Web Mining

Web Usage Mining and Recommender Systems – Part 2 –

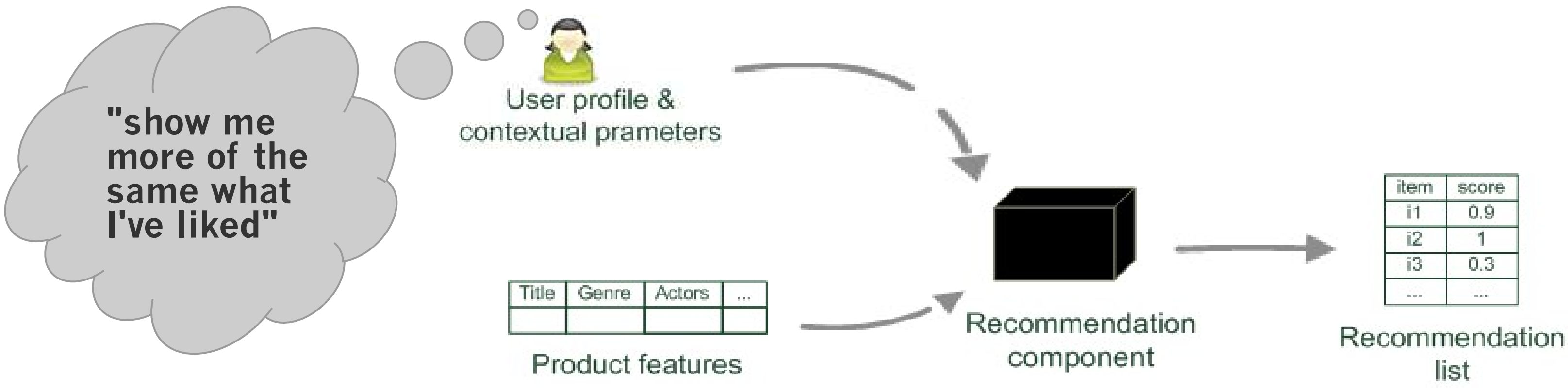
Prof. Dr. Christian Bizer

FSS 2023

Today’s Menu

4.2 Content-based Recommendation

- **While collaborative filtering methods do not use any information about the items, it might be reasonable to exploit such information.**
  - e.g., recommend fantasy novels to people who liked fantasy novels in the past
- **What do we need?**
  - information about the available items (content)
  - some sort of user profile describing what the user likes (user preferences)
- **The tasks:**
  1. learn user preferences from what she has bought/seen before
  2. recommend items that are "similar" to the user preferences



Structured Content and User Profile Representation

Content Representation: Item description

Title	Genre	Author	Type	Price	Keywords
The Night of the Gun	Memoir	David Carr	Paperback	29.90	Press and journalism, personal memoirs, detective, New York
The Lace Reader	Fiction, Mystery	Brunonia Barry	Hardcover	49.90	American contemporary fiction, detective, historical
Into the Fire	Romance, Suspense	Suzanne Brockmann	Hardcover	45.90	American fiction, murder, neo-nazism

User Profile: Summarizes seen items

Title	Genres	Authors	Types	Avg. Price	Keywords
…	Fiction. Mystery	Brunonia, Barry, Ken Follett	Paperback	25.65	Detective, murder, New York

Simple recommendation approach

- Compute the similarity of an unseen item with the user profile based on keyword overlap (e.g. using Dice)

$$sim = \frac{2 \times |keywords(b_i) \cap keywords(u)|}{|keywords(b_i)| + |keywords(u)|}$$

Keywords for *i*-th book

Keywords for user *u*

More sophisticated approach

- include other attributes: Genre, Author, Type

See: Web Data Integration: Identity Resolution

Textual Content and User Profile Representation

- Content-based recommendation techniques are often applied to recommend text documents, like news articles or blog posts.
- Documents and user profiles can be represented as term-vectors containing, for example, term frequencies:

Content Representation

	Doc 1	Doc 2	Doc 3
Antony	157	73	0
Brutus	4	157	0
Caesar	232	227	0
Calpurnia	0	10	123
Cleopatra	17	0	52
mercy	1	0	43

User Profile

	Liked Doc X1	Liked Doc X2	Liked Doc X3
Antony	0	1	0
Brutus	2	2	0
Caesar	4	3	0
Calpurnia	233	99	132
Cleopatra	57	12	42
mercy	22	23	34

Similarity of Text Documents

- **Challenges**
  - terms vectors are very sparse
  - not every word has the same importance
  - long documents have higher chance to overlap with user profile
  - semantic similarity of words might be relevant

- **Methods for handling these challenges**
  - similarity metric: cosine similarity, as it ignores M_{00}
  - preprocessing: remove stop words
  - vector creation:
    - Term-Frequency - Inverse Document Frequency ($TF - IDF$)
    - use word embeddings instead of one-hot-encoded term vectors
  - combined feature creation and similarity calculation :
    - Transformer-based methods (e.g. Sentence BERT)

See: Data Mining I: Text Mining

Recap: The TF-IDF Term Weighting Scheme

- **The TF-IDF weight (term frequency–inverse document frequency) is used to evaluate how important a word is to a corpus of documents.**
  - TF: Term Frequency (frequency/length doc)
  - IDF: Inverse Document Frequency.

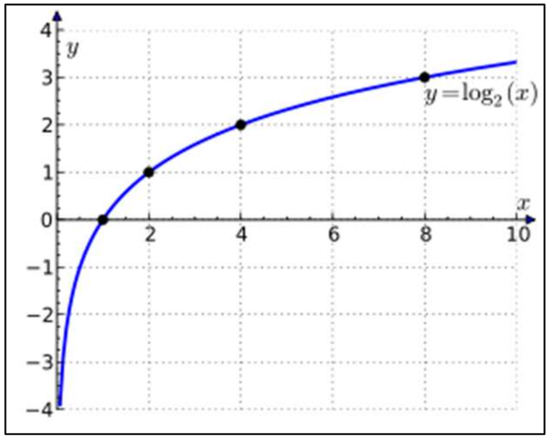
  N: total number of docs in corpus

  df_i: the number of docs in which t_i appears

$$w_{ij} = tf_{ij} \times idf_i.$$

$$idf_i = \log \frac{N}{df_i}$$

- **Gives more weight to rare words**
- **Give less weight to common words (domain-specific stopwords)**



Recap: Cosine Similarity and TF-IDF

- **Sample document set**

  d1 = “Saturn is the gas planet with rings.”

  d2 = “Jupiter is the largest gas planet.”

  d3 = “Saturn is the Roman god of sowing.”

Saturn	is
1/7 * log(3/2)	1/7*log(3/3)

- **Documents as TF-IDF vectors**

	Saturn	is	the	gas	planet	with	rings	Jupiter	largest	Roman	god	of	sowing
d1	0.03	0	0	0.03	0.03	0.07	0.07	0	0	0	0	0	0
d2	0	0	0	0.03	0.03	0	0	0.08	0.08	0	0	0	0
d3	0.03	0	0	0	0	0	0	0	0	0.07	0.07	0.07	0.07

- **Cosine similarities between the documents**
  - cos(d1,d2) = 0.13
  - cos(d1,d3) = 0.05
  - cos(d2,d3) = 0.00

$$\cos(d_1, d_2) = \frac{d_1 \bullet d_2}{\| d_1 \| \| d_2 \|}$$

Recommending Documents

- **Given a set of documents D already rated by the user**
  - either explicitly via user interface
  - or implicitly by monitoring user behavior

1. **Find the k nearest neighbors of a not-yet-seen item i in D**
   - measure similarity of item i with neighbors using cosine similarity
2. **Use ratings from Alice for neighbors k to predict a rating for item i**
   - weight Alice ratings by the similarity of the neighbors to item i

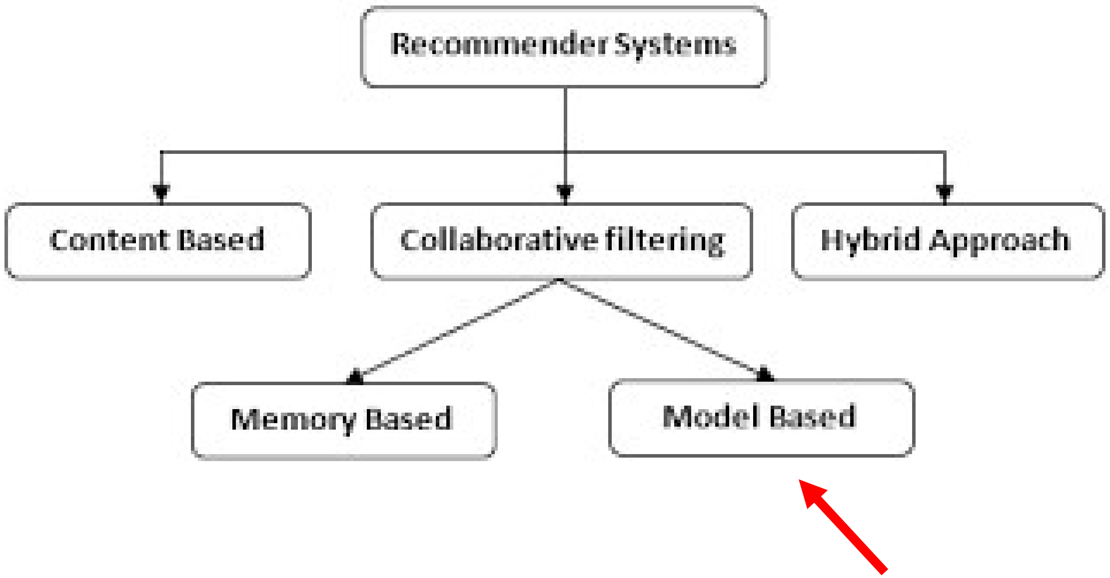
- **Variations:**
  - use similarity threshold instead of neighborhood size k
  - use upper similarity threshold to prevent system from recommending too similar texts (variations of texts the user has already seen)

Content-based Filtering Discussion

- **Pros:**
  - in contrast to collaborative approaches, content-based techniques do not require a user community
  - no problems with recommending new items (cold-start-problem)
- **Cons:**
  - Require to learn a suitable model of user's preferences based on explicit or implicit feedback
    - ramp-up phase required for new users (users needs to view/rate some items)
  - Overspecialization
    - algorithms tend to propose "more of the same"
    - recommendations might be boring as items are too similar

4.3 Model-based Collaborative Filtering

- **Key idea: Learn a model from training data "offline" and apply it "online" to compute ratings and perform recommendations.**
  - requires less online computation than memory-based KNN approaches



- **Last week**
  - Item-based Collaborative Filtering (model = pre-calculated similarities to set of neighbors)
- **This week**
  1. Probabilistic Recommendation using Naïve Bayes
  2. Recommendation using Matrix Factorization

1.1 Probabilistic Recommendation using Naïve Bayes

- **Basic idea:**
  - given the user/item rating matrix
  - determine the probability that Alice will give item i a specific rating
  - do this for all rating values and select the one with the highest probability

$$f(x) = \arg\max_{C_i \in C} P(C_i | X)$$

- The conditional probability $P(C_i \mid X)$, where

  C_1 = "Item5=1", C_2 = "Item5=2", C_3 = "Item5=3", …

  X = Alice's previous ratings (Item1=1, Item2=3, Item3= …)
- can be estimated using Bayes' theorem and the independence assumption

Probability of evidence given the class

Class prior without evidence

See: Data Mining I: Classification 3

Posterior probability

$$P(C_i \mid X) = \frac{P(X \mid C_i)P(C_i)}{P(X)}$$

Probability of evidence

Class Conditional Independence Assumption

Given the class label, the values of the features are treated as conditionally independent of one another:

Probability of seeing the evidence together with C_i

Independence assumption

$$P(X \mid C_i) = \prod_{k=1}^{n} P(x_k \mid C_i)$$

See: Data Mining I: Classification 3

Effect: We can estimate all probabilities from the training examples.

$$P(x_k \mid C_i) = \frac{s_{ik}}{s_i}$$

where s_{ik} is the number of training examples in C_i having the value x_k and s_i is the total number of training examples in C_i

Class Prior

$$P(C_i) = \frac{s_i}{s}$$

where s_i is the number of training examples in C_i and s is the total number of training examples.

Applying Naïve Bayes for Recommendation

	Item1	Item2	Item3	Item4	Item5
Alice	1	3	3	2	?
User1	2	4	2	2	4
User2	1	3	3	5	1
User3	4	5	2	3	2
User4	1	1	5	2	1

X = (Item1 =1, Item2=3, Item3=3, Item3=2)

$$P(C_i \mid X) = \frac{P(X \mid C_i)P(C_i)}{P(X)}$$

$$P(X|Item5 = 1)$$
$$= P(Item1 = 1|Item5 = 1) \times P(Item2 = 3|Item5 = 1) \times P(Item3 = 3|Item5 = 1)$$
$$\times P(Item4 = 2|Item5 = 1) = \frac{2}{2} \times \frac{1}{2} \times \frac{1}{2} \times \frac{1}{2} \approx 0.125$$

$$P(Item5 = 1) = \frac{2}{4} = 0.5$$ ← Class Prior

$$P(Item5 = 1|X) = \frac{P(X|Item5 = 1)P(Item5 = 1)}{P(X)} = \frac{(\frac{2}{2} \times \frac{1}{2} \times \frac{1}{2} \times \frac{1}{2}) \times \frac{2}{4}}{P(X)} == \frac{0.0625}{P(X)}$$

Applying Naïve Bayes for Recommendation

	Item1	Item2	Item3	Item4	Item5
Alice	1	3	3	2	?
User1	2	4	2	2	4
User2	1	3	3	5	1
User3	4	5	2	3	2
User4	1	1	5	2	1

$$P(Item5 = 1|X) = \frac{0.0625}{P(X)}$$

$$P(Item5 = 2|X) = \frac{\ldots}{P(X)}$$

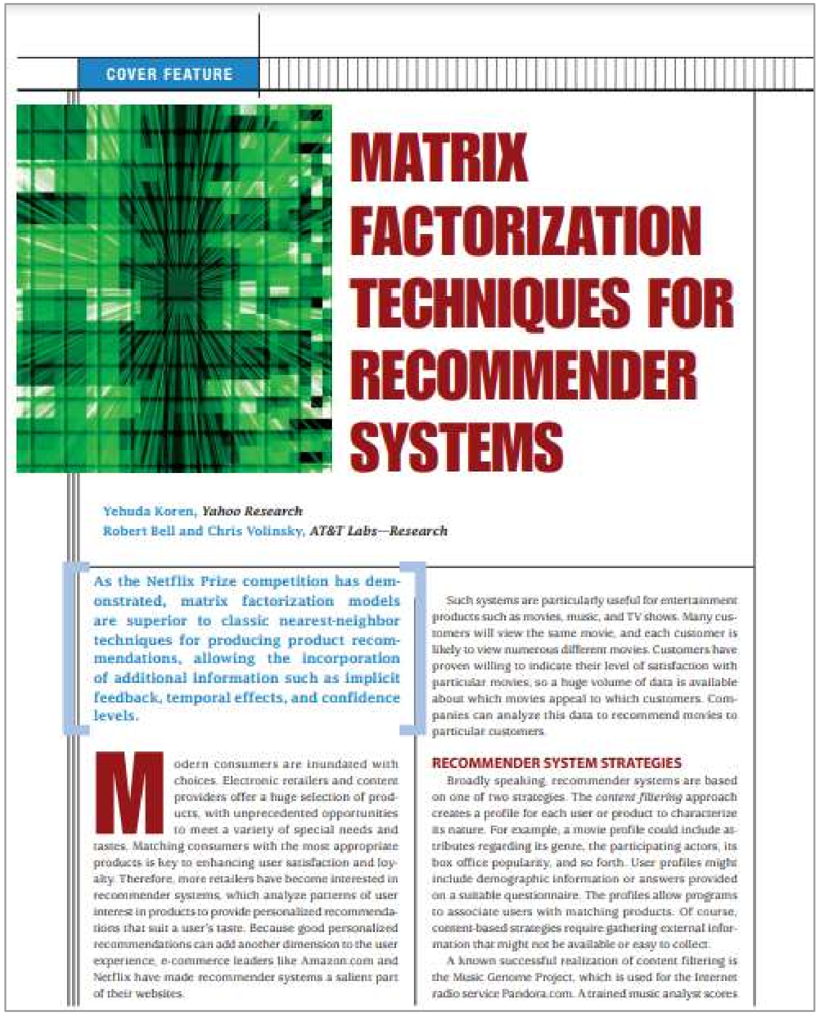
...

$$\hat{r} = \underset{r \in \{1,2,3,4,5\}}{\arg\max} \; P(\text{Item5} = r|X)$$

- Going through all calculations, we see that P(Item5=1|X) is higher than all other probabilities, which means the classifier will predict the rating of 1 for Item5 for the user Alice.
- **Discussion**
  - empirical analysis shows that probabilistic methods often lead to good results
  - small memory-footprint of leaned model as only the probabilities need to be stored
  - fast calculation of predictions at runtime (online)

1.2 Recommendation using Matrix Factorization

- **popularized in the context of the Netflix Challenge 2009**
- **Netflix Movie Dataset**
  - 100 million ratings that 500,000 users gave to 17,000 movies
- **Grand Prize of 1 Million $ won by team from Yahoo and AT&T**
  - beating Netflix's neighborhood-based method by 10%
  - using matrix factorization extended with modelling of biases and temporal dynamics



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

Y. Koren, R. Bell, C. Volinsky: Matrix Factorization Techniques for Recommender Systems.
In *Computer*, vol. 42, no. 8, pp. 30-37, Aug. 2009.

Latent Factor Models

- **Item characteristics and user preferences are represented as numerical factor values in the same space**
  - some latent factors are human understandable, others are not
  - amount of latent factors f is set as hyperparameter
  - 40 to 1500 factors were used by Netflix Challenge winners
- **Ratings $\hat{r}_{ui}$ are estimated as the dot product of the user and item factor values**

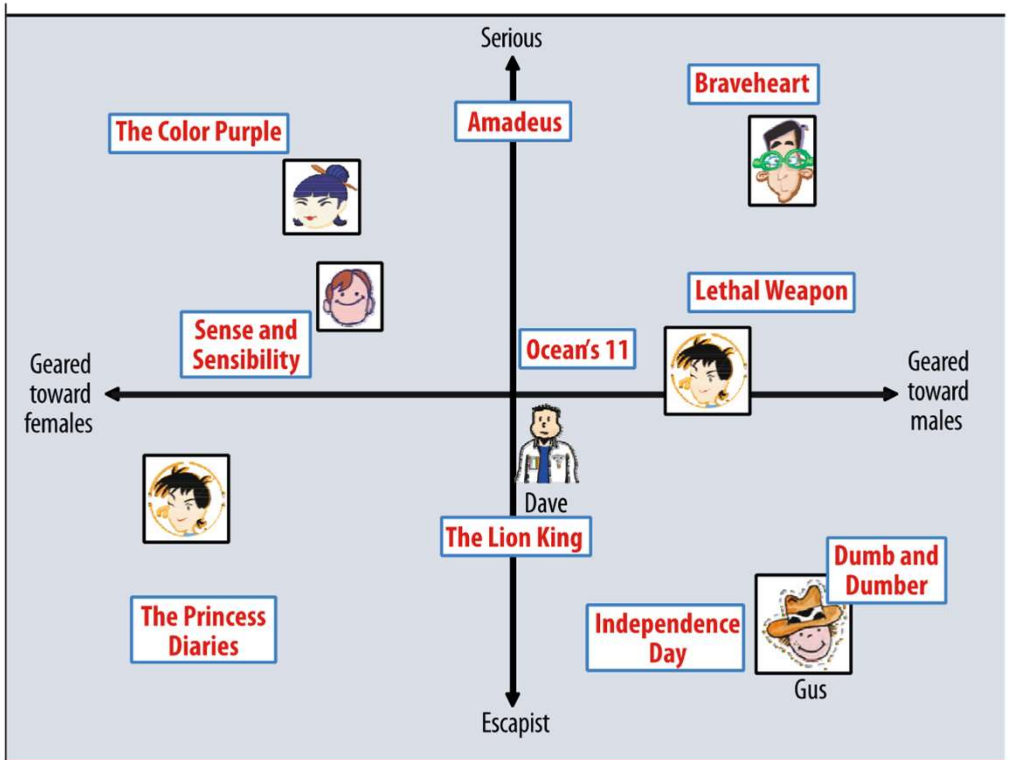


Figure 2. A simplified illustration of the latent factor approach, which characterizes both users and movies using two axes—male versus female and serious versus escapist.

Source: Koren, et al.

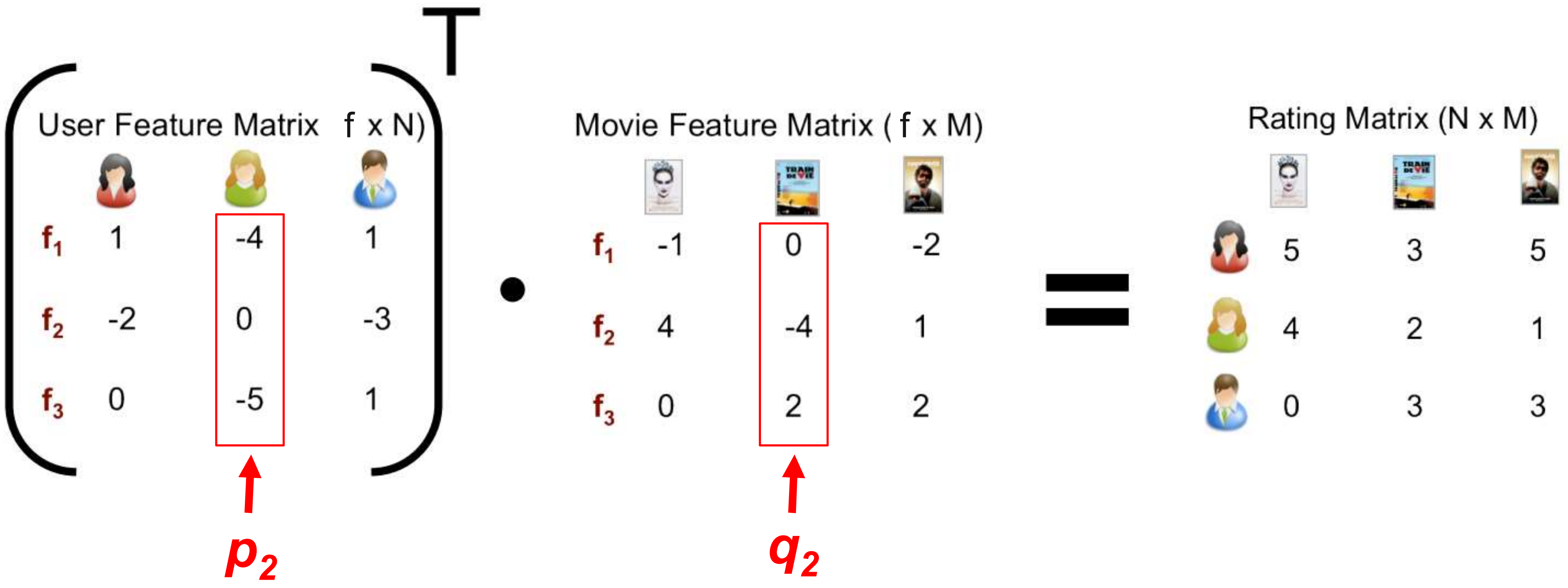
Latent Factor Models

- **Latent Factor models map both users and items to a joint latent factor space of dimensionality *f***
  - Each item *i* and user *u* is associated with a factor vector q_i , $p_u \in R^f$
  - For a given item *i*, elements of q_i measure the extent to which the item possesses those factors (positive or negative)
  - For a given user *u*, elements of p_u measure the extent of interest the user has in items that are high on the corresponding factors (positive or negative)
- **User-item interactions are modelled as dot product in that space**
  - The dot product, captures the interaction between user *u* and item *i* – namely the user's overall interest in the item's characteristics

$$\hat{r}_{ui} = q_i^T p_u$$

Matrix Factorization

- **How to learn the mapping of items and users to the corresponding factor vectors q_i , $p_u \in R^f$?**
- **Approach: approximately decompose rating matrix into dot product of user feature and item feature matrices**



- rating matrix is usually sparse: e.g. Netflix 1% filled, 99% ratings missing
- thus, we need an approach that can ignore missing ratings

Matrix Factorization

- **To learn the factor vectors (p_u and q_i) a solution is to minimizes the squared error on the set of known ratings**

$$\min_{q^*,p^*} \sum_{(u,i)\in K} (r_{ui} - q_i^T p_u)^2$$

- r_{ui} is the known rating of user u for item *i*
- $\hat{r}_{ui} = q_i^T p_u$ is the predicted rating

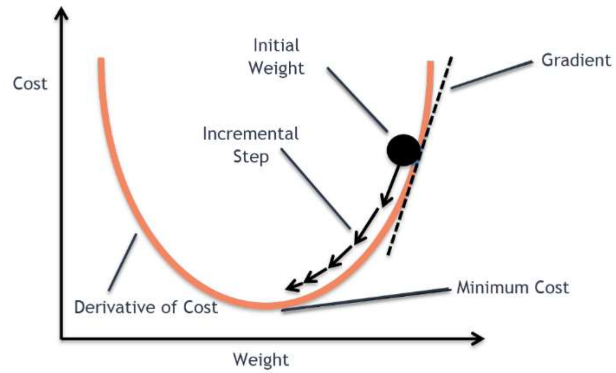
- **We add a regularization term to avoid overfitting**

$$\min_{q^*,p^*} \sum_{(u,i)\in K} (r_{ui} - q_i^T p_u)^2 + \lambda(\|q_i\|^2 + \|p_u\|^2)$$

- Lambda λ is a hyperparameter to control the extend of the regularization

Stochastic Gradient Descent

- **Simon Funk popularized stochastic gradient descent for optimizing the previous equation**



1. loop through all ratings in the training set. For each rating in the training set predict r_{ui} and compute the prediction error e_{ui}

$$e_{ui} \overset{def}{=} r_{ui} - q_i^T p_u$$

2. modify the parameters by a magnitude proportional to the momentum Gamma γ in the opposite direction of the gradient

$$q_i \leftarrow q_i + \gamma \cdot (e_{ui} \cdot p_u - \lambda \cdot q_i)$$
$$p_u \leftarrow p_u + \gamma \cdot (e_{ui} \cdot q_i - \lambda \cdot p_u)$$

See: Machine Learning (Rainer Gemulla)

http://sifter.org/~simon/journal/20061211.html
Bing Liu: Web Data Mining. Chapter 12.4.5.

Item and User Bias

- **Item or user specific rating variations are called *biases***
  - *Some users always give lower rating than others*
  - *Good items receive on average higher ratings*
- **Explicitly modelling the biases improves model performance**

$$b_{ui} = \mu + b_i + b_u$$

- μ is the overall average rating
- b_i and b_u indicate the observed deviations of user *u* and item *i* from the overall average

Example: Item and User Bias

$$b_{ui} = \mu + b_i + b_u$$

- The average rating over all movies, μ, is 3.7 stars
- *Titanic* is better than an average movie, so it tends to be rated 0.5 stars above the average (b_i)
- Joe is a critical user, who tends to rate 0.3 stars lower than the average (b_u).
- The bias for *Titanic*'s rating by Joe would be 3.9 stars (3.7 + 0.5 - 0.3)

Adding Item and User Biases to the Model

- **To include biases, the equations for predicting ratings and learning latent factor vectors are extended as follows**

$$\hat{r}_{ui} = \mu + b_i + b_u + q_i^T p_u$$

Biases + Item/user interaction

$$\min_{p*,q*,b*} \sum_{(u,i)\in\kappa} (r_{ui} - \mu - b_u - b_i - p_u^T q_i)^2 + \lambda$$
$$(\| p_u \|^2 + \| q_i \|^2 + b_u^2 + b_i^2)$$

Results on the Netflix Challenge

- **Winning team further extended the model with**
  - implicit feedback in addition to ratings to overcome cold start problem
  - temporal dynamics: change of user preferences and biases over time
- **Results show that matrix factorization techniques**
  - outperform KNN
  - scale to large use cases
  - allow flexible modelling of use case

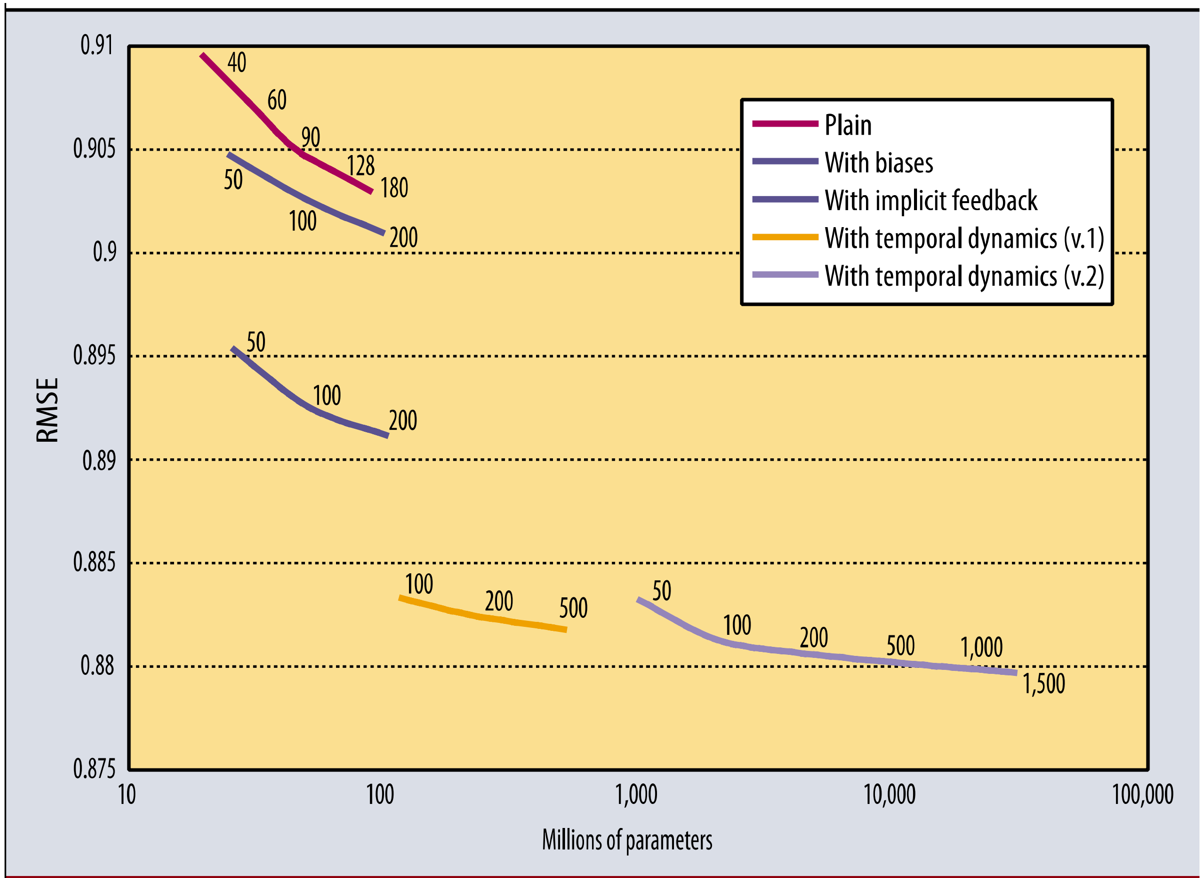
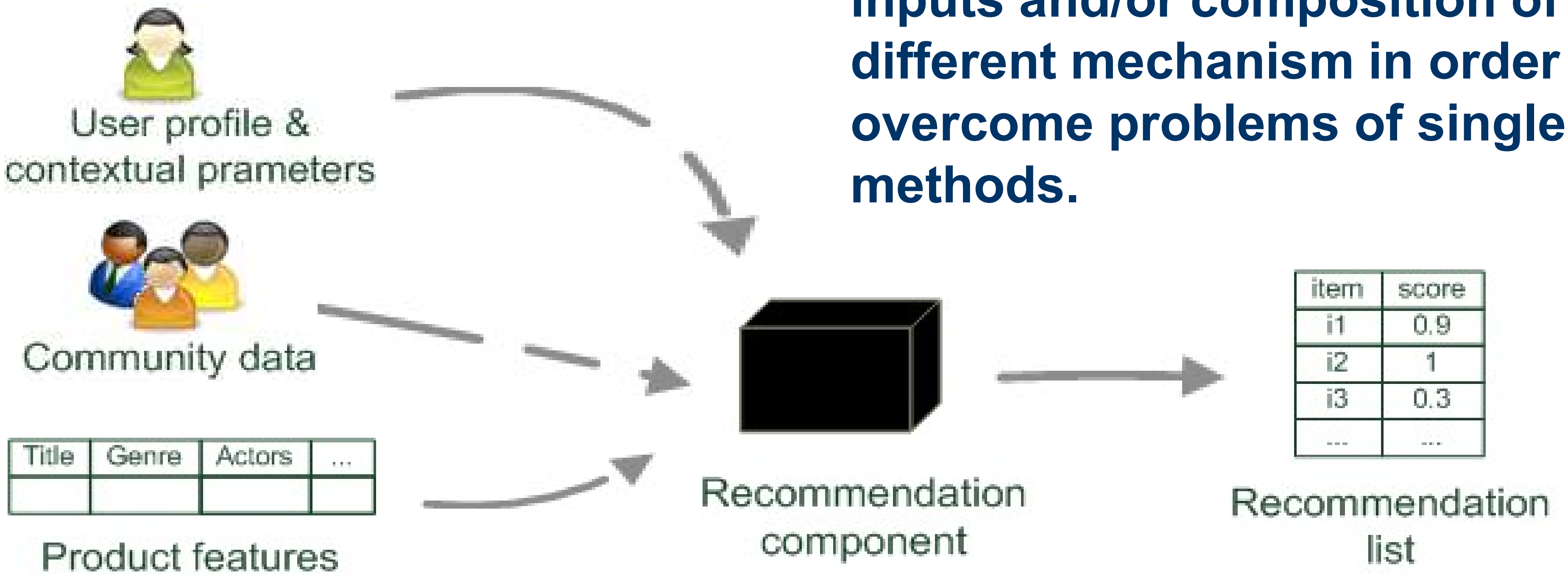


Figure 4. Matrix factorization models' accuracy. The plots show the root-mean-square error of each of four individual factor models (lower is better). Accuracy improves when the factor model's dimensionality (denoted by numbers on the charts) increases. In addition, the more refined factor models, whose descriptions involve more distinct sets of parameters, are more accurate. For comparison, the Netflix system achieves RMSE = 0.9514 on the same dataset, while the grand prize's required accuracy is RMSE = 0.8563.

2. Hybrid Recommender Systems



Hybrid: Combinations of various inputs and/or composition of different mechanism in order to overcome problems of single methods.

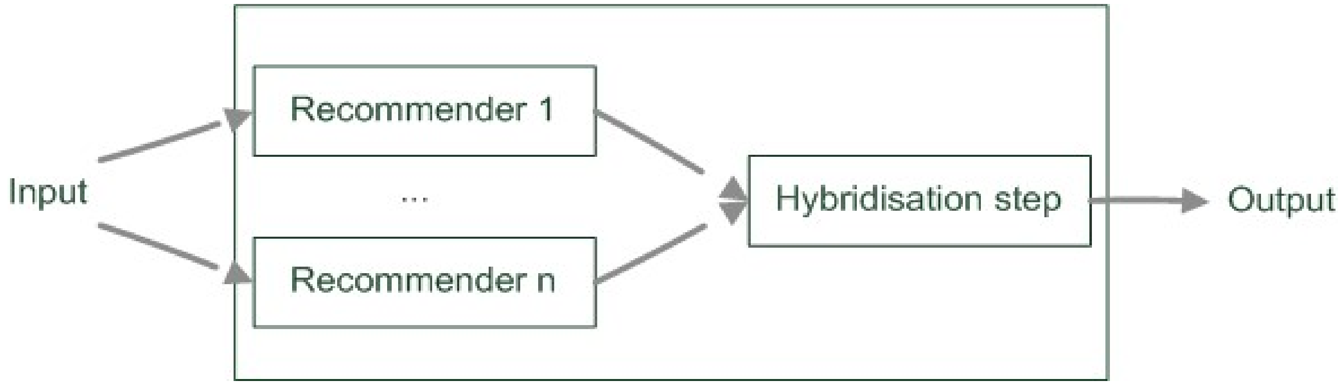
Collaborative: **"Tell me what's popular among my peers"**

Content-based: **"Show me more of the same what I've liked"**

Demographic: **“Offer American plugs to people from the US“**

Parallelized Hybridization Design

- **Output of several recommenders is combined**
- **Least invasive design**
- **Requires some weighting or voting scheme**
  - **Static weights: Can be learned using existing ratings as supervision**
  - **Dynamic weighting: Adjust weights or switch between different recommenders as more information about users and items becomes available**
    - To deal with cold start problem: If too few ratings available for a new item, then use content-based recommendation, otherwise use collaborative filtering
  - **More expressive aggregation: Random Forest, Neural Net**



Parallelized Hybridization Design: Weighted

- Compute weighted sum: $$rec_{weighted}(u,i) = \sum_{k=1}^{n} \beta_k \times rec_k(u,i)$$

Recommender 1		
Item1	0.5	1
Item2	0	
Item3	0.3	2
Item4	0.1	3
Item5	0	

Recommender 2		
Item1	0.8	2
Item2	0.9	1
Item3	0.4	3
Item4	0	
Item5	0	

Recommender weighted (0.5:0.5)		
Item1	0.65	1
Item2	0.45	2
Item3	0.35	3
Item4	0.05	4
Item5	0.00	

Suitable for blending user taste and content to be pushed by service provider

Learning the Weights for Each User

- **Use existing ratings to learn individual weights for each user**
- **Compare prediction of recommenders with actual ratings by user**
- **For each user adapt weights to minimize Mean Absolute Error (MAE)**

Absolute errors and MAE

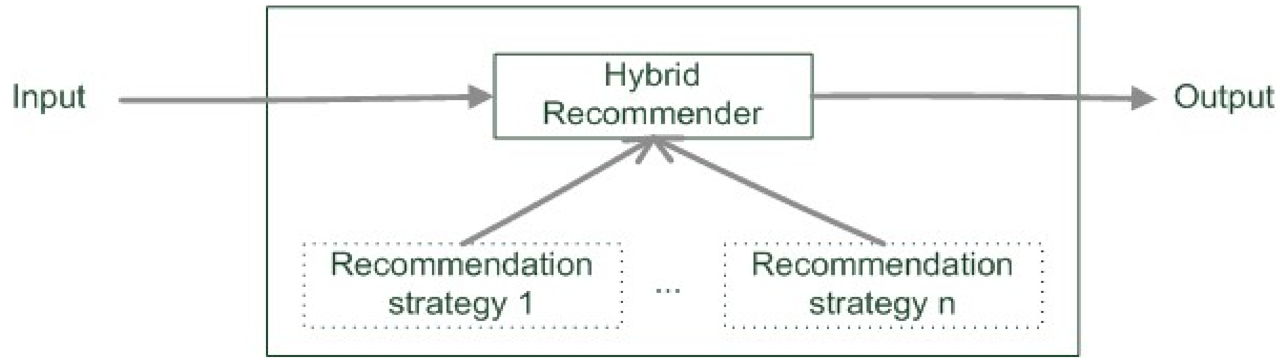
Weight1	Weight2		rec1	rec2	error	MAE
0.1	0.9	Item1	0.5	0.8	0.23	0.61
		Item4	0.1	0.0	0.99	
0.3	0.7	Item1	0.5	0.8	0.29	0.63
		Item4	0.1	0.0	0.97	
0.5	0.5	Item1	0.5	0.8	0.35	0.65
		Item4	0.1	0.0	0.95	
0.7	0.3	Item1	0.5	0.8	0.41	0.67
		Item4	0.1	0.0	0.93	
0.9	0.1	Item1	0.5	0.8	0.47	0.69
		Item4	0.1	0.0	0.91	

$$MAE = \frac{\sum_{r_i \in R} \sum_{k=1}^{n} \beta_k \times |rec_k(u,i) - r_i|}{|R|}$$

MAE improves as rec2 is given more weight

Monolithic Hybridization Design

- **Features/knowledge of different recommendation paradigms are combined in a single recommendation component. E.g.:**
  - Ratings and user demographics: Users living in town y currently like x
  - Ratings and content features: user rated many movies positive which are comedies ➔ recommend more comedies



- **Example: Content-boosted Collaborative Filtering**
  - based on content features additional ratings are created
  - e.g. Alice likes Items 1 and 3 (unary ratings)
    - Item7 is similar to 1 and 3 by a degree of 0.75
    - Thus, add rating of 0.75 for Alice/Item7 to rating matrix
  - rating matrix becomes less sparse

3. Evaluating Recommender Systems

How to quantify the performance of a recommender system?

- **Different views on performance:**
  - How good is the system with respect to a performance measures like mean absolute error (MAE) or F1 given ground-truth judgements?
  - Do customers buy items they otherwise would have not bought?
  - Do the recommendations help to increase the merchant's profit?
  - We need to determine the view that matters to us
- **Here we focus on measuring the degree of performance when compared to ground truth judgements**
  - useful for comparing different systems, optimizing hyperparameters, hybridization, etc.

Evaluating Recommender Systems

- **Assume we have ground-truth judgements that tell us what good and bad recommendations for a user are**
- **Popular Evaluation Measures**
  - **for numerical ratings – e.g., on a Likert scale between 1 and 5**
    - Mean Absolute Error (MAE)
    - Root Mean Squared Error (RMSE)
  - **for categorical ratings – e.g., like/dislike or good, neutral, bad**
    - Accuracy
    - Precision, Recall, F1-Score
  - **for ranked results – useful when items are presented as ranked lists**
    - Average Precision (AP), Precision at rank k (P@k)
    - Normalized Discounted Cumulative Gain (nDCG)
- **In addition to selecting a measure, we need an evaluation setup that ensures a good estimate for unseen data**

Evaluation Setup

Training Data | Validation Data | Test Data

- **If dataset is large, use fixed training/validation/test split**
  - Use the training data for training the model
  - Optimize the hyperparameters on validation (held-out) data
  - Once trained, evaluate the model on the test set
- **If dataset is small, optimize hyperparameters using *k-fold cross-validation (CV)***
  - test afterwards using hold-out test set (see next slide)

k-fold cross-validation (CV):

1. Split the training set into *k* portions of approx. equal size
2. For each fold *i* from 1 to *k*
3. Train the model on all folds but *i*
4. Test the model on fold *i*
5. Evaluate average model performance over *k* folds

Model Selection

- Overall process: selecting the hyperparameters, training, testing:

Select – Train – Evaluate:

1. Split the data set into a training set and a test set (e.g.,70–30%)
2. **Model selection**: for each hyperparameter configuration
   Cross-validate the model on the training set (e.g., 5-fold CV)
3. Choose the best performing hyperparameter configuration
4. **Train** model with best hyperparameters on the whole train set
5. **Evaluate** the trained model on the test set

- This ensures that your model is not overfitted to the test set
- You get a realistic estimate of its performance on unseen data

See: Data Mining I: Classification 3

3.1 Evaluation with Numerical Ratings

- **The gold standard consists of ground-truth judgements of how much a user likes an item**
  - **e.g., on a Likert scale between 1 and 5**

- **Mean Absolute Error (*MAE*) computes the deviation between predicted ratings and actual ratings**

$$MAE = \frac{1}{n}\sum_{i=1}^{n} | p_i - r_i |$$

- **Root Mean Square Error (*RMSE*) is similar to *MAE*, but places more emphasis on larger deviation**

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n} (p_i - r_i)^2}$$

Example: MAE versus RMSE

Nr.	UserID	MovieID	Rating (r_i)	Prediction (p_i)	$\|p_i-r_i\|$	$(p_i-r_i)^2$
1	1	134	5	4.5	0.5	0.25
2	1	238	4	5	1	1
3	1	312	5	5	0	0
4	2	134	3	5	2	4
5	2	767	5	4.5	0.5	0.25
6	3	68	4	4.1	0.1	0.01
7	3	212	4	3.9	0.1	0.01
8	3	238	3	3	0	0
9	4	68	4	4.2	0.2	0.04
10	4	112	5	4.8	0.2	0.04

MAE = 0.46
RMSE = 0.75

emphasis on larger deviation

3.2 Evaluation with Binary Categorical Ratings

- **The gold standard consists of ground-truth judgements of whether a user likes or dislikes an item.**

- **Precision: Measure of exactness.**
  - determines the fraction of relevant items retrieved out of all items retrieved
  - fraction of recommended movies that are actually good / liked by the user

$$Precision = \frac{tp}{tp + fp} = \frac{|good\ movies\ recommended|}{|\text{all recommendations}|}$$

- **Recall: Measure of completeness.**
  - determines the fraction of relevant items retrieved out of all relevant items
  - E.g. the fraction of all good movies recommended

$$Recall = \frac{tp}{tp + fn} = \frac{|good\ movies\ recommended|}{|all\ good\ movies|}$$

- **F1-Measure**
  - combines Precision and Recall into a single value for comparison purposes.
  - May be used to gain a more balanced view of performance

$$F_1 = 2 \cdot \frac{precision \cdot recall}{precision + recall}$$

3.3 Evaluation with Multi-Class Categorical Ratings

- The gold standard consists of discrete ground-truth judgements towards an item.
  - e.g. good, neutral, bad

- If we have K > 2 classes, we obtain a K x K confusion matrix
- E.g., for K = 3 (rows = predicted labels, columns = actual labels)

$$\begin{array}{c c} & \begin{array}{ccc} 1 & 2 & 3 \end{array} \\ \begin{array}{c} 1 \\ 2 \\ 3 \end{array} & \begin{pmatrix} 1 & 1 & 0 \\ 2 & 2 & 3 \\ 0 & 0 & 4 \end{pmatrix} \end{array}$$

- We can derive 2-way confusion matrices for each class:

$$\begin{pmatrix} tp & fp \\ fn & tn \end{pmatrix} \Rightarrow \underset{y=1}{\begin{pmatrix} 1 & 1 \\ 2 & 9 \end{pmatrix}} \quad \underset{y=2}{\begin{pmatrix} 2 & 5 \\ 1 & 5 \end{pmatrix}} \quad \underset{y=3}{\begin{pmatrix} 4 & 0 \\ 3 & 6 \end{pmatrix}}$$

- Two options: micro and macro measure aggregation

Macro Measures

- **Compute Acc, P, R, and F1 for each class separately:**

$$Acc_1 = 0.77, P_1 = 0.50, R_1 = 0.33, F_{1,1} = 0.40$$
$$Acc_2 = 0.54, P_2 = 0.29, R_2 = 0.67, F_{1,2} = 0.40$$
$$Acc_3 = 0.77, P_3 = 1.00, R_3 = 0.57, F_{1,3} = 0.73$$

- **Average measures across all classes:**

$$Acc^M = \frac{\sum Acc_i}{K} \quad P^M = \frac{\sum P_i}{K} \quad R^M = \frac{\sum R_i}{K} \quad F_1^M = \frac{\sum F_{1,i}}{K}$$

$$Acc^M = 0.69, P^M = 0.60, R^M = 0.52, F_1^M = 0.51$$

- **Macro measures give equal weight to each class**
- **If the classifier performs very poorly on one of the classes, this can have a big effect on the average score**

Micro Measures

- Micro Measures give **equal weight to every instance**
- Obtain global *tp*, *fp*, *fn*, and *tn* counts by summing 2-way confusion matrices for all classes

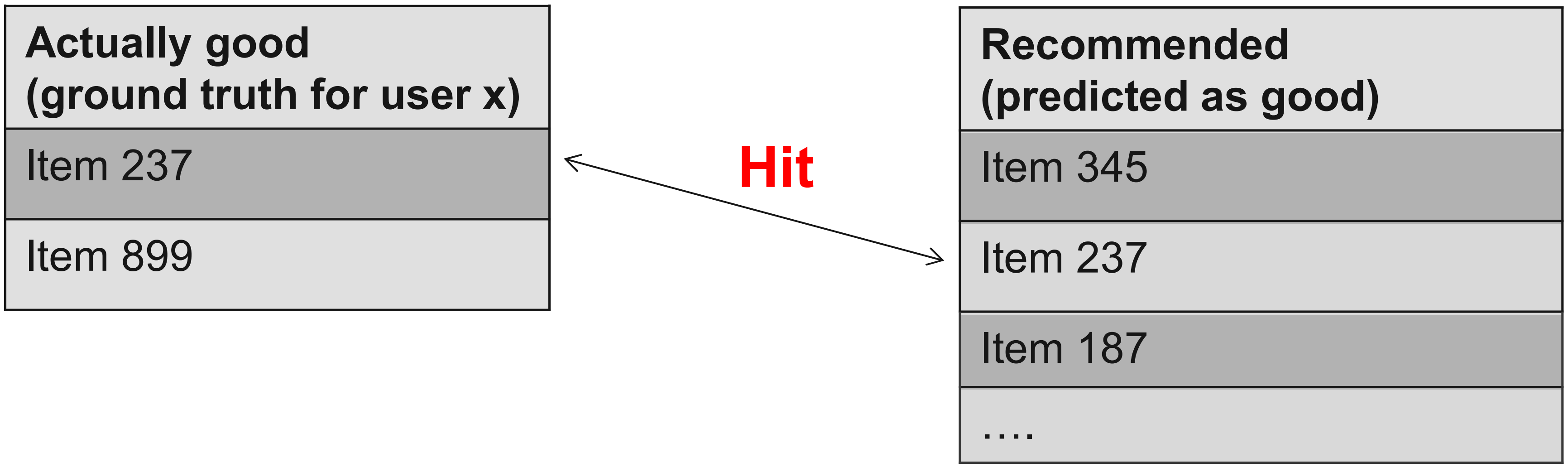
$$\begin{pmatrix} 1 & 1 \\ 2 & 9 \end{pmatrix} + \begin{pmatrix} 2 & 5 \\ 1 & 5 \end{pmatrix} + \begin{pmatrix} 4 & 0 \\ 3 & 6 \end{pmatrix} = \begin{pmatrix} tp = 7 & fp = 6 \\ fn = 6 & tn = 20 \end{pmatrix}$$

- From the aggregated 2-way confusion matrix we can compute the measures in a usual way

$$Acc = 0.69, P^{\mu} = R^{\mu} = F_1^{\mu} = 0.54$$

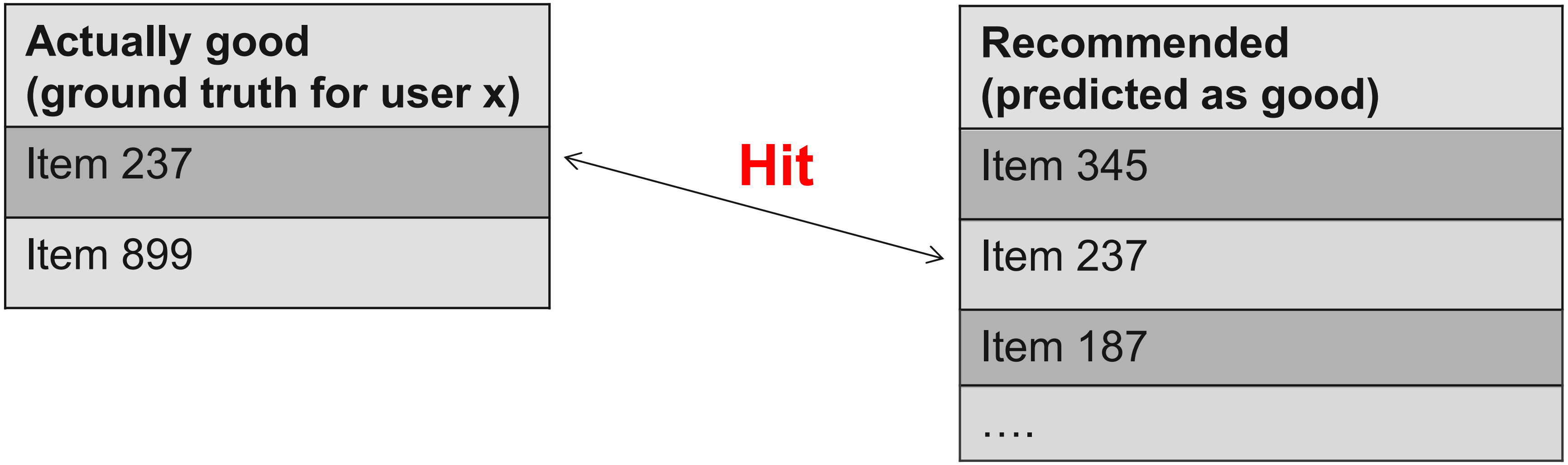
- Note:
  - Always *fp* = *fn*, thus micro P, R, and F1 **are the same**
  - Micro and macro accuracy are equal
  - Commonly micro F1 > macro F1 because classifiers tend to fail on **classes with fewer instances** and such classes **impact micro average less**

3.4 Evaluation of Ranked Results



- **Rank position also matters!**
- **Rank metrics take the positions of relevant items in a ranked list into account**
  - Relevant items are more useful when they appear higher in the recommendation list
  - Particularly important in recommender systems as lower ranked items may be overlooked by users

Evaluation of Ranked Results



- **The gold standard consists of ground-truth judgements of whether an item is relevant (i.e., to be recommended) for a user, i.e., binary relevance annotations**
  - Average Precision (AP), P@K, R-Precision
- **Alternatively, we can have graded relevance annotations e.g., from 1 (marginally relevant) to 5 (highly relevant)**
  - Normalized Discounted Cumulative Gain (nDCG)

Average Precision

- **Average Precision (*AP*) is computed by averaging the precision scores measured at ranks of relevant items (hits)**

$$AP = \frac{1}{n}\sum_{k=1}^{n} P(R_k)$$

Rank	Hit?
1	
2	X
3	X
4	X
5	

$$AP = \frac{1}{3}\left(\frac{1}{2}+\frac{2}{3}+\frac{3}{4}\right) = \frac{23}{36} \approx 0.639$$

Rank	Hit?
1	X
2	
3	
4	X
5	X

$$AP = \frac{1}{3}\left(\frac{1}{1}+\frac{2}{4}+\frac{3}{5}\right) = \frac{21}{30} = 0.7$$

P@k and R-precision

- Average Precision considers *all recall levels*, even at very low ranks
- This might be inappropriate since most users will look only at a few top recommendations

- **Precision at k (P@k) is precision at the fixed rank *k* in the ranking (e.g., P@5, P@10, P@20)**
  - number of relevant items in top-k list
- **R-Precision is the P@k where k equals to the number of relevant items**
  - number of relevant items is used as the cutoff for calculation
  - k varies from user to user, e.g., if 5 items are in total relevant for user X, then R-precision = P@5

Normalized Discounted Cumulative Gain (nDCG)

- **Sometimes we have graded relevance annotations**
  - **E.g., from 1 (marginally relevant) to 5 (highly relevant)**
- **Assumptions**
  - Highly relevant items are more useful than marginally relevant items
  - The higher the relevance of the item, the higher it should appear in the relevance ranking

Rank	Rel
1	1
2	2
3	0
4	4
5	0

Better ranking

Rank	Rel
1	4
2	2
3	0
4	1
5	0

- **nDCG takes into account the graded relevancies of items when evaluating the ranking**

Normalized Discounted Cumulative Gain (nDCG)

- **Discounted Cumulative Gain**
  - Idea: Normalize the relevance scores of items at every position with the position itself
  - That way, highly relevant but low-ranked items contribute less to the overall score, i.e., they get penalized more

$$\mathrm{DCG(k)} = \sum_{i=1}^{k} \frac{rel_i}{\log_2(i+1)}$$

  - There is an alternative formulation of DCG, that places stronger emphasis on retrieving relevant items (and a bit less on their mutual relative ranking)

$$\mathrm{DCG(k)} = \sum_{i=1}^{k} \frac{2^{rel_i} - 1}{\log_2(i+1)}$$

Normalized Discounted Cumulative Gain (nDCG)

- **Maximal DCG score depends on the number of relevant items**
- **For comparing DCG scores across users, we need to normalize them:**
- **Ideal DCG (IDCG) is the maximal DCG score any ranking can have**

$$\mathrm{IDCG(k)} = \sum_{i=1}^{|relevant|} \frac{2^{rel_i} - 1}{\log_2(i+1)}$$

- **Normalized nDCG is the DCG(k) score normalized with the IDCG(k), where k is the total number of relevant items**

$$nDCG = \frac{DCG(k)}{IDCG(k)}$$

- **Value range nDCG 0 to 1**

nDCG Example

$$\mathrm{DCG(k)} = \sum_{i=1}^{k} \frac{2^{rel_i} - 1}{\log_2(i+1)}$$

$$\mathrm{DCG}(5) = \frac{2^2-1}{\log_2(2)} + \frac{2^1-1}{\log_2(3)} + \frac{2^4-1}{\log_2(5)}$$

$$= \frac{3}{1} + \frac{1}{1.58} + \frac{15}{2.32} = 8.09$$

$$\mathrm{IDCG}(5) = \frac{2^4-1}{\log_2(2)} + \frac{2^2-1}{\log_2(3)} + \frac{2^1-1}{\log_2(4)}$$

$$= \frac{15}{1} + \frac{3}{1.58} + \frac{1}{2} = 17.40$$

$$\mathrm{nDCG}(5) = \frac{\mathrm{DCG}(5)}{\mathrm{IDCG}(5)} = \frac{8.09}{17.40} = 0.46$$

Ranking by System

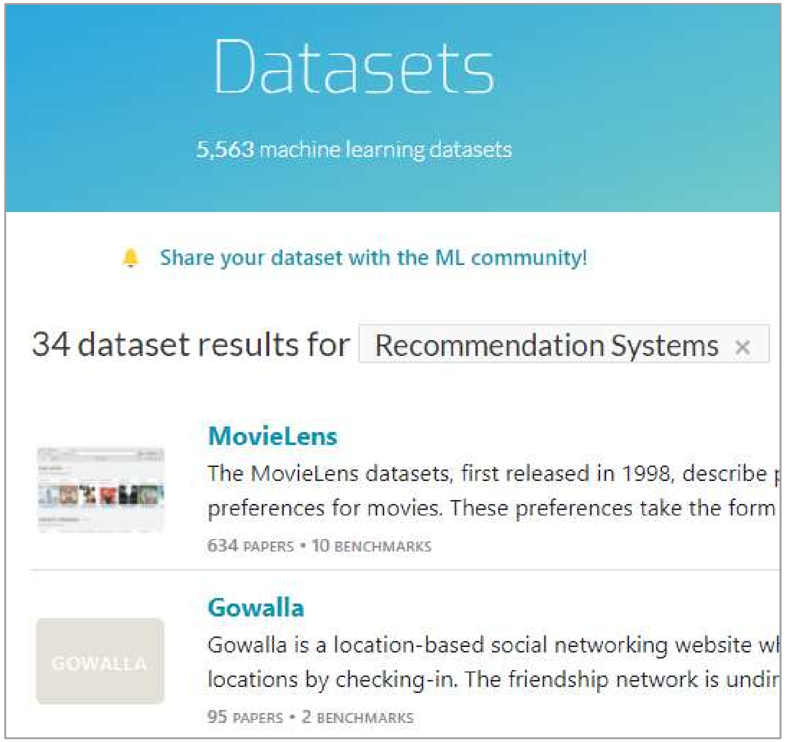
Rank	Rel
1	2
2	1
3	0
4	4
5	0

Idealized Ranking

Rank	Rel
1	4
2	2
3	1
4	0
5	0

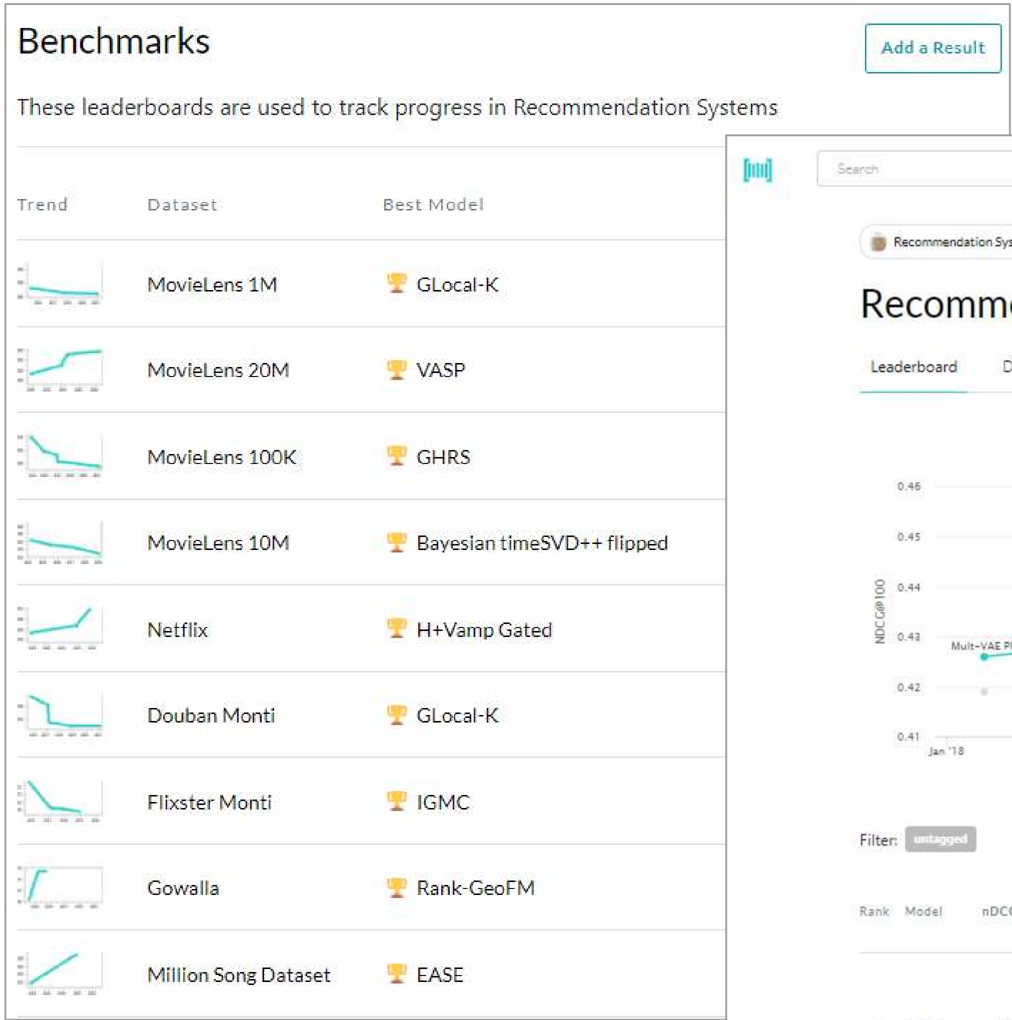
Benchmark Datasets

- **MovieLens**
  - 1M Dataset: 6.000 users, 3.900 movies, 1 million ratings
  - 10M Dataset: 71.000 users, 10.600 movies, 10 million ratings
  - included in Surprise library used in the lab
- **Netflix Challenge**
  - 100M Dataset: 500.000 users, 18.000 movies, 100M ratings
- **Amazon Product Reviews**
  - 230M product reviews including star ratings
  - https://nijianmo.github.io/amazon/
- **Microsoft MIND**
  - 160k English news articles and
  - 15 million impression logs by 1 million users
  - https://msnews.github.io/
- **Papers with Code**
  - collects benchmark datasets
  - https://paperswithcode.com/datasets?task=recommendation-systems



Benchmark Results

https://paperswithcode.com/task/recommendation-systems

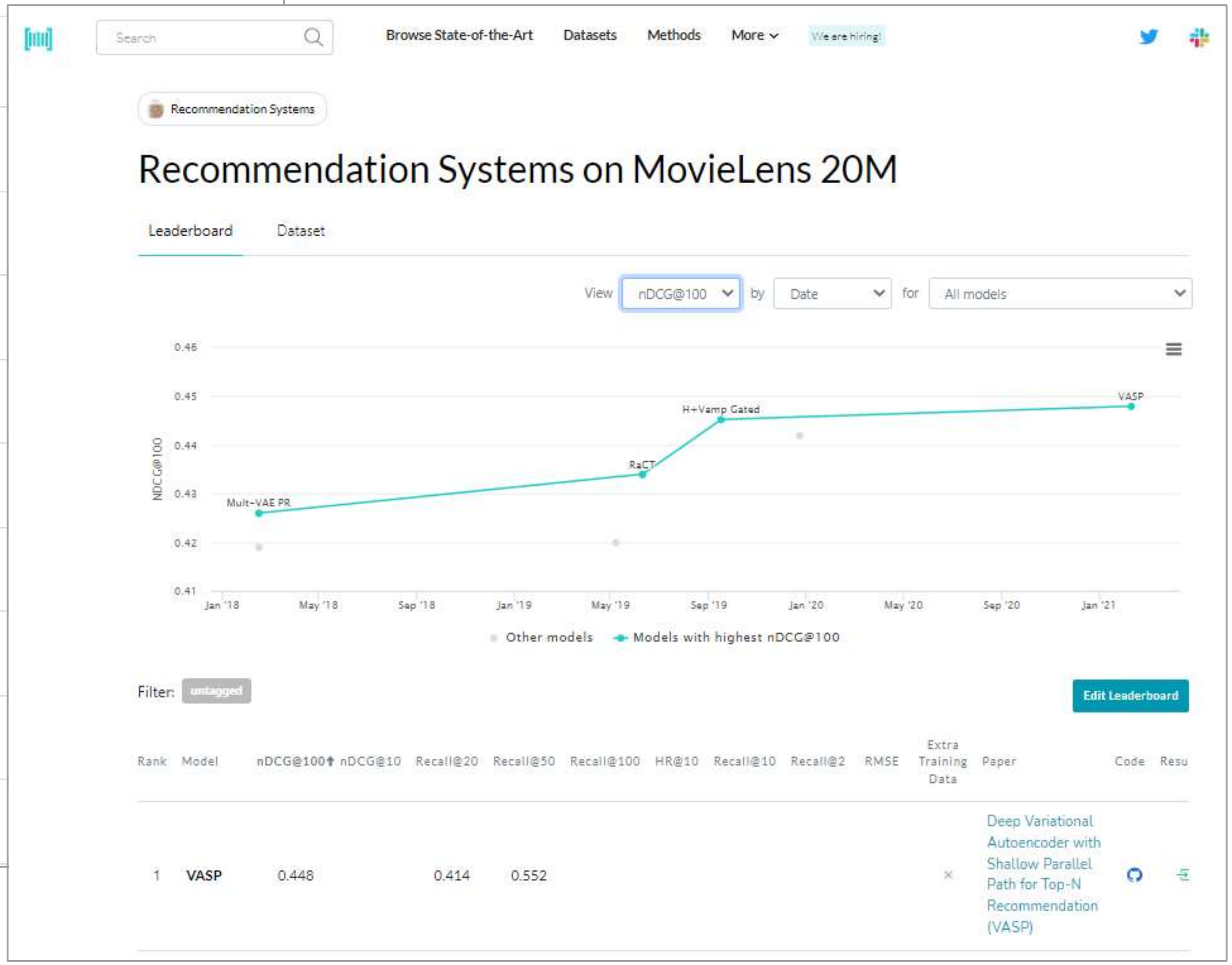


Benchmarks

Add a Result

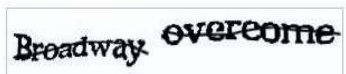
These leaderboards are used to track progress in Recommendation Systems

Trend	Dataset	Best Model
	MovieLens 1M	GLocal-K
	MovieLens 20M	VASP
	MovieLens 100K	GHRS
	MovieLens 10M	Bayesian timeSVD++ flipped
	Netflix	H+Vamp Gated
	Douban Monti	GLocal-K
	Flixster Monti	IGMC
	Gowalla	Rank-GeoFM
	Million Song Dataset	EASE



4. Attacks on Recommender Systems

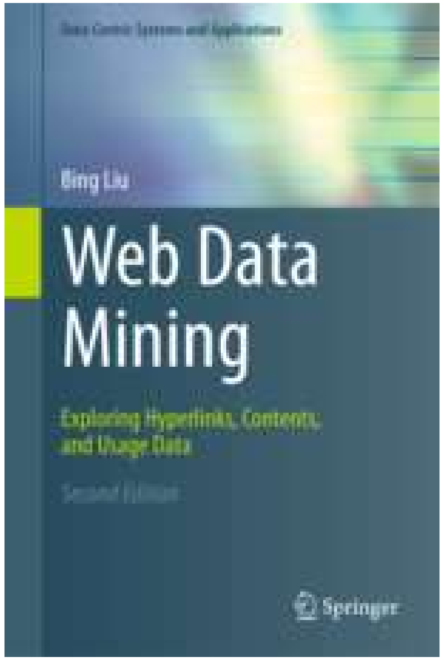
- **As there is (monetary) value in being on recommendation lists**
  - individuals/companies may be interested to push or nuke some items by manipulating the recommender system
- **Basic Attack Strategies**
  - automatically create numerous fake accounts / profiles
  - issue high or low ratings for target item
  - rate additional filler items in order to
    - make fake profile appear in neighborhood of many real-world users and
    - camouflage fake profiles
  - for implicit ratings: Use crawler that automatically navigates the site
- **Counter measures**

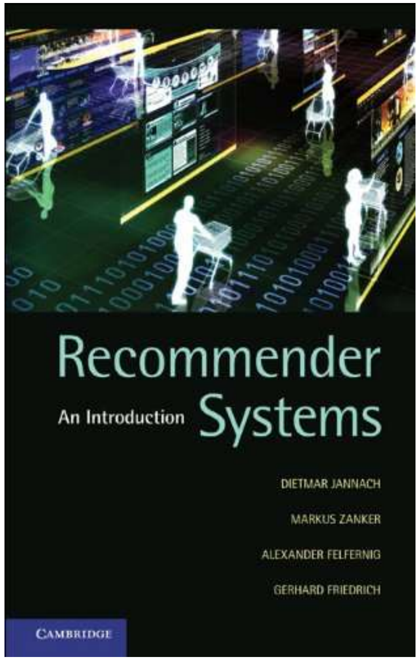


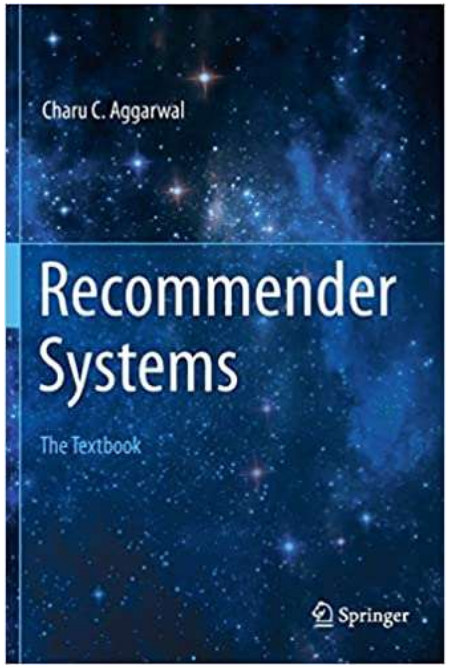
  1. make it difficult to generate fake profiles (e.g. using Captchas)
  2. use machine-learning methods to discriminate real from fake profiles
- **Details on attacks and countermeasures**
  - Jannach et al.: Recommender Systems. Chapter 9

Literature

- **Bing Lui: Web Data Mining. Chapter 12: Web Usage Mining**
- **Jannach et al.: Recommender Systems. 2011.**
- **Charu Aggarwal: Recommender Systems: The Textbook. 2016.**







Summary

1. Usage Data Collection
2. Usage Data Preprocessing
   1. User and Session Identificat
   2. Data Aggregatio
3. Usage

   mendation

   Recommendation

   ybrid Recommendation

   5. Evaluating Recommender Systems
   6. Attacks on Recommender Systems

Next week: Web Structure Mining