering* (CF) [10], which relies only on past user behavior—e.g., their previous transactions or product ratings—and does not require the creation of explicit profiles. Notably, CF techniques require no domain knowledge and avoid the need for extensive data collection. In addition, relying directly on user behavior allows uncovering complex and unexpected patterns that would be difficult or impossible to profile using known data attributes. As a consequence, CF attracted much of attention in the past decade, resulting in significant progress and being adopted by some successful commercial systems, including Amazon [15], TiVo and Netflix.

In order to establish recommendations, CF systems need to compare fundamentally different objects: items against users. There are two primary approaches to facilitate such a comparison, which constitute the two main disciplines of CF: *the neighborhood approach* and *latent factor models*.

Neighborhood methods are centered on computing the relationships between items or, alternatively, between users. An item-oriented approach evaluates the preference of a user to an item based on ratings of similar items by the same user. In a sense, these methods transform users to the item space by viewing them as baskets of rated items. This way, we no longer need to compare users to items, but rather directly relate items to items.

Latent factor models, such as Singular Value Decomposition (SVD), comprise an alternative approach by transforming both items and users to the same latent factor space, thus making them directly comparable. The latent space tries to explain ratings by characterizing both products and users on factors automatically inferred from user feedback. For example, when the products are movies, factors might measure obvious dimensions such as comedy vs. drama, amount of action, or orientation to children; less well defined dimensions such as depth of character development or "quirkiness"; or completely uninterpretable dimensions.

The CF field has enjoyed a surge of interest since October 2006, when the Netflix Prize competition [5] commenced. Netflix released a dataset containing 100 million movie ratings and challenged the research community to develop algorithms that could beat the accuracy of its recommendation system, Cinematch. A lesson that we learnt through this competition is that the neighborhood and latent factor approaches address quite different levels of structure in the data, so none of them is optimal on its own [3].

Neighborhood models are most effective at detecting very localized relationships. They rely on a few significant neighborhood-relations, often ignoring the vast majority of ratings by a user. Consequently, these methods are unable to capture the totality of weak signals encompassed in all of a user's ratings. Latent factor models are generally effective at estimating overall structure that relates simultaneously to most or all items. However, these models are poor at detecting strong associations among a small set of closely related items, precisely where neighborhood models do best.

In this work we suggest a combined model that improves predic-

KDD'08, August 24–27, 2008, Las Vegas, Nevada, USA.

Optional Reading (3)

Evaluating Recommendation Systems

Guy Shani and Asela Gunawardana

Abstract *Recommender systems* are now popular both commercially and in the research community, where many approaches have been suggested for providing recommendations. In many cases a system designer that wishes to employ a recommendation system must choose between a set of candidate approaches. A first step towards selecting an appropriate algorithm is to decide which properties of the application to focus upon when making this choice. Indeed, recommendation systems have a variety of properties that may affect user experience, such as accuracy, robustness, scalability, and so forth. In this paper we discuss how to compare recommenders based on a set of properties that are relevant for the application. We focus on comparative studies, where a few algorithms are compared using some evaluation metric, rather than absolute benchmarking of algorithms. We describe experimental settings appropriate for making choices between algorithms. We review three types of experiments, starting with an offline setting, where recommendation approaches are compared without user interaction, then reviewing user studies, where a small group of subjects experiment with the system and report on the experience, and finally describe large scale online experiments, where real user populations interact with the system. In each of these cases we describe types of questions that can be answered, and suggest protocols for experimentation. We also discuss how to draw trustworthy conclusions from the conducted experiments. We then review a large set of properties, and explain how to evaluate systems given relevant properties. We also survey a large set of evaluation metrics in the context of the property that they evaluate.

Guy Shani
Microsoft Research, One Microsoft Way, Redmond, WA, e-mail: guyshani@microsoft.com

Asela Gunawardana
Microsoft Research, One Microsoft Way, Redmond, WA, e-mail: aselag@microsoft.com