University of Antwerp
Faculty of Science

Theoretically Sound Item Similarity Measures for Recommender Systems

Noah Daniëls

TCS Seminar 25/02/2026

Recommender Systems

What are Recommender Systems (RecSys)?

"Recommender systems utilize various sources of data to infer customer interests."
– From *Recommender Systems Textbook by Aggarwal (IBM, PhD MIT)*

"A recommender system (...) is a subclass of information filtering system that provides suggestions for items that are most pertinent to a particular user"
– From *wikipedia*

- Predict customer behavior
- With aim of aiding their decision making
- Personalization

Personalization Techniques

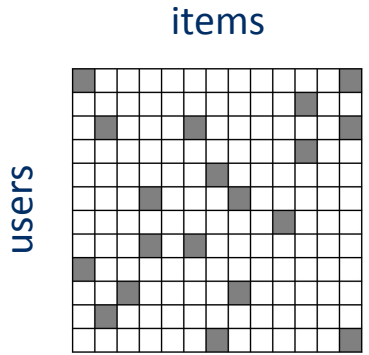
- Similar users will like the same items
- Similar items will be liked by the same users

- How to determine similarities?
  - **Content-based Filtering***:* based on attributes of users/items
  - **Collaborative Filtering**: based on interaction data

Interaction Data

- **Explicit:** user states what they like (e.g. ratings or like/dislike button)
- **Implicit:** infer what user likes based on their actions (e.g. clicks)
  - More common and easily obtainable
  - Ambiguous: negative preferences vs unknowns

- **Organized in a (user x item) matrix X**
  - 1 indicates an interaction
  - 0 indicates a missing value
  - Sparsity allows for efficient operations despite size



RecSys Algorithms

k-Nearest Neighbors (kNN)

Idea: rank items based on their “similarity” to items in the user’s history

Scoring function

- Sum similarities between target item i and all history items j
- Quantify similarities using an (item x item) similarity matrix S

$$\hat{y}(u,i) = \sum_j X_{uj} S_{ij} = \langle X_u, S_i \rangle$$

Computation

- Efficient sparse matrix product: $\mathbf{Y = XS^T}$

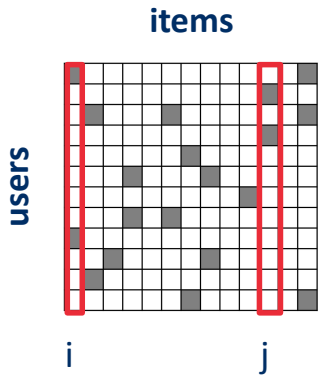
Heuristic Similarity Matrix

Typically based on (co-)occurrence counts

- n(i, j) = number of users who interacted with i and j
- n(i) = number of users who interacted with item i

$$S_{ij}^{(\text{Cosine})} = \cos(X_{\cdot i}, X_{\cdot j}) = \frac{n(i,j)}{\sqrt{n(i) \cdot n(j)}}$$

$$S_{ij}^{(\text{Cond. prob.})} = P(i \mid j) = \frac{n(i,j)}{n(j)}$$



EASE: Embarrassingly Shallow Autoencoders

Idea: learn a linear function to reconstruct interaction matrix (autoencoder)

- Optimize **X = XB** s.t. diagonal of **B** is zero
- Closed form solution for **B** exists when using square loss
- Predict scores **Y = XB**
  - Same as kNN with learned "similarity matrix" $\mathbf{B}^T$

- State of the art performance despite simple linear model

Search for Theoretically Sound Similarities

- **kNN** uses a heuristic similarity matrix
  - Computationally fast
  - Interpretable and explainable
  - Lacks performance

- **EASE** uses an optimized item weight matrix
  - Computationally expensive
  - Difficult to interpret

Is there a middle ground?

Naive Bayes Recommendations

Basic Naive Bayes Recommender

$$\hat{y}(u, i) = P(i \mid \{j : X_{uj} = 1\})$$

Basic Naive Bayes Recommender

$$\hat{y}(u,i) = P(i \mid \{j : X_{uj} = 1\})$$
$$\propto P(i, \{j : X_{uj} = 1\})$$

Basic Naive Bayes Recommender

$$\begin{aligned}\hat{y}(u,i) &= P(i \mid \{j : X_{uj} = 1\}) \\ &\propto P(i, \{j : X_{uj} = 1\}) \\ &= P(i)P(\{j : X_{uj} = 1\} \mid i)\end{aligned}$$

Basic Naive Bayes Recommender

$$
\begin{aligned}
\hat{y}(u,i) &= P(i \mid \{j : X_{uj} = 1\}) \\
&\propto P(i, \{j : X_{uj} = 1\}) \\
&= P(i) P(\{j : X_{uj} = 1\} \mid i) \\
&\overset{\text{NB}}{=} P(i) \prod_{j : X_{uj} = 1} P(j \mid i)
\end{aligned}
$$

Basic Naive Bayes Recommender

$$
\begin{aligned}
\hat{y}(u,i) &= P(i \mid \{j : X_{uj} = 1\}) \\
&\propto P(i, \{j : X_{uj} = 1\}) \\
&= P(i) P(\{j : X_{uj} = 1\} \mid i) \\
&\overset{\text{NB}}{=} P(i) \prod_{j : X_{uj} = 1} P(j \mid i) \\
&\rightarrow \log P(i) + \sum_{j} X_{uj} \log P(j \mid i)
\end{aligned}
$$

Basic Naive Bayes Recommender

$$
\begin{aligned}
\hat{y}(u,i) &= P(i \mid \{j : X_{uj} = 1\}) \\
&\propto P(i, \{j : X_{uj} = 1\}) \\
&= P(i) P(\{j : X_{uj} = 1\} \mid i) \\
&\overset{\text{NB}}{=} P(i) \prod_{j : X_{uj} = 1} P(j \mid i) \\
&\rightarrow \underbrace{\log P(i)}_{\text{popularity}} + \underbrace{\sum_j X_{uj} \log P(j \mid i)}_{\text{kNN}}
\end{aligned}
$$

Basic Naive Bayes Recommender

- **Similar to kNN**
  - Additional popularity term
  - Similarity measure is the log of the conditional probability (but flipped)

- **Conditional independence assumption**
  - Prevents higher order interactions between items
  - Also present in kNN

$$
\begin{aligned}
\hat{y}(u,i) &= P(i \mid \{j : X_{uj} = 1\}) \\
&\propto P(i, \{j : X_{uj} = 1\}) \\
&= P(i) P(\{j : X_{uj} = 1\} \mid i) \\
&\overset{\text{NB}}{=} P(i) \prod_{j:X_{uj}=1} P(j \mid i) \\
&\rightarrow \underbrace{\log P(i)}_{\text{popularity}} + \underbrace{\sum_j X_{uj} \log P(j \mid i)}_{\text{kNN}}
\end{aligned}
$$

Formalizing the Naive Bayes Recommender

- How do we represent a user's history?
  - **Bernoulli**: as a binary feature vector (vector of Bernoulli variables)
  - **Multinomial**: as a bag of items (multinomial vector of counts)

- Results in different scoring functions:

$$\hat{y}(u,i) = \log p_i + \sum_j X_{uj} \log p_{j|i} + \sum_j (1 - X_{uj}) \log(1 - p_{j|i})$$

$$p_i = \frac{n(i)}{n} \qquad p_{j|i} = \frac{n(i,j)}{n(i)}$$

Bernoulli

$$\hat{y}(u,i) = \log p_i + \sum_j X_{uj} \log p_{j|i}$$

$$p_i = \frac{n(i)}{\sum_k n(k)} \qquad p_{j|i} = \frac{n(i,j)}{\sum_k n(i,k)}$$

Multinomial

Bernoulli vs Multinomial

Our first derivation found Bernoulli probabilities, but the multinomial form

- **Bernoulli** scoring uses the presence and absence of items as features
- **Multinomial** scoring only considers items that are present

$$\hat{y}(u,i) = \log p_i + \sum_j X_{uj} \log p_{j|i} + \sum_j (1 - X_{uj}) \log(1 - p_{j|i})$$

$$p_i = \frac{n(i)}{n} \qquad p_{j|i} = \frac{n(i,j)}{n(i)}$$

Bernoulli

$$\hat{y}(u,i) = \log p_i + \sum_j X_{uj} \log p_{j|i}$$

$$p_i = \frac{n(i)}{\sum_k n(k)} \qquad p_{j|i} = \frac{n(i,j)}{\sum_k n(i,k)}$$

Multinomial

Parameter Estimation

Parameter Estimation

- **Maximum Likelihood Estimation (MLE)**
  - Used to estimate parameters of probability models
  - Maximize the likelihood of the data given the model parameters
- **Maximum A Posteriori (MAP) estimation**
  - Incorporate prior belief about parameter
  - Natural way to add regularization

For Naive Bayes models -> optimization can be done analytically

Bernoulli NB Likelihood Function

$$L_i^{(\text{Bernoulli})} = \prod_u P\left(Z_i = 1 \mid \{Z_j = X_{uj} : j \neq i\}, b_i, B_i^+\right)^{X_{ui}}$$
$$\cdot \prod_u P\left(Z_i = 0 \mid \{Z_j = X_{uj} : j \neq i\}, b_i, B_i^-\right)^{1-X_{ui}}$$

Bernoulli NB Likelihood Function

$$
\begin{aligned}
L_i^{(\text{Bernoulli})} &= \prod_u P\left(Z_i = 1 \mid \{Z_j = X_{uj} : j \neq i\}, b_i, B_i^+\right)^{X_{ui}} \\
&\quad \cdot \prod_u P\left(Z_i = 0 \mid \{Z_j = X_{uj} : j \neq i\}, b_i, B_i^-\right)^{1-X_{ui}} \\
&= \prod_u \left(\sigma(b_i) \prod_{j \neq i} \sigma(B_{ij}^+)^{X_{uj}} \sigma(-B_{ij}^+)^{1-X_{uj}}\right)^{X_{ui}} \\
&\quad \cdot \prod_u \left(\sigma(-b_i) \prod_{j \neq i} \sigma(B_{ij}^-)^{X_{uj}} \sigma(-B_{ij}^-)^{1-X_{uj}}\right)^{1-X_{ui}}
\end{aligned}
$$

Bernoulli NB Likelihood Function

$$L_i^{(\text{Bernoulli})} = \prod_u P\left(Z_i = 1 \mid \{Z_j = X_{uj} : j \neq i\}, b_i, B_i^+\right)^{X_{ui}}$$

$$\cdot \prod_u P\left(Z_i = 0 \mid \{Z_j = X_{uj} : j \neq i\}, b_i, B_i^-\right)^{1-X_{ui}}$$

$$= \prod_u \left(\sigma(b_i) \prod_{j \neq i} \sigma(B_{ij}^+)^{X_{uj}} \sigma(-B_{ij}^+)^{1-X_{uj}}\right)^{X_{ui}}$$

$$\cdot \prod_u \left(\sigma(-b_i) \prod_{j \neq i} \sigma(B_{ij}^-)^{X_{uj}} \sigma(-B_{ij}^-)^{1-X_{uj}}\right)^{1-X_{ui}}$$

$$\sigma(b_i) = p_i = \frac{n(i)}{n}$$

$$\sigma(B_{ij}^+) = p_{j|i} = \frac{n(i,j)}{n(i)}$$

$$\sigma(B_{ij}^-) = \frac{n(j) - n(i,j)}{n - n(i)}$$

Weighting the Negative Features

Weight the negative features by including γ in the exponent

- The weight affects the parameters/similarities
- Ignoring the negatives (γ=0) -> all similarities become 1

$$L_i = \prod_u \left(\sigma(b_i) \prod_{j \neq i} \sigma(B_{ij}^+)^{X_{uj}} \sigma(-B_{ij}^+)^{\gamma(1-X_{uj})} \right)^{X_{ui}} \cdot \prod_u \left(\sigma(-b_i) \prod_{j \neq i} \sigma(B_{ij}^-)^{X_{uj}} \sigma(-B_{ij}^-)^{\gamma(1-X_{uj})} \right)^{1-X_{ui}}$$

$$\sigma(b_i) = \frac{n(i)}{n}$$

$$\sigma(B_{ij}^+) = \frac{n(i,j)}{(1-\gamma)n(i,j) + \gamma n(i)}$$

$$\sigma(B_{ij}^-) = \frac{n(j) - n(i,j)}{(1-\gamma)\big(n(j) - n(i,j)\big) + \gamma\big(n - n(i)\big)}$$

Multinomial NB Likelihood Function

PROPOSITION 2. *The parameters that maximize the following likelihood function are equivalent to those of the multinomial Naive Bayes model:*

$$L^{(\text{Multinomial})} = \prod_{u,i} P(Z = i \mid \{j : X_{uj} = 1\}, \vec{b}, B_i)^{X_{ui}}$$

$$= \prod_{u,i} \left(\sigma(\vec{b}, i) \prod_{j \neq i} \sigma(B_i, j)^{X_{uj}} \right)^{X_{ui}}$$

Specifically the optimal parameters are given by:

$$\sigma(\vec{b}, i) = p_i = \frac{n(i)}{\sum_k n(k)}$$

$$\sigma(B_i, j) = p_{j|i} = \frac{n(i, j)}{\sum_k n(i, k)}$$

PROOF. The maximum of $L^{(\text{Multinomial})}$ w.r.t. each parameter is independently found by taking the logarithm and setting the derivative to zero. □

Smoothing

- **Some (co-)occurrence counts are zero**
  - Resulting probabilities are zero => entire score becomes zero

- **Smoothing: add pseudo-counts when calculating probabilities**
  - Form of *regularization*
  - Corresponds with adding priors in the likelihood (MAP)
  - Beta prior for Bernoulli parameters
  - Dirichlet prior for multinomial parameters

Smoothing with MAP Estimation

$$L_i^{(\text{Bernoulli, MAP})} = L_i^{(\text{Bernoulli})} \cdot \prod_j \text{Beta}(\sigma(B_{ij}^+) \mid \alpha, \beta)$$

$$= L_i^{(\text{Bernoulli})} \cdot \prod_j \left(\frac{\sigma(B_{ij}^+)^{\alpha-1} \sigma(-B_{ij}^+)^{\beta-1}}{\text{B}(\alpha, \beta)} \right)$$

$$\sigma(B_{ij}^+) = \frac{n(i,j) + \alpha - 1}{n(i) + \alpha + \beta - 2}$$

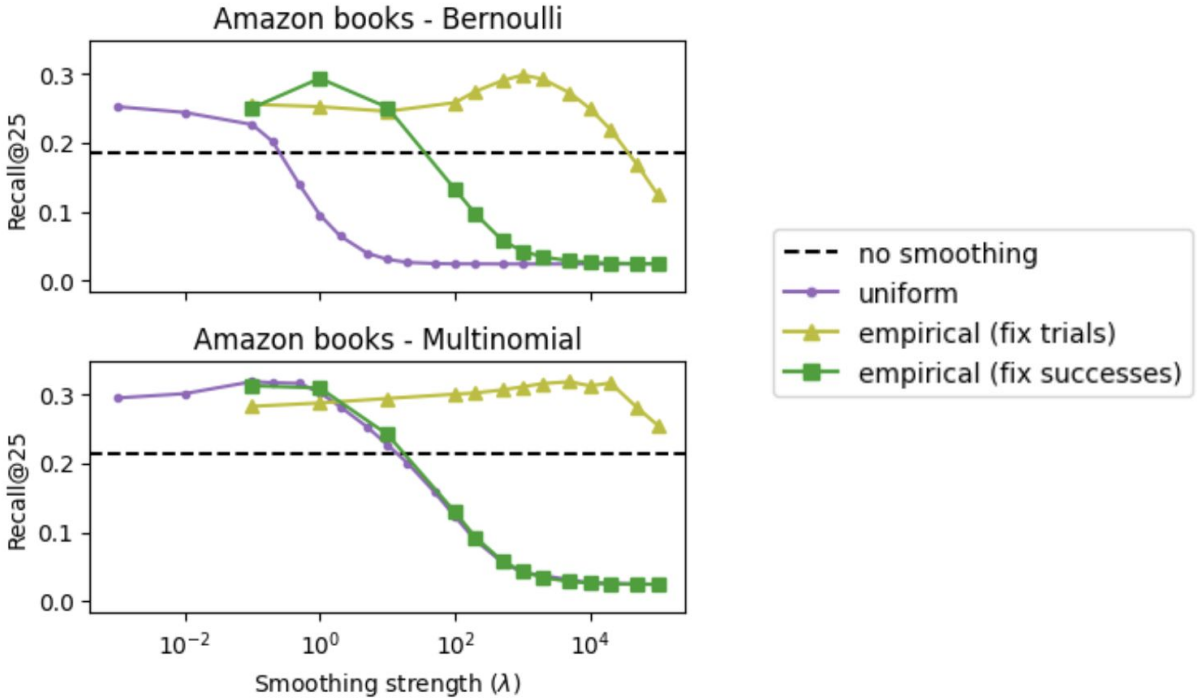
Choosing the Prior

Smoothing moves the value towards the mode of the prior

- **Additive smoothing:** pseudocounts
  - Same prior for each item: **Beta(1+λ, 1+λ)**
  - Move towards **0.5**, smoothing strength controlled by λ

- **Empirical smoothing**
  - Vary the prior based on the item: Beta with mode **P(j)**
  - Represents independence -> limits spurious correlations

Impact of Smoothing

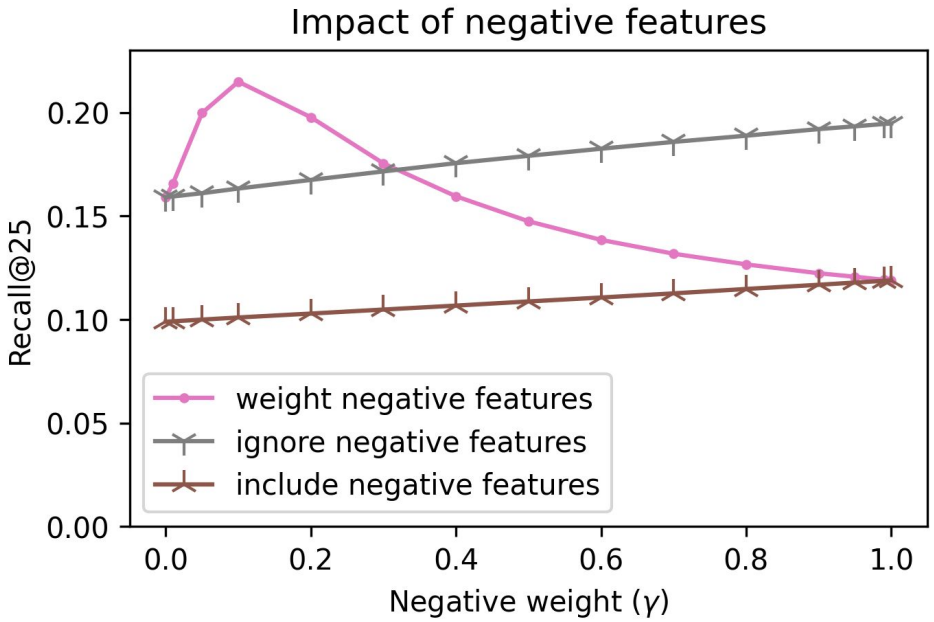


Smoothing and Negative Features

- **Smoothing solves degeneracy when γ=0**
  - Smoothing interpolates between 0.5 and 1
  - High co-occurrence count = closer to 1
  - Low co-occurrence count = closer to 0.5

$$\sigma(B_{ij}^{+}) = \frac{n(i,j) + \alpha - 1}{n(i,j) + \alpha + \beta - 2}$$

Impact of Negative Features on Performance



Some Empirical Results

- **Multinomial similarities always scored best**
  - Compared to cosine, conditional probability, and Bernoulli
  - Inherently well suited for implicit feedback
- **Weighting the negatives features improved the Bernoulli version**
- **Smoothing improves performance greatly**

- **Not better than EASE**

Revisiting Heuristics

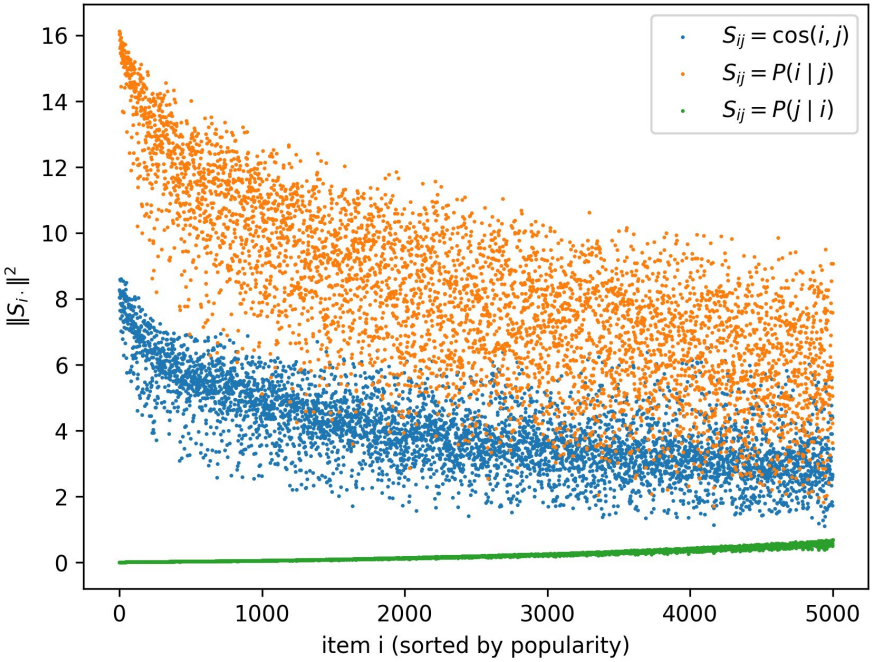
Heuristics as Optimization

$$\mathcal{L}(B \mid X) = -\sum_u \sum_i X_{ui} \sum_j X_{uj} B_{ij} + \frac{1}{2} \sum_i \sum_j \lambda(i,j) B_{ij}^2$$

- Depending on regularization different similarities are found
  - **Cosine similarity** $\lambda(i,j) = \|X_{\cdot i}\| \|X_{\cdot j}\|$
  - **Conditional Probability** $\lambda(i,j) = \|X_{\cdot j}\|^2$

- Regularization highlights popularity bias

Popularity Bias



Conclusion

Similarities as Optimization

- **Interpretability and insight**
  - Suitability of Bernoulli and multinomial for implicit feedback data
  - Smoothing as priors on the model weights
  - Popularity bias in heuristic similarities

- **Extensibility**
  - Weighting the negative features for the Bernoulli similarities
  - Empirical smoothing