Web Mining and Recommender Systems

Advanced Recommender Systems: Bayesian Personalized Ranking

Coming up

Methodological papers

- Bayesian Personalized Ranking

$$\sigma(\gamma_u \cdot \gamma_i - \gamma_u \cdot \gamma_j)$$

- Factorizing Personalized Markov Chains

$$f(u, i) + f(i, \text{previous item})$$

- Personalized Ranking Metric Embedding

$$\gamma_u \cdot \gamma_i \qquad d(\gamma_u, \gamma_i)$$

Coming up

Application papers

- Recommending Product Sizes to Customers
- Playlist Prediction via Metric Embedding
- Google's Smart Reply

Bayesian Personalized Ranking

$$\sigma(\gamma_u \cdot \gamma_i - \gamma_u \cdot \gamma_j)$$

BPR: Bayesian Personalized Ranking from Implicit Feedback

Steffen Rendle, Christoph Freudenthaler, Zeno Gantner and Lars Schmidt-Thieme
{srendle, freudenthaler, gantner, schmidt-thieme}@ismll.de
Machine Learning Lab, University of Hildesheim
Marienburger Platz 22, 31141 Hildesheim, Germany

Abstract

Item recommendation is the task of predicting a personalized ranking on a set of items (e.g. websites, movies, products). In this paper, we investigate the most common scenario with implicit feedback (e.g. clicks, purchases). There are many methods for item recommendation from implicit feedback like matrix factorization (MF) or adaptive k-nearest-neighbor (kNN). Even though these methods are designed for the item predic-

sonalization is attractive both for content providers, who can increase sales or views, and for customers, who can find interesting content more easily. In this paper, we focus on item recommendation. The task of item recommendation is to create a user-specific ranking for a set of items. Preferences of users about items are learned from the user's past interaction with the system – e.g. his buying history, viewing history, etc.

Recommender systems are an active topic of research. Most recent work is on scenarios where users provide explicit feedback, e.g. in terms of ratings. Nevertheless, in real-world scenarios most feedback is not

Bayesian Personalized Ranking

Goal: Estimate a personalized **ranking function** for each user

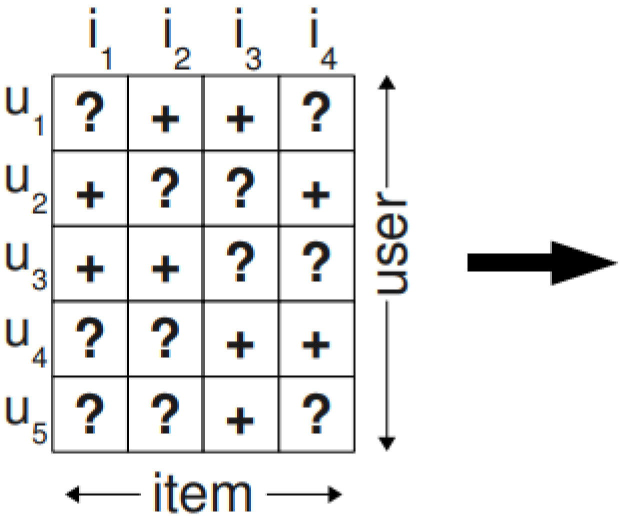
$$i >_u j$$

$x_{u,i,j} > 0$ u prefers i

≤ 0 u prefers j

Bayesian Personalized Ranking

Why? Compare to "traditional" approach of replacing "missing values" by 0:



	i_1	i_2	i_3	i_4
u_1	?	+	+	?
u_2	+	?	?	+
u_3	+	+	?	?
u_4	?	?	+	+
u_5	?	?	+	?

0	1	1	0
1	0	0	1
1	1	0	0
0	0	1	1
0	0	1	0

$\sigma(\gamma_u \cdot \gamma_i)$

But! "0"s aren't necessarily negative!

Bayesian Personalized Ranking

Why? Compare to “traditional” approach of replacing “missing values” by 0:

$$u: \quad \overbrace{+ \quad + \quad + \quad +}^{\text{likes}} \quad \overbrace{? \quad ? \quad ? \quad ? \quad ? \quad ?}^{\text{dislikes + likes but doesn't know}}$$

This suggests a possible solution based on **ranking**

Bayesian Personalized Ranking

Defn: AUC (for a user *u*)

$$\mathrm{AUC}(u) := \frac{1}{|I_u^+|\,|I \setminus I_u^+|} \sum_{\substack{i \in I_u^+ \\ \uparrow \\ \text{likes}}} \sum_{\substack{j \in |I \setminus I_u^+| \\ \uparrow \\ \text{?'s}}} \delta(\hat{x}_{uij} > 0)$$

$$\left(\mathrm{AUC} := \frac{1}{|U|} \sum_{u \in U} AUC(u) \right)$$

scoring function that compares an item *i* to an item *j* for a user *u*

The AUC essentially **counts** how many times the model correctly identifies that u prefers the item they bought (positive feedback) over the item they did not

Bayesian Personalized Ranking

Defn: AUC (for a user *u*)

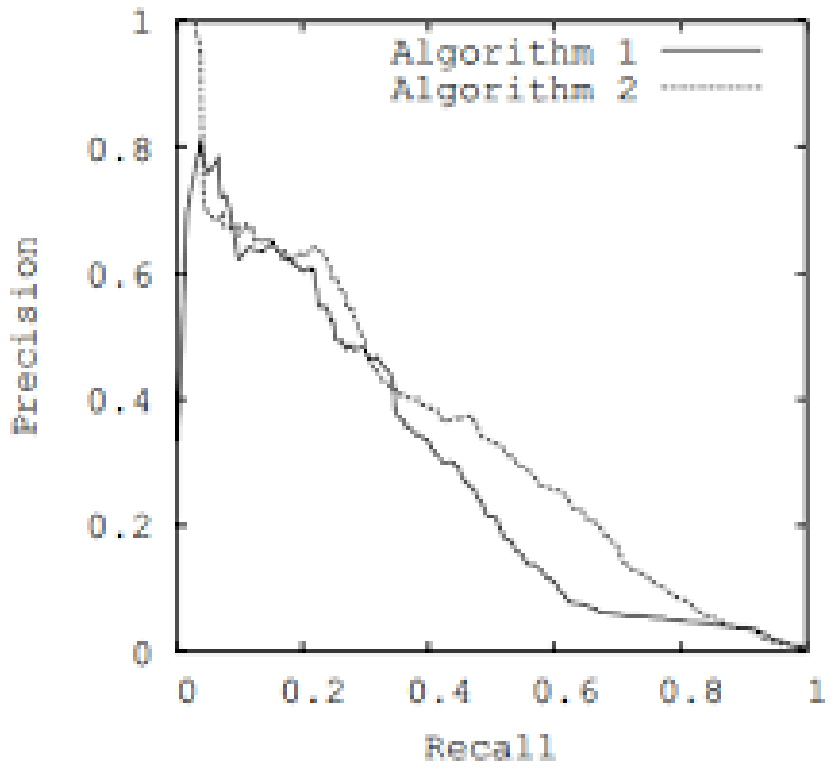
$$\mathrm{AUC}(u) := \frac{1}{|I_u^+|\,|I \setminus I_u^+|} \sum_{i \in I_u^+} \sum_{j \in |I \setminus I_u^+|} \delta(\hat{x}_{uij} > 0)$$

AUC = 1: We **always** guess correctly among two potential items *i* and *j*

AUC = 0.5: We guess **no better than random**

Bayesian Personalized Ranking

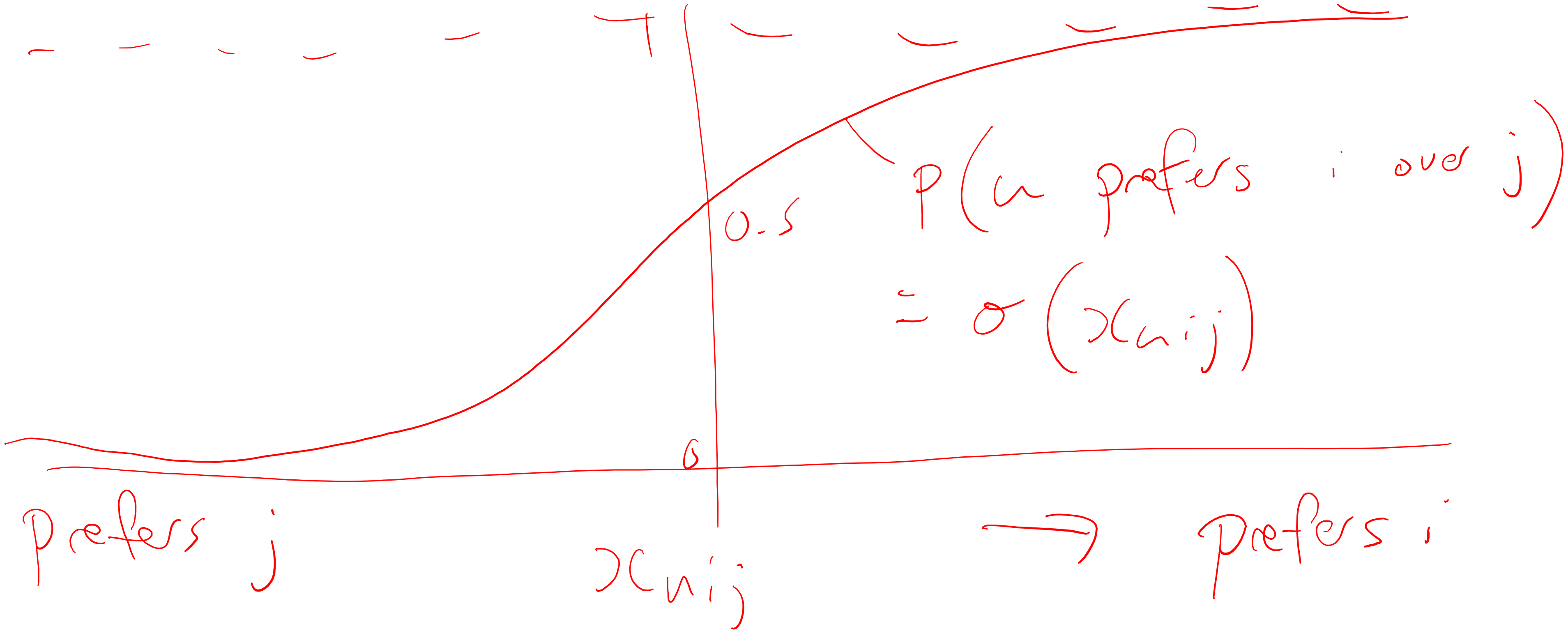
Defn: AUC
= Area Under Precision Recall Curve



Bayesian Personalized Ranking

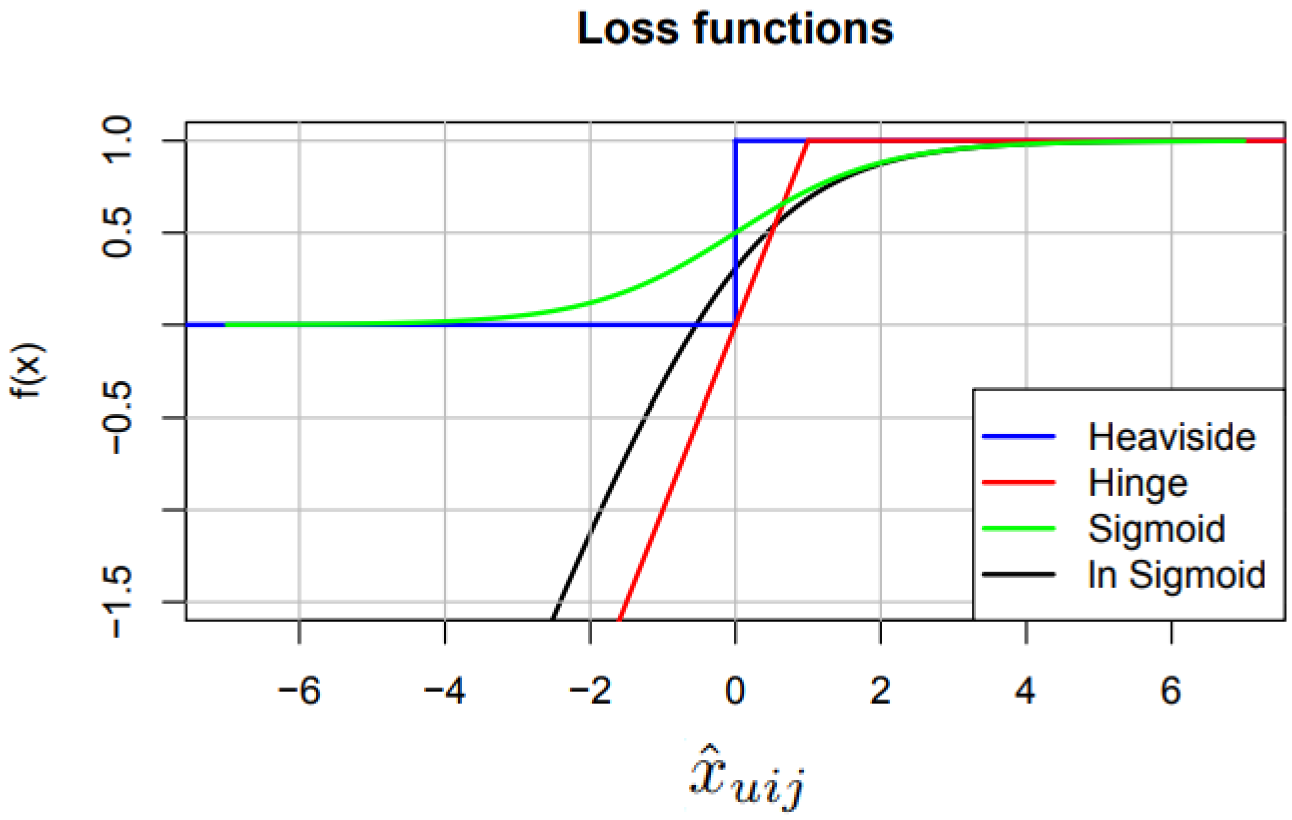
Summary: Goal is to count how many times we identified ***i*** as being more preferable than ***j*** for a user ***u***

$$\delta(\hat{x}_{uij} > 0)$$



Bayesian Personalized Ranking

Summary: Goal is to count how many times we identified ***i*** as being more preferable than ***j*** for a user ***u*** $\delta(\hat{x}_{uij} > 0)$



Bayesian Personalized Ranking

Idea: Replace the counting function $\delta(\hat{x}_{uij} > 0)$ by a smooth function

$$\sigma(\hat{x}_{uij})$$

$\hat{x}_{uij}$ is any function that compares the compatibility of *i* and *j* for a user *u*

e.g. could be based on matrix factorization:

$$\sigma(\gamma_u \cdot \gamma_i - \gamma_u \cdot \gamma_j)$$

Bayesian Personalized Ranking

Idea: Replace the counting function $\delta(\hat{x}_{uij} > 0)$ by a smooth function

$$
\begin{aligned}
\text{BPR-Opt} &:= \ln p(\Theta| >_u) \\
&= \ln p(>_u |\Theta)\, p(\Theta) \\
&= \ln \prod_{(u,i,j) \in D_S} \sigma(\hat{x}_{uij})\, p(\Theta) \\
&= \sum_{(u,i,j) \in D_S} \ln \sigma(\hat{x}_{uij}) + \ln p(\Theta) \\
&= \sum_{(u,i,j) \in D_S} \ln \sigma(\hat{x}_{uij}) - \lambda_\Theta ||\Theta||^2
\end{aligned}
$$

Bayesian Personalized Ranking

Idea: Replace the counting function $\delta(\hat{x}_{uij} > 0)$ by a smooth function

$$\arg\max_{\gamma} \sum_{uij \in T} \log \sigma(\gamma_u \cdot \gamma_i - \gamma_u \cdot \gamma_j) - \lambda \|\gamma\|_2^2$$

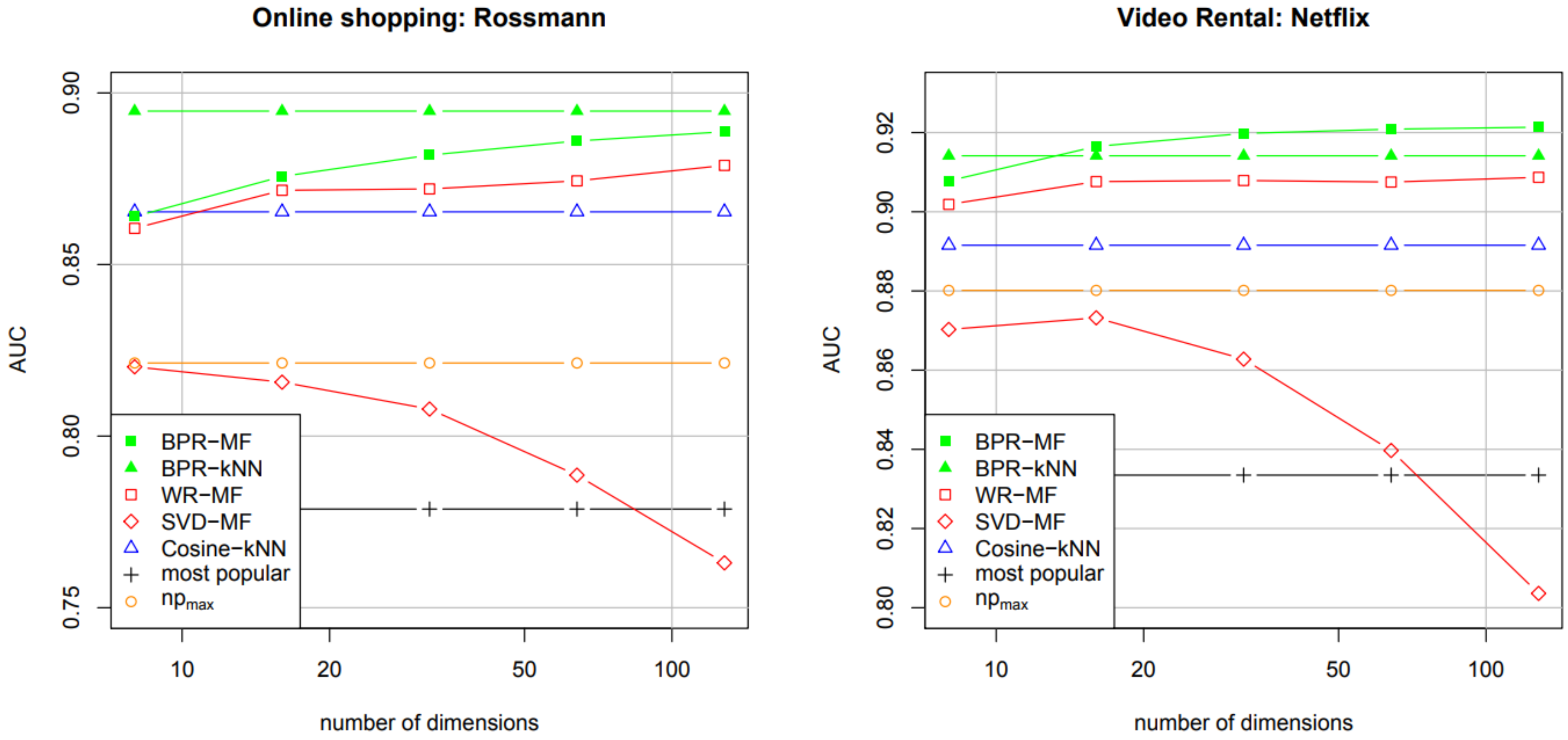
Bayesian Personalized Ranking

Experiments:

- RossMann (online drug store)
- Netflix (treated as a binary problem)

Bayesian Personalized Ranking

Experiments:



Bayesian Personalized Ranking

Morals of the story:

- Given a "one-class" prediction task (like purchase prediction) we might want to optimize a ranking function rather than trying to factorize a matrix directly
- The AUC is one such measure that counts among a users *u*, items they consumed *i*, and items they did not consume, *j*, **how often we correctly guessed that *i* was preferred by *u***
- We can optimize this approximately by maximizing $\sigma(\hat{x}_{uij})$ where $\hat{x}_{uij} = \gamma_u \cdot \gamma_i - \gamma_u \cdot \gamma_j$

Web Mining and Recommender Systems

Advanced Recommender Systems: Factorized Personalized Markov Chains

Factorizing Personalized Markov Chains for Next-Basket Recommendation

$$f(u, i) + f(i, \text{previous})$$

Factorizing Personalized Markov Chains for Next-Basket Recommendation

Steffen Rendle*
Department of Reasoning for Intelligence
The Institute of Scientific and Industrial Research
Osaka University, Japan
rendle@ar.sanken.osaka-u.ac.jp

Christoph Freudenthaler
Information Systems and Machine Learning Lab
Institute for Computer Science
University of Hildesheim, Germany
freudenthaler@ismll.uni-hildesheim.de

Lars Schmidt-Thieme
Information Systems and Machine Learning Lab
Institute for Computer Science
University of Hildesheim, Germany
schmidt-thieme@ismll.uni-hildesheim.de

ABSTRACT

Recommender systems are an important component of many websites. Two of the most popular approaches are based on matrix factorization (MF) and Markov chains (MC). MF methods learn the general taste of a user by factorizing the matrix over observed user-item preferences. On the other hand, MC methods model sequential behavior by learning a transition graph over items that is used to predict the next action based on the recent actions of a user. In this paper, we present a method bringing both approaches together. Our method is based on personalized transition graphs over un-

1. INTRODUCTION

A core technology of many recent websites are recommender systems. They are used for example to increase sales in e-commerce, clicking rates on websites or visitor satisfaction in general. In this paper, we deal with the problem setting where sequential basket data is given per user. An obvious example is an online shop where a user buys items (e.g. books or CDs). In these applications, usually several items are bought at the same time, i.e. we have a set/basket of items at one point of time. The target is now to recommend items to the user that he might want to buy in his next visit.

Factorizing Personalized Markov Chains for Next-Basket Recommendation

Goal: build **temporal** models just by looking at the item the user purchased **previously**

$$r(u, i|j)$$

(or $p_u(i|j)$)

Factorizing Personalized Markov Chains for Next-Basket Recommendation

Assumption: all of the information contained by temporal models is captured by the previous action

this is what's known as a **first-order Markov** property

Factorizing Personalized Markov Chains for Next-Basket Recommendation

Is this assumption realistic?

good

bad

song

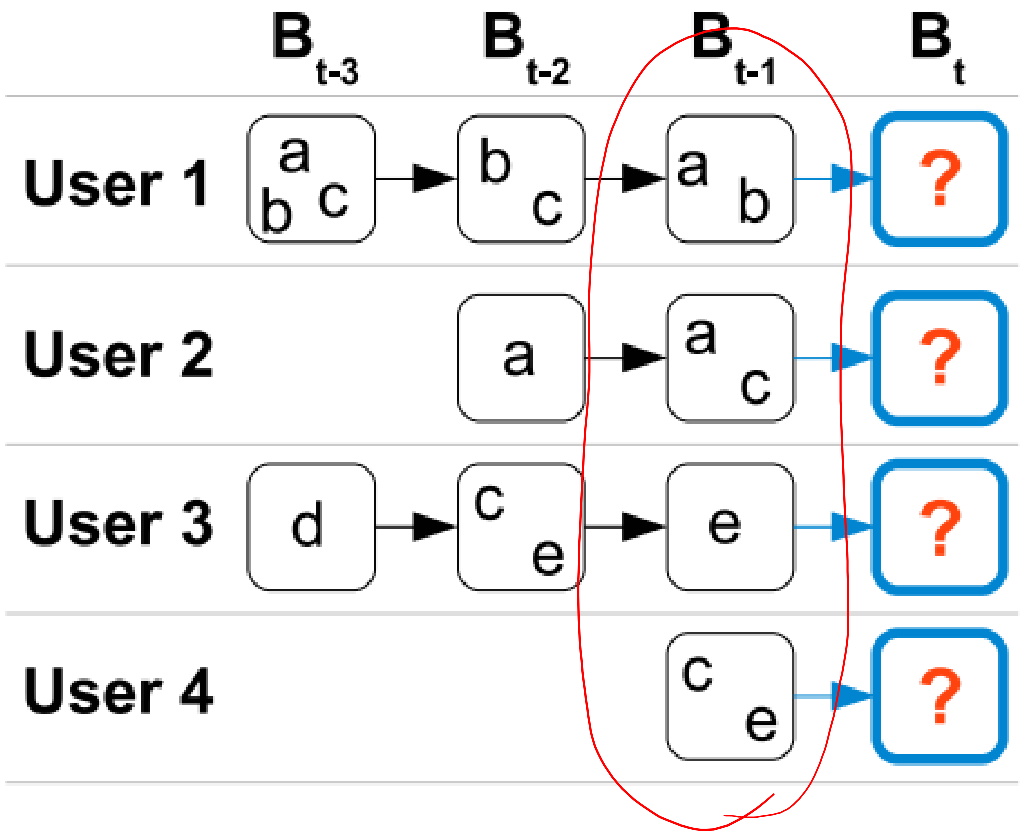
movies

groceries

electronics

Factorizing Personalized Markov Chains for Next-Basket Recommendation

Data setup: Rossmann basket data



Factorizing Personalized Markov Chains for Next-Basket Recommendation

Prediction task:

$$p(i \in B_t | B_{t-1}) := \frac{1}{|B_{t-1}|} \sum_{l \in B_{t-1}} p(i \in B_t | l \in B_{t-1})$$

i in B

previous basket

l in previous basket

$$p(B_t | B_{t-1}) \propto \prod_{i \in B_t} p(i | B_{t-1})$$

Factorizing Personalized Markov Chains for Next-Basket Recommendation

Could we try and compute such probabilities
just by **counting?**

$$\hat{a}_{l,i} = \hat{p}(i \in B_t | l \in B_{t-1}) = \frac{\hat{p}(i \in B_t \wedge l \in B_{t-1})}{\hat{p}(l \in B_{t-1})} =$$

$$= \frac{|\{(B_t, B_{t-1}) : i \in B_t \wedge l \in B_{t-1}\}|}{|\{(B_t, B_{t-1}) : l \in B_{t-1}\}|}$$

$$\frac{i \text{ in } B \wedge l \text{ in previous}}{l \text{ in previous}}$$

Seems okay, as long as the item vocabulary is small
(I^2 possible item/item combinations to count)

But it's not **personalized**

Factorizing Personalized Markov Chains for Next-Basket Recommendation

What if we try to **personalize?**

$$\hat{a}_{u,l,i} = \hat{p}(i \in B_t^u | l \in B_{t-1}^u) = \frac{\hat{p}(i \in B_t^u \wedge l \in B_{t-1}^u)}{\hat{p}(l \in B_{t-1}^u)}$$

$$= \frac{|\{(B_t^u, B_{t-1}^u) : i \in B_t^u \wedge l \in B_{t-1}^u\}|}{|\{(B_t^u, B_{t-1}^u) : l \in B_{t-1}^u\}|}$$

same but for user u

Now we would have U*I^2 counts to compare

Clearly not feasible, so we need to try and estimate/model this quantity (e.g. by matrix factorization)

Factorizing Personalized Markov Chains for Next-Basket Recommendation

What if we try to
personalize?

$$\hat{A} := \mathcal{C} \times_U V^U \times_L V^L \times_I V^I$$

$$\mathcal{C} \in \mathbb{R}^{k_U, k_L, k_I}, \quad V^U \in \mathbb{R}^{|U| \times k_U},$$
$$V^L \in \mathbb{R}^{|I| \times k_L}, \quad V^I \in \mathbb{R}^{|I| \times k_I}$$

Factorizing Personalized Markov Chains for Next-Basket Recommendation

What if we try to
personalize?

$$\hat{a}_{u,l,i} := \sum_{f=1}^{k_{U,I}} v^{U,I}_{u,f}\, v^{I,U}_{i,f} + \sum_{f=1}^{k_{I,L}} v^{I,L}_{i,f}\, v^{L,I}_{l,f} + \sum_{f=1}^{k_{U,L}} v^{U,L}_{u,f}\, v^{L,U}_{l,f}$$

$$a_{uli} = \gamma_u \cdot \gamma_i + \gamma_i \cdot \gamma_l + \gamma_u \cdot \gamma_l$$

Factorizing Personalized Markov Chains for Next-Basket Recommendation

Prediction task:

$$p(i \in B_t | B_{t-1}) := \frac{1}{|B_{t-1}|} \sum_{l \in B_{t-1}} p(i \in B_t | l \in B_{t-1})$$

$$\hat{p}(i \in B_t^u | B_{t-1}^u) = \frac{1}{|B_{t-1}^u|} \sum_{l \in B_{t-1}^u} \hat{a}_{u,l,i}$$

$$= \frac{1}{|B_{t-1}^u|} \sum_{l \in B_{t-1}^u} (\langle v_u^{U,I}, v_i^{I,U} \rangle + \langle v_i^{I,L}, v_l^{L,I} \rangle$$

$$+ \langle v_u^{U,L}, v_l^{L,U} \rangle)$$

Factorizing Personalized Markov Chains for Next-Basket Recommendation

Prediction task:

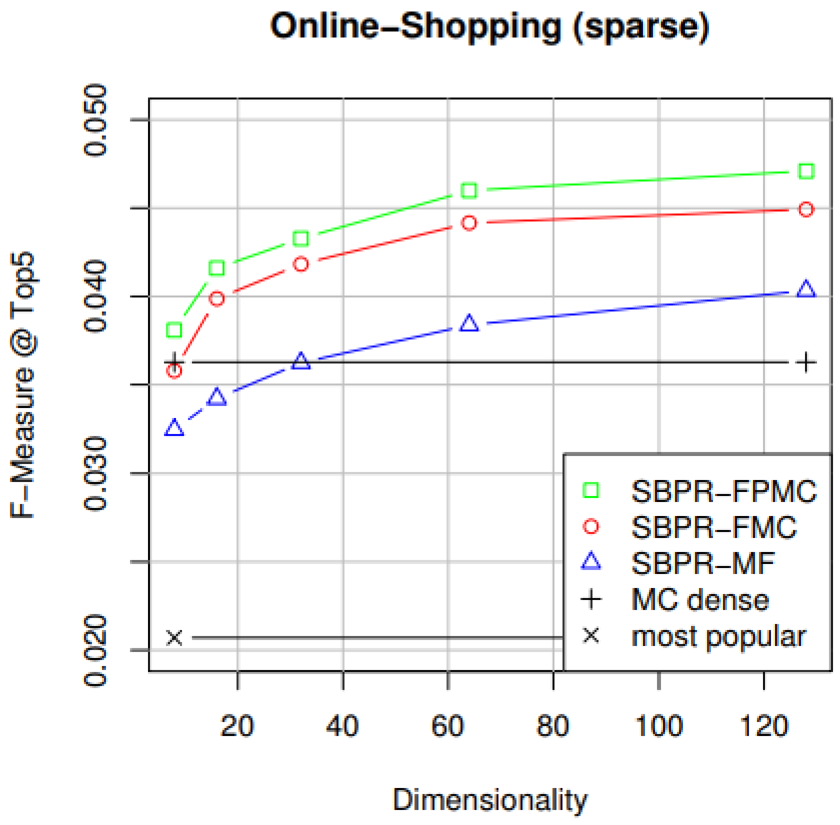
$$\underset{\Theta}{\operatorname{argmax}} \prod_{u \in U} \prod_{B_t \in \mathcal{B}^u} p(>_{u,t} | \Theta) \, p(\Theta)$$

$$\prod_{u \in U} \prod_{B_t \in \mathcal{B}^u} \prod_{i \in B_t} \prod_{j \notin B_t} p(i >_{u,t} j | \Theta)$$

$$x_{uijl} = \boxed{\gamma_u \cdot \gamma_i + \gamma_i \cdot \gamma_l - \gamma_u \cdot \gamma_j - \gamma_j \cdot \gamma_j} + \cancel{\gamma_u \cdot \gamma_l} - \cancel{\gamma_u \cdot \gamma_l}$$

Factorizing Personalized Markov Chains for Next-Basket Recommendation

F@5



FMC: not personalized

$\gamma_i \cdot \gamma_l$

MF: personalized, but not sequentially-aware

$\gamma_u \cdot \gamma_i$

Factorizing Personalized Markov Chains for Next-Basket Recommendation

Morals of the story:

- Can improve performance by modeling **third order** interactions between the user, the item, and the previous item
- This is simpler than temporal models – but makes a big assumption
- Given the blowup in the interaction space, this can be handled by **tensor decomposition** techniques

Web Mining and Recommender Systems

Advanced Recommender Systems:
Personalized Ranking Metric Embedding

Personalized Ranking Metric Embedding for Next New POI Recommendation

Personalized Ranking Metric Embedding for Next New POI Recommendation

Shanshan Feng,[1] Xutao Li,[2] Yifeng Zeng,[3] Gao Cong,[2] Yeow Meng Chee,[4] Quan Yuan[2]

[1]Interdisciplinary Graduate School,
Nanyang Technological University, Singapore, sfeng003@e.ntu.edu.sg
[2]School of Computer Engineering, Nanyang Technological University, Singapore,
{lixutao@, gaocong@, qyuan1@e.}ntu.edu.sg
[3]School of Computing, Teesside University, UK, Y.Zeng@tees.ac.uk
[4]School of Physical and Mathematical Sciences,
Nanyang Technological University, Singapore, ymchee@ntu.edu.sg

Abstract

The rapidly growing of Location-based Social Networks (LBSNs) provides a vast amount of check-in data, which enables many services, e.g., point-of-interest (POI) recommendation. In this paper, we study the *next new* POI recommendation problem in which *new* POIs with respect to users' current location are to be recommended. The challenge lies in the difficulty in precisely learning users' sequential information and personalizing the recommendation model. To this end, we resort to the Metric Embedding method for the recommendation, which avoids drawbacks of the Matrix Factorization technique. We propose a personalized ranking metric embedding method (PRME) to model personalized check-in sequences. We further develop a PRME-G

mation of users' check-ins. The sequential behavior is important for POI recommendation because human movement exhibits sequential patterns [Ye *et al.*, 2013]. We verify users' sequential behavior in the analysis of two real-world datasets. Meanwhile, we observe that users often visit *new* POIs that they have not been visited before. In this paper, we focus on the *Next New* POI recommendation problem (simplified as N^2-POI recommendation), which is to recommend *new* POIs to be visited *next* given a user's current location.

The challenge of N^2-POI recommendation is to learn transitions of users' check-ins that are commonly represented by a first-order Markov chain model. Due to the sparse transition data, it is difficult to estimate the transition probability in Markov chain, especially for the unobserved transition. Factorized Personalized Markov Chain (FPMC) [Rendle *et al.*, 2010] method has been used to calculate the item transitions. FPMC exploits matrix factorization technique to factorize the

Personalized Ranking Metric Embedding for Next New POI Recommendation

Goal: Can we build better sequential recommendation models by using the idea of **metric embeddings**

$$\gamma_u \cdot \gamma_i \qquad \text{vs.} \qquad d(\gamma_u, \gamma_i)$$

Personalized Ranking Metric Embedding for Next New POI Recommendation

Why would we expect this to work (or not)?

Personalized Ranking Metric Embedding for Next New POI Recommendation

Otherwise, goal is the same as the previous paper:

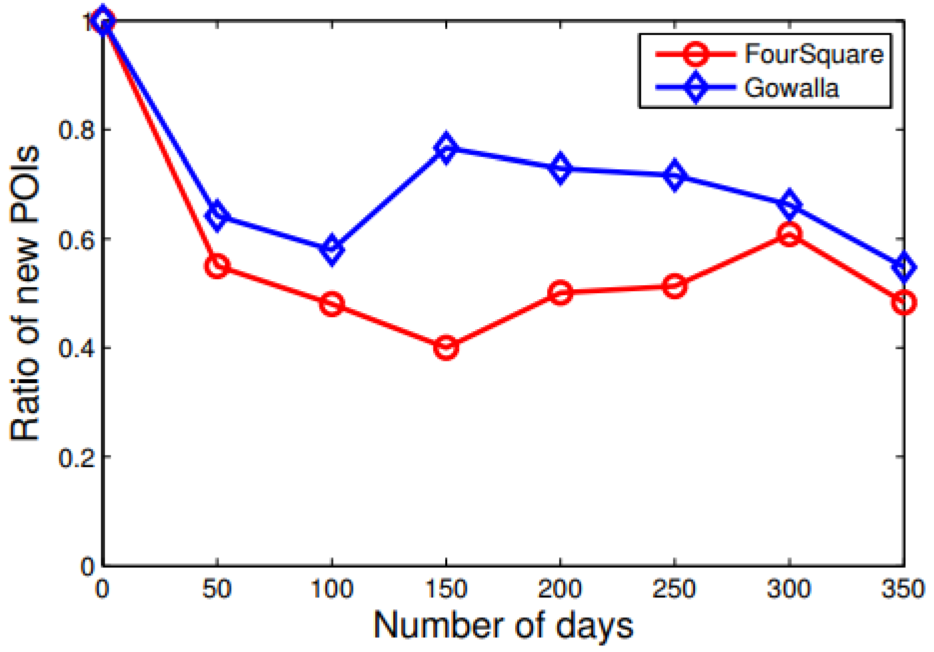
$$p_u(i|j)$$

Personalized Ranking Metric Embedding for Next New POI Recommendation

Data

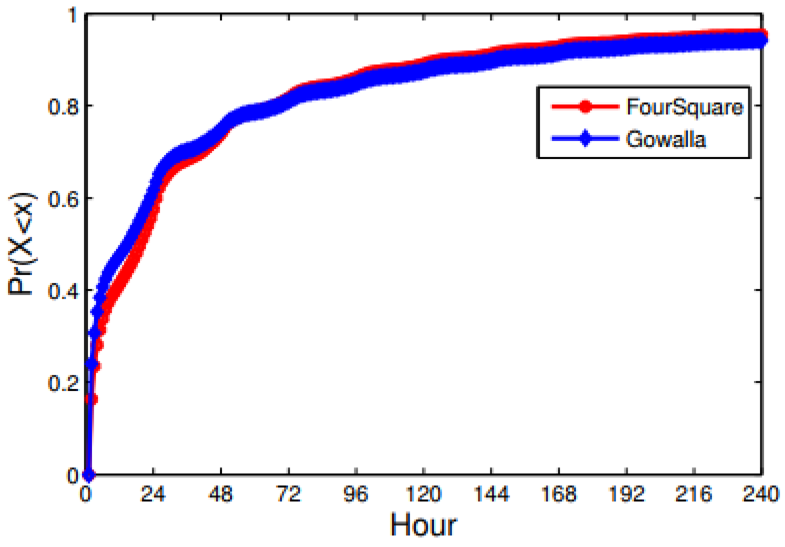
Dataset	#User	#POI	#Check-in	Time range
FourSquare	1917	2675	155365	08/2010-07/2011
Gowalla	4996	6871	245157	11/2009-10/2010

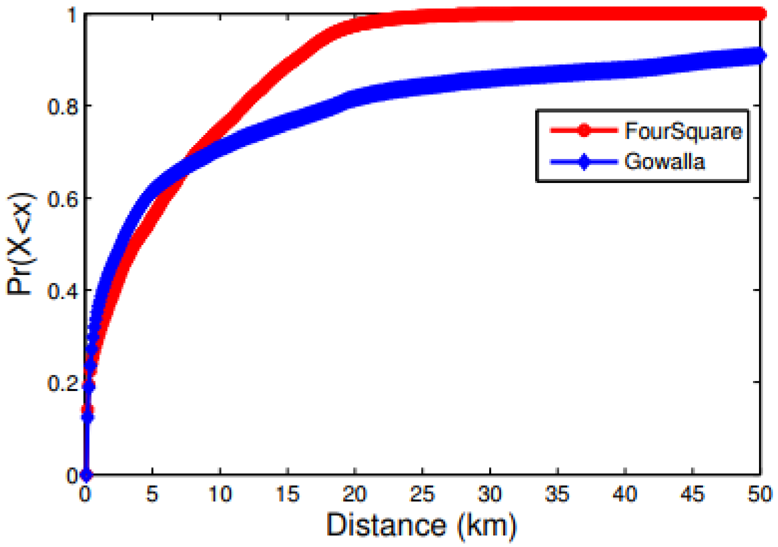
Qualitative analysis



Personalized Ranking Metric Embedding for Next New POI Recommendation

Qualitative analysis





Personalized Ranking Metric Embedding for Next New POI Recommendation

Basic model (not personalized)

$$\hat{P}(l_j|l_i) = \frac{e^{-||X(l_j)-X(l_i)||^2}}{Z(l_i)}$$

Personalized Ranking Metric Embedding for Next New POI Recommendation

Basic model (not personalized)

$$l_i >_{l^c} l_j \Leftrightarrow \hat{P}(l_i|l^c) > \hat{P}(l_j|l^c)$$

Personalized Ranking Metric Embedding for Next New POI Recommendation

Personalized version

$$\mathcal{D}_{u,l^c,l} = \alpha \mathcal{D}^P_{u,l} + (1-\alpha)\mathcal{D}^S_{l^c,l}$$

Personalized Ranking Metric Embedding for Next New POI Recommendation

Personalized version

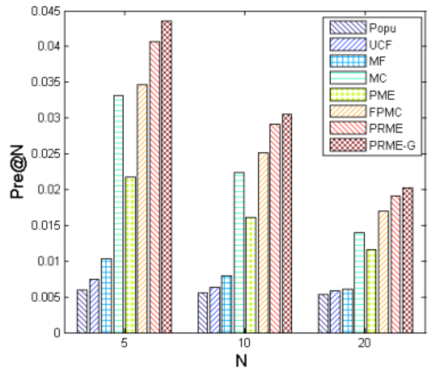
$$\mathcal{D}_{u,l^c,l} = \begin{cases} \mathcal{D}^P_{u,l} & \text{if } \Delta(l, l^c) > \tau \\ \alpha \mathcal{D}^P_{u,l} + (1-\alpha)\mathcal{D}^S_{l^c,l} & \text{otherwise} \end{cases}$$

Personalized Ranking Metric Embedding for Next New POI Recommendation

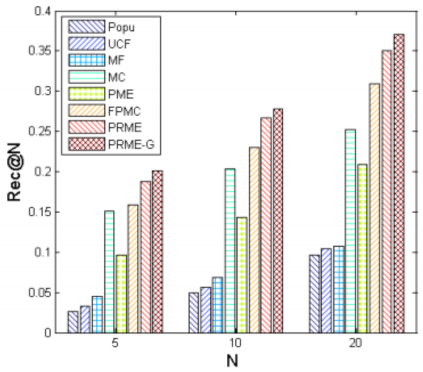
Learning

$$P(>_{u,l^c}|\Theta) = P\left((\mathcal{D}_{u,l^c,l_j} - \mathcal{D}_{u,l^c,l_i}) > 0|\Theta\right)$$
$$= \sigma(\mathcal{D}_{u,l^c,l_j} - \mathcal{D}_{u,l^c,l_i})$$

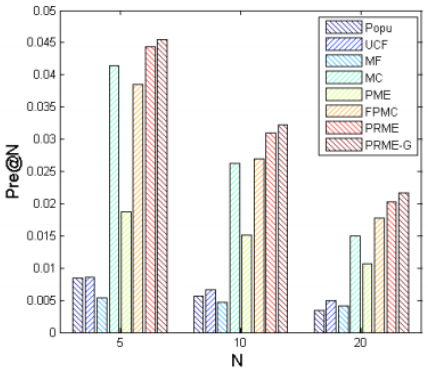
Results



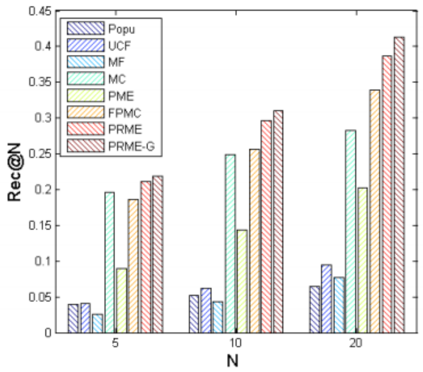
(a) Precision on FourSquare



(b) Recall on FourSquare



(c) Precision on Gowalla



(d) Recall on Gowalla

Personalized Ranking Metric Embedding for Next New POI Recommendation

Morals of the story:

- In some applications, **metric embeddings** might be better than inner products
- Examples could include geographical data, but also others (e.g. playlists?)

Overview

Morals of the story:

- Today we looked at two main ideas that extend the recommender systems we saw in class:

1. **Sequential Recommendation:** Most of the dynamics due to time can be captured purely by knowing the *sequence* of items

2. **Metric Recommendation:** In some settings, using inner products may not be the correct assumption

Web Mining and Recommender Systems

Real-world applications of recommender systems:
Recommending Product Sizes

Recommending product sizes to customers

Recommending Product Sizes to Customers

Vivek Sembium
Amazon Development Center India
viveksem@amazon.com

Rajeev Rastogi
Amazon Development Center India
rastogi@amazon.com

Atul Saroop
Amazon Development Center India
asaroop@amazon.com

Srujana Merugu*
srujana@gmail.com

ABSTRACT

We propose a novel latent factor model for recommending product size fits {`Small`, `Fit`, `Large`} to customers. Latent factors for customers and products in our model correspond to their physical true size, and are learnt from past product purchase and returns data. The outcome for a customer, product pair is predicted based on the difference between customer and product true sizes, and efficient algorithms are proposed for computing customer and product true size values that minimize two loss function variants. In experiments with Amazon shoe datasets, we show that our latent factor models incorporating personas, and leveraging return codes show a 17-21% AUC improvement compared to baselines. In an online A/B test, our algorithms show an improvement of 0.49% in percentage of Fit transactions over control.

In the size recommendation problem, a customer implicitly provides the context of a desired product by viewing the detail page of a product and requires a recommendation for the appropriate size variant of the product. For example, the customer might be viewing the detail page of Nike Women's Tennis Classic shoe and needs to choose from 10 different size variants corresponding to sizes from 6 to 15. Thus, given the context of a desired product, our objective is to recommend the appropriate size variant for a customer.

The problem of recommending sizes to customers is challenging due to the following reasons:

- *Data sparsity.* Typically, a small fraction of customers and products account for the bulk of purchases. A majority of customers and products have very few purchases.
- *Cold start.* The environment is highly dynamic with new customers and products (that have no past purchases) for

Recommending product sizes to customers

Goal: Build a recommender system that predicts whether an item will "fit":

$$(u, i) \rightarrow \{\text{small}, \text{fit}, \text{large}\}$$

Recommending product sizes to customers

Challenges:

- **Data sparsity:** people have very few purchases from which to estimate size
- **Cold-start:** How to handle new customers and products with no past purchases?
- **Multiple personas:** Several customers may use the same account

Recommending product sizes to customers

Data:

- Shoe transactions from Amazon.com

- For each shoe *j*, we have a reported size *c_j* (from the manufacturer), but this may not be correct!

- Need to estimate the customer's size *(s_i)*, as well as the product's **true** size *(t_j)*

Recommending product sizes to customers

Loss function:

$$f_w(s_i, t_j) + b$$

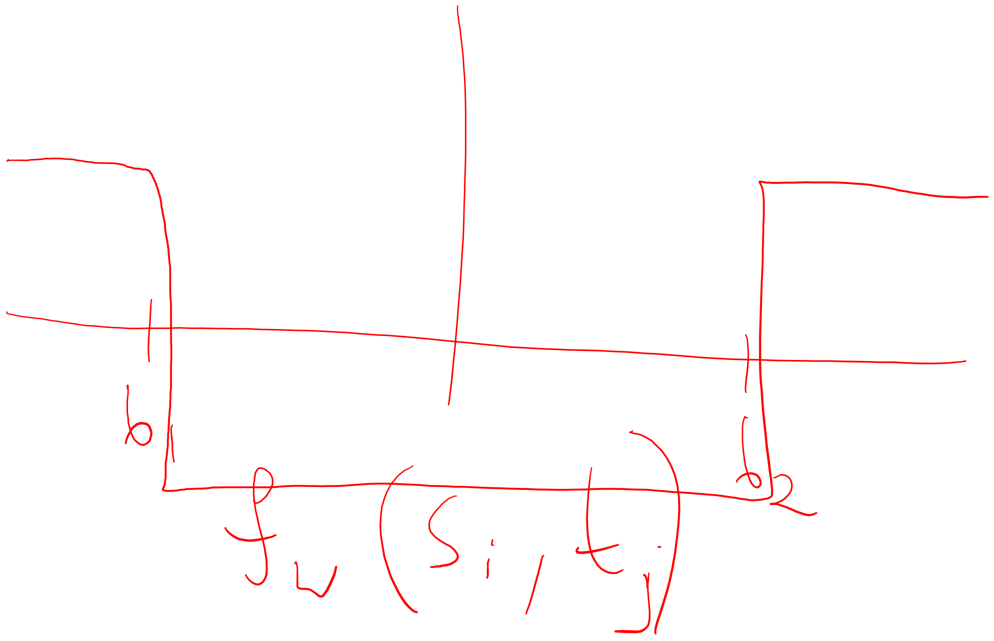
$$f_w(s_i, t_j) = w \cdot (s_i - t_j)$$

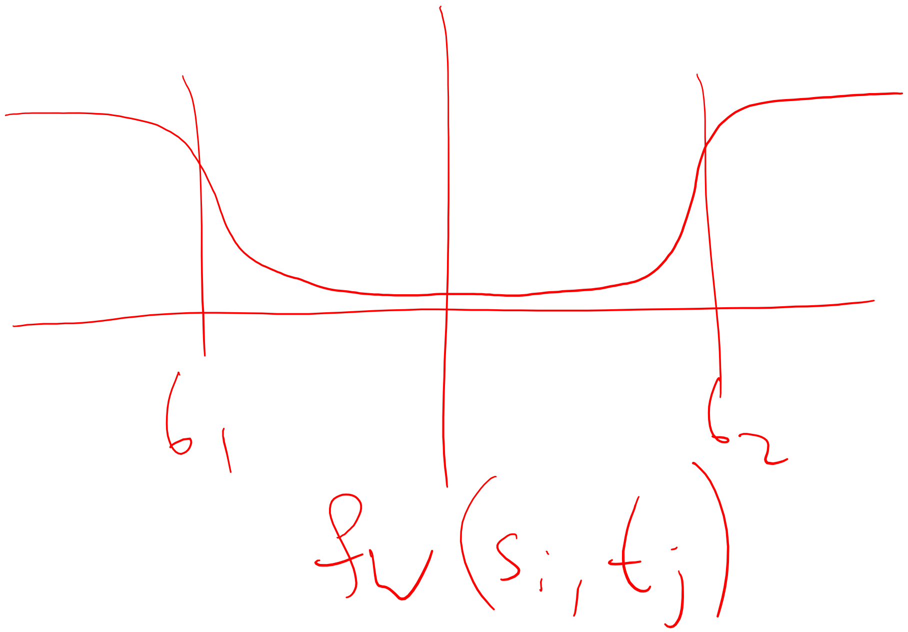
Recommending product sizes to customers

Loss function:

$$L(y_{ij}, f_w(s_i,t_j)) = \begin{cases} L^{bin}(+1, f_w(s_i,t_j) - b_2) & \text{if } y_{ij} = \texttt{Small} \\ (L^{bin}(-1, f_w(s_i,t_j) - b_2) \\ +L^{bin}(+1, f_w(s_i,t_j) - b_1)) & \text{if } y_{ij} = \texttt{Fit} \\ L^{bin}(-1, f_w(s_i,t_j) - b_1) & \text{if } y_{ij} = \texttt{Large} \end{cases}$$

$$L(y_{ij}, f_w(s_i,t_j)) = \begin{cases} \log(\frac{1}{1+e^{-f_w(s_i,t_j)+b_2}}) & \text{if } y_{ij} = \texttt{Small} \\ (\log(\frac{1}{1+e^{f_w(s_i,t_j)-b_2}}) \\ +\log(\frac{1}{1+e^{-f_w(s_i,t_j)+b_1}})) & \text{if } y_{ij} = \texttt{Fit} \\ \log(\frac{1}{1+e^{f_w(s_i,t_j)-b_1}}) & \text{if } y_{ij} = \texttt{Large} \end{cases}$$

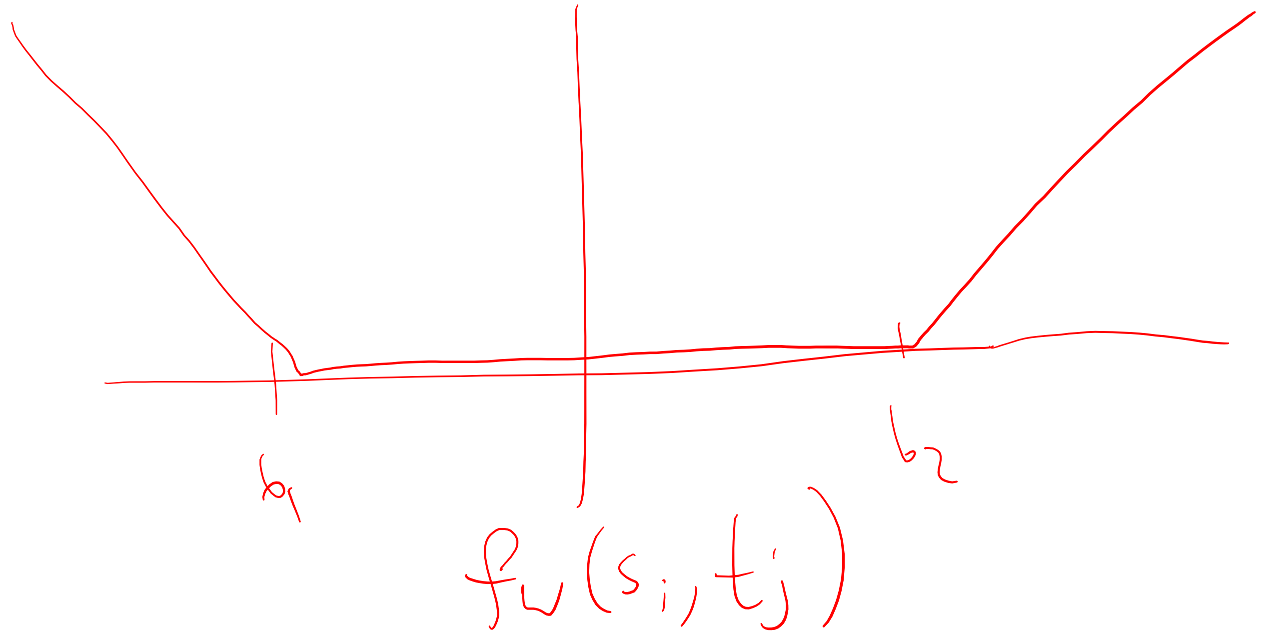




Recommending product sizes to customers

Loss function:

$$L(y_{ij}, f_w(s_i, t_j)) = \begin{cases} \max\{0, 1 - f_w(s_i, t_j) + b_2)\} & \text{if } y_{ij} = \texttt{Small} \\ (\max\{0, 1 + f_w(s_i, t_j) - b_2)\} & \\ + \max\{0, 1 - f_w(s_i, t_j) + b_1)\}) & \text{if } y_{ij} = \texttt{Fit} \\ \max\{0, 1 + f_w(s_i, t_j) - b_1)\} & \text{if } y_{ij} = \texttt{Large} \end{cases}$$



Recommending product sizes to customers

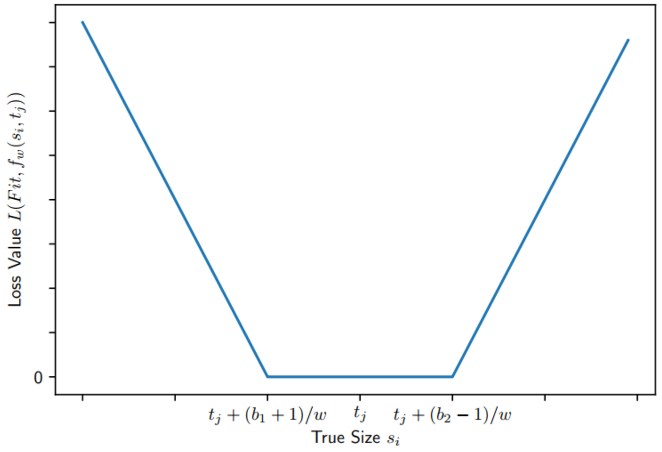


Figure 1: Hinge loss value for a `Fit` transaction vs s_i.

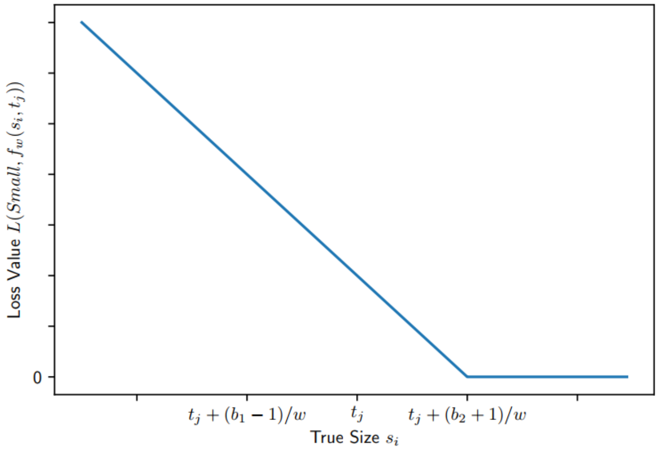


Figure 2: Hinge loss value for a `Small` transaction vs s_i.

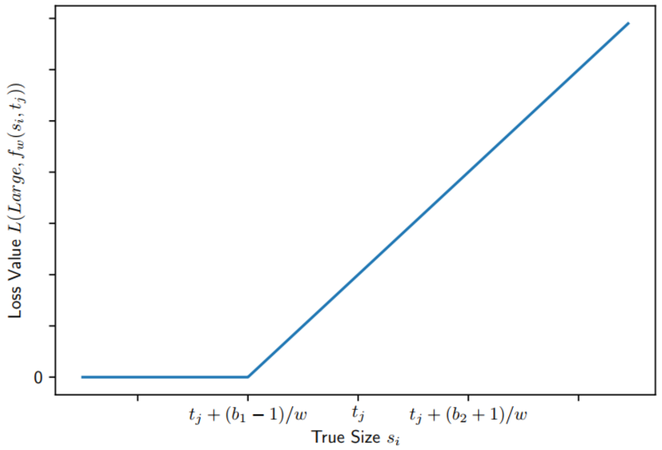


Figure 3: Hinge loss value for a `Large` transaction vs s_i.

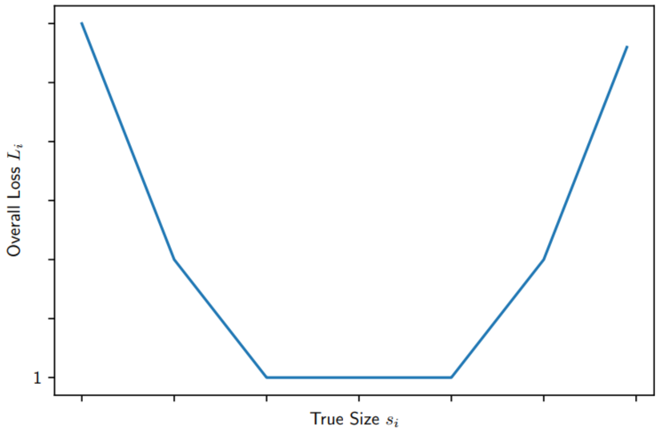


Figure 4: Illustrative overall hinge loss vs s_i.

Recommending product sizes to customers

Loss function:

$$
\begin{aligned}
\mathcal{L}_i = & \sum_{(i,j,y_{ij}) \in \mathcal{D} \wedge y_{ij} = \texttt{Small}} \max\{0, 1 - f_w(s_i, t_j) + b_2\} \\
& + \sum_{(i,j,y_{ij}) \in \mathcal{D} \wedge y_{ij} = \texttt{Fit}} (\max\{0, 1 + f_w(s_i, t_j) - b_2\} \\
& \qquad + \max\{0, 1 - f_w(s_i, t_j) + b_1\}) \\
& + \sum_{(i,j,y_{ij}) \in \mathcal{D} \wedge y_{ij} = \texttt{Large}} \max\{0, 1 + f_w(s_i, t_j) - b_1\}
\end{aligned}
$$

Recommending product sizes to customers

Model fitting:

$$\sim \quad \alpha + \beta_n + \beta_i$$

init: $t_j = c_j$

① fix t_j, update s_i

② fix s_i, update t_j

Recommending product sizes to customers

Extensions:

- *Multi-dimensional sizes*

$$w_1(s_{i_1} - t_{j_1}) + w_2(s_{i_2} - t_{j_2})$$

- *Customer and product features*

$$w(s_i - t_j) + \phi(x, i)w'$$

- *User personas*

clustering

Recommending product sizes to customers

Experiments:

Dataset	Baseline RF	Baseline Persona Linear	Baseline Persona RF	Algorithm 1 Linear	Algorithm 1 RF	Algorithm 1 Persona Linear	Algorithm 1 Persona RF
A	0.6%	0%	0%	16.4%	17.2%	17.9%	**18.1%**
B	2.1%	0.4%	2.1%	20.3%	21.3%	**21.3%**	20.5%
C	3.8%	1.9%	1.9%	15.7%	16.1%	**18.4%**	17.8%
D	2.7%	1.2%	1.6%	20.0%	20.2%	**21.3%**	20.7%
E	1.5%	0.2%	0.6%	15.8%	15.6%	**17.4%**	**17.4%**
F	2.5%	2.7%	2.3%	18.1%	17.3%	**18.5%**	17.3%

categories

Recommending product sizes to customers

Experiments:

Online A/B test

Recommending product sizes to customers

Morals of the story:

- Very simple model that actually works well in production
- Only a single parameter per user and per item!

Web Mining and Recommender Systems

Real-world applications of recommender systems:
Playlist Prediction via Metric Embedding

Playlist prediction via Metric Embedding

1 $d(\gamma_u, \gamma_i)$ vs $\gamma_u \cdot \gamma_i$ 2 seq. recsys

Playlist Prediction via Metric Embedding

Shuo Chen
Cornell University
Dept. of Computer Science
Ithaca, NY, USA
shuochen@cs.cornell.edu

Joshua L. Moore
Cornell University
Dept. of Computer Science
Ithaca, NY, USA
jlmo@cs.cornell.edu

Douglas Turnbull
Ithaca College
Dept. of Computer Science
Ithaca, NY, USA
dturnbull@ithaca.edu

Thorsten Joachims
Cornell University
Dept. of Computer Science
Ithaca, NY, USA
tj@cs.cornell.edu

ABSTRACT

Digital storage of personal music collections and cloud-based music services (e.g. Pandora, Spotify) have fundamentally changed how music is consumed. In particular, automatically generated playlists have become an important mode of accessing large music collections. The key goal of automated playlist generation is to provide the user with a *coherent* listening experience. In this paper, we present Latent Markov Embedding (LME), a machine learning algorithm for generating such playlists. In analogy to matrix factorization methods for collaborative filtering, the algorithm does not require songs to be described by features a priori, but it learns a representation from example playlists. We formulate this problem as a regularized maximum-likelihood embedding of Markov chains in Euclidian space, and show how

addition, when using a cloud-based service like Rhapsody or Spotify, the consumer has instant on-demand access to millions of songs. This has created substantial interest in automatic playlist algorithms that can help consumers explore large collections of music. Companies like Apple and Pandora have developed successful commercial playlist algorithms, but relatively little is known about how these algorithms work and how well they perform in rigorous evaluations.

Despite the large commercial demand, comparably little scholarly work has been done on automated methods for playlist generation (e.g., [13, 4, 9, 11]), and the results to date indicate that it is far from trivial to operationally define what makes a playlist coherent. The most comprehensive study was done by [11]. Working under a model where

Playlist prediction via Metric Embedding

Goal: Build a recommender system that recommends sequences of songs

Idea: Might also use a metric embedding (consecutive songs should be “nearby” in some space)

Playlist prediction via Metric Embedding

Basic model:

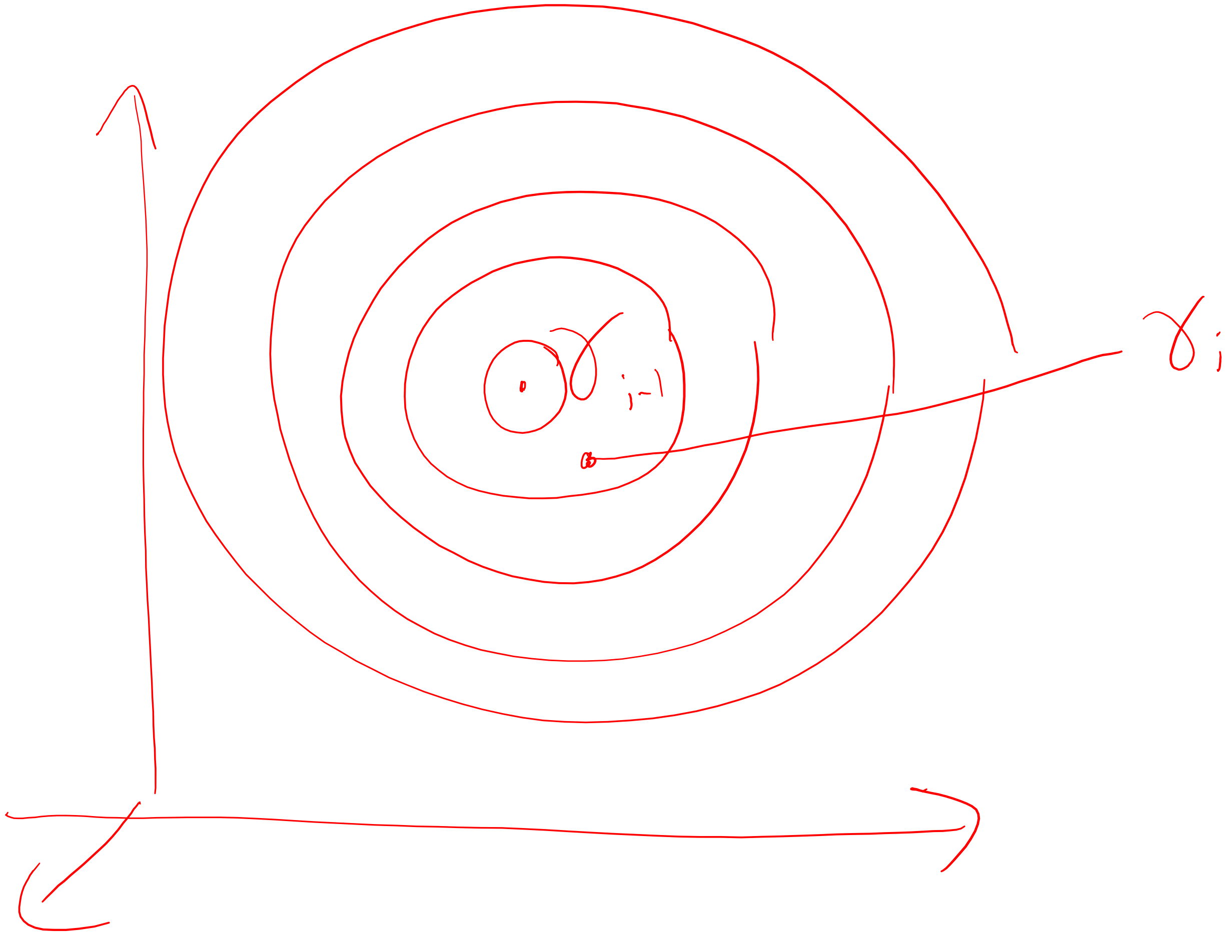
$$\Pr(p^{[i]}|p^{[i-1]}) = \frac{e^{-||X(p^{[i]})-X(p^{[i-1]})||_2^2}}{\sum_{j=1}^{|S|} e^{-||X(s_j)-X(p^{[i-1]})||_2^2}}$$

(compare with metric model from previous lecture)

$$d(i, i-1) = ||\gamma_i - \gamma_{i-1}||_2^2$$

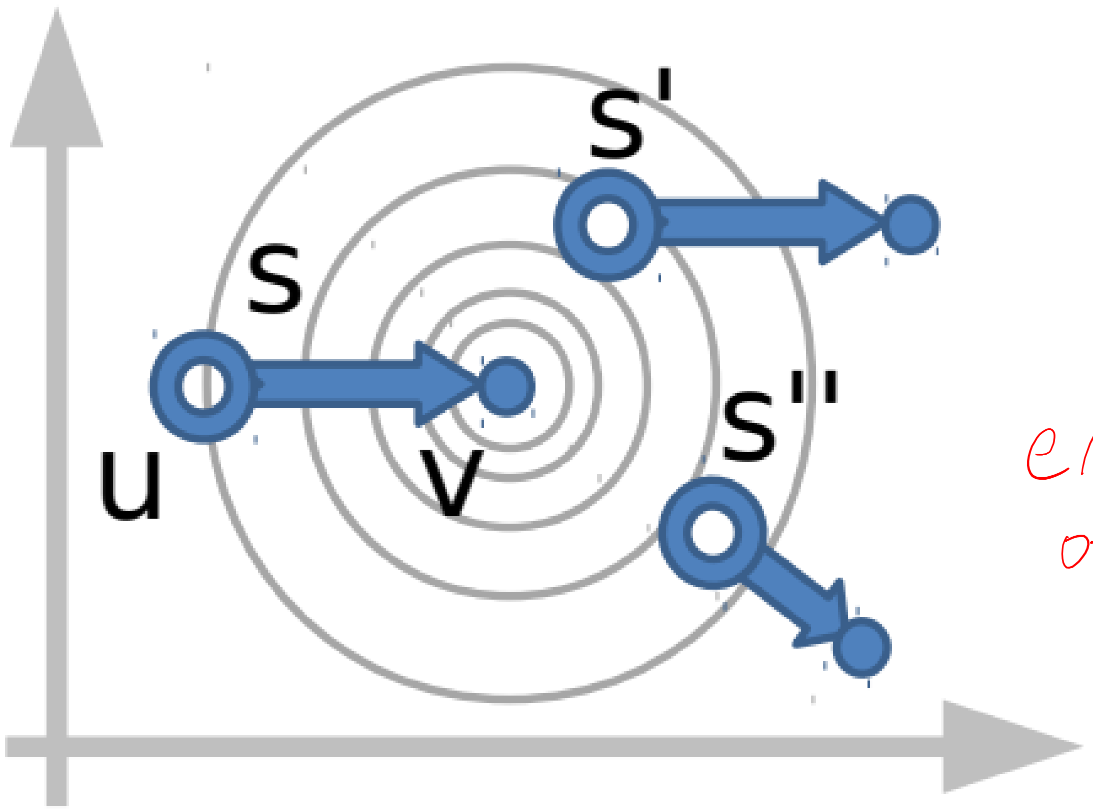
Playlist prediction via Metric Embedding

Basic model ("single point"):



Playlist prediction via Metric Embedding

"Dual-point" model



$$||X(s) - X(s')||_2$$

$$||V(s) - U(s')||_2$$

end point of prev

start point of next

Playlist prediction via Metric Embedding

Extensions:

- Popularity biases

$$\Pr(p^{[i]}|p^{[i-1]}) = \frac{e^{-\Delta(p^{[i]},p^{[i-1]})^2+b_i}}{\sum_j e^{-\Delta(s_j,p^{[i-1]})^2+b_j}}$$

more popular =
more likely to be in list

Extensions:

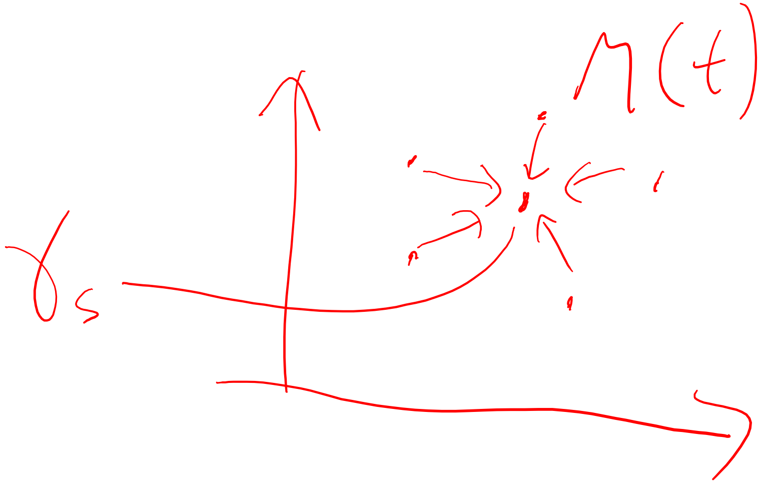
- Personalization

$$\Pr(p^{[i]} | p^{[i-1]}, u) = \frac{e^{-\Delta(p^{[i]}, p^{[i-1]})^2 + A(p^{[i]})^T B(u)}}{\sum_j e^{-\underbrace{\Delta(s_j, p^{[i-1]})^2}_{d(\gamma_i, \gamma_{i-1})} + \underbrace{A(s_j)^T B(u)}_{\gamma_u \cdot \gamma_i}}}$$

Extensions:

- Semantic Tags

$$\Pr(X(s)|T(s)) = \mathcal{N}\left(\frac{1}{|T(s)|}\sum_{t \in T(s)} M(t),\ \frac{1}{2\lambda} I_d\right)$$



Playlist prediction via Metric Embedding

Extensions:

- Observable Features

$$\Pr(p^{[i]}|p^{[i-1]}) = \frac{e^{-\Delta(p^{[i]},p^{[i-1]})^2 + O(p^{[i]})^T W O(p^{[i-1]})}}{\sum_j e^{-\Delta(s_j,p^{[i-1]})^2 + O(s_j)^T W O(p^{[i-1]})}}$$

$f_i W f_{i-1}^T$

Experiments:

Yes.com playlists

- Dec 2010 – May 2011

“Small” dataset:

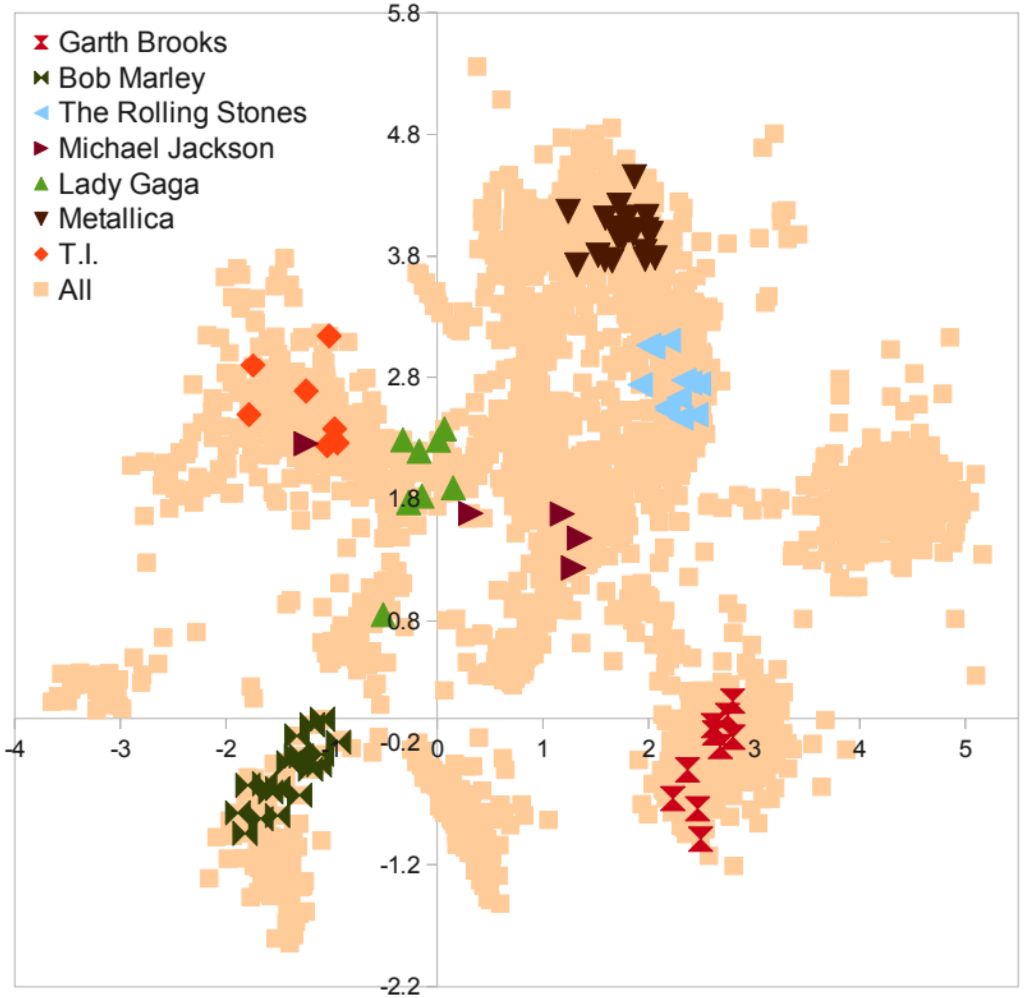
- 3,168 songs
- 134,431 playlists + 1,191,279 transitions

“Large” dataset

- 9,775 songs
- 172,510 playlists + 1,602,079 transitions

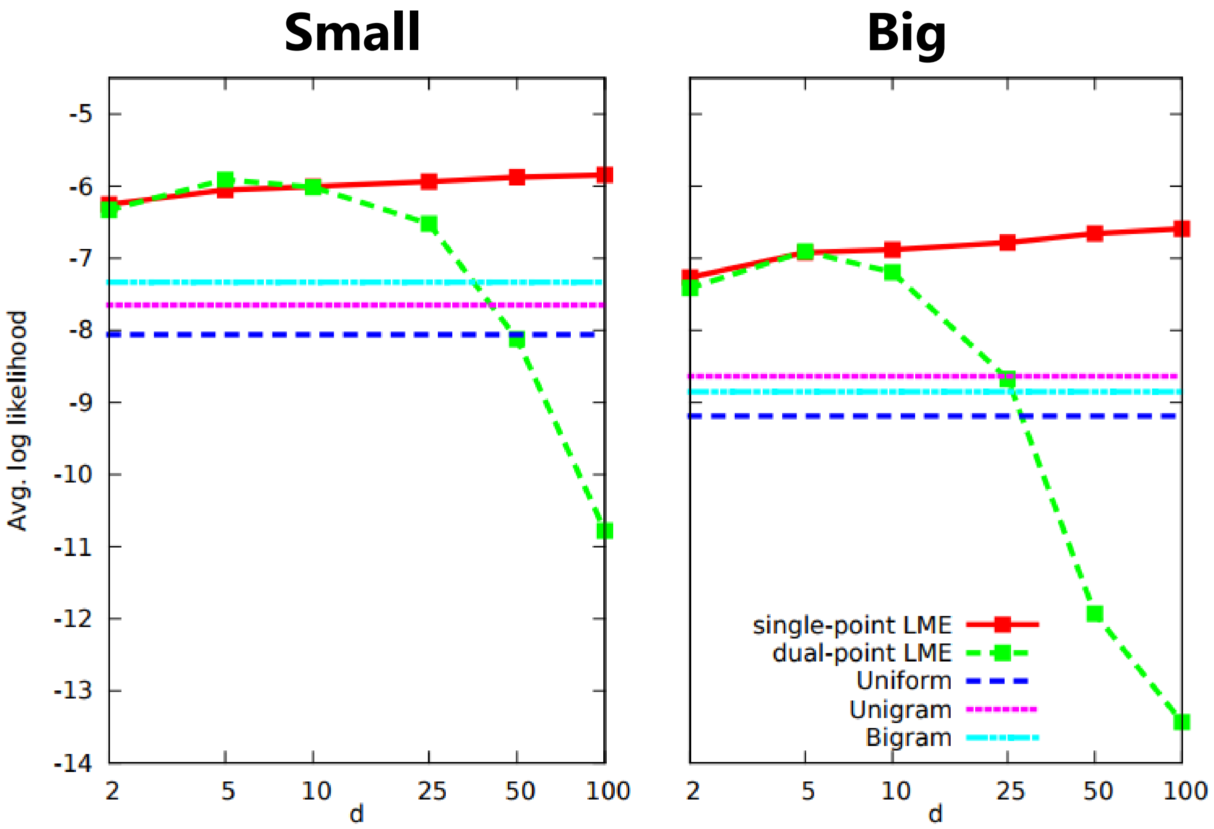
Playlist prediction via Metric Embedding

Experiments:



Playlist prediction via Metric Embedding

Experiments:



Playlist prediction via Metric Embedding

Morals of the story:

- Metric assumption works well in settings other than "geographical" data!
- However, they require some modifications in order to work well (e.g. "start points" and "end points")
- Effective combination of latent + observed features, as well as metric + inner-product models

Web Mining and Recommender Systems

Real-world applications of recommender systems: Efficient Natural Language Responses (Smart Reply)

Efficient Natural Language Response Suggestion for Smart Reply

Efficient Natural Language Response Suggestion for Smart Reply

MATTHEW HENDERSON, RAMI AL-RFOU, BRIAN STROPE, YUN-HSUAN SUNG, LÁSZLÓ LUKÁCS, RUIQI GUO, SANJIV KUMAR, BALINT MIKLOS, and RAY KURZWEIL, Google

This paper presents a computationally efficient machine-learned method for natural language response suggestion. Feed-forward neural networks using n-gram embedding features encode messages into vectors which are optimized to give message-response pairs a high dot-product value. An optimized search finds response suggestions. The method is evaluated in a large-scale commercial e-mail application, *Inbox by Gmail*. Compared to a sequence-to-sequence approach, the new system achieves the same quality at a small fraction of the computational requirements and latency.

Additional Key Words and Phrases: Natural Language Understanding; Deep Learning; Semantics; Email

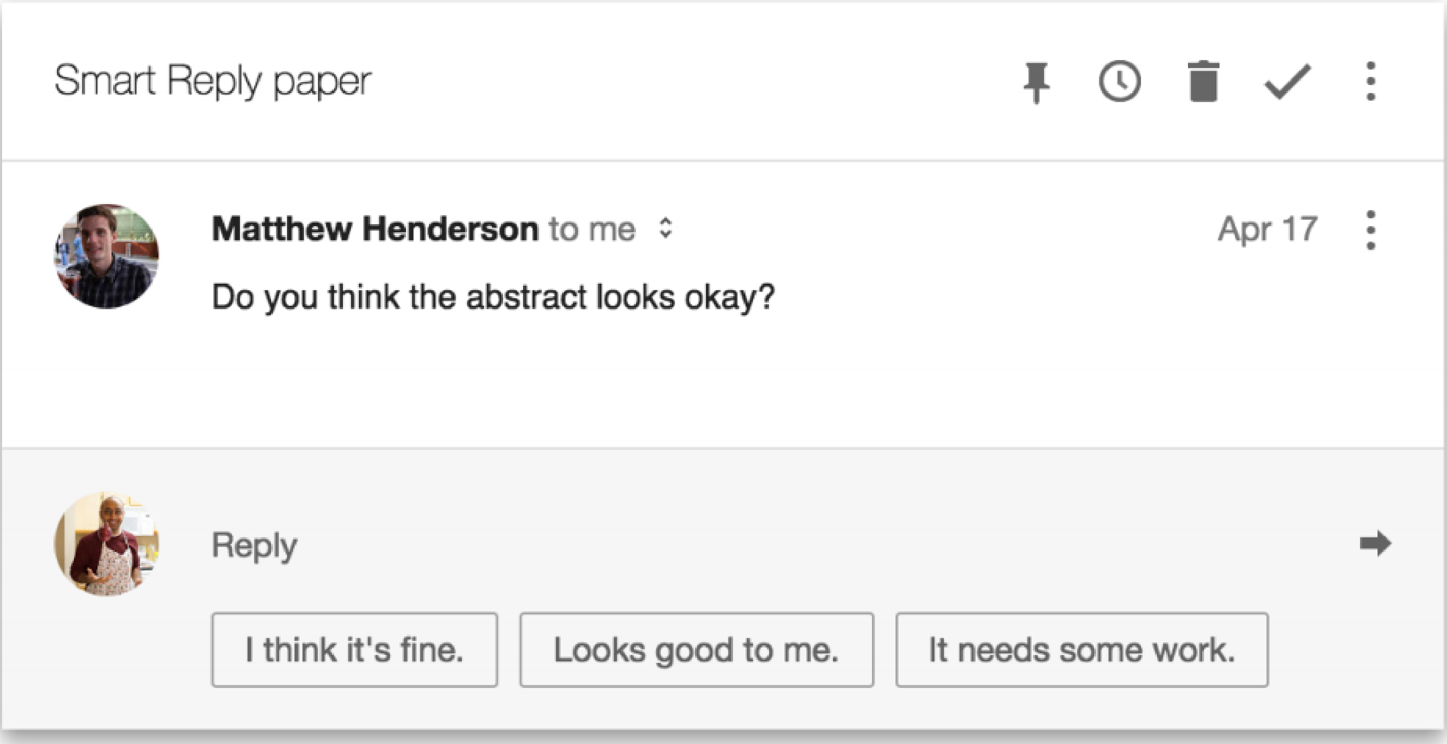
1 INTRODUCTION

Applications of natural language understanding (NLU) are becoming increasingly interesting with scalable machine learning, web-scale training datasets, and applications that enable fast and nuanced quality evaluations with large numbers of user interactions.

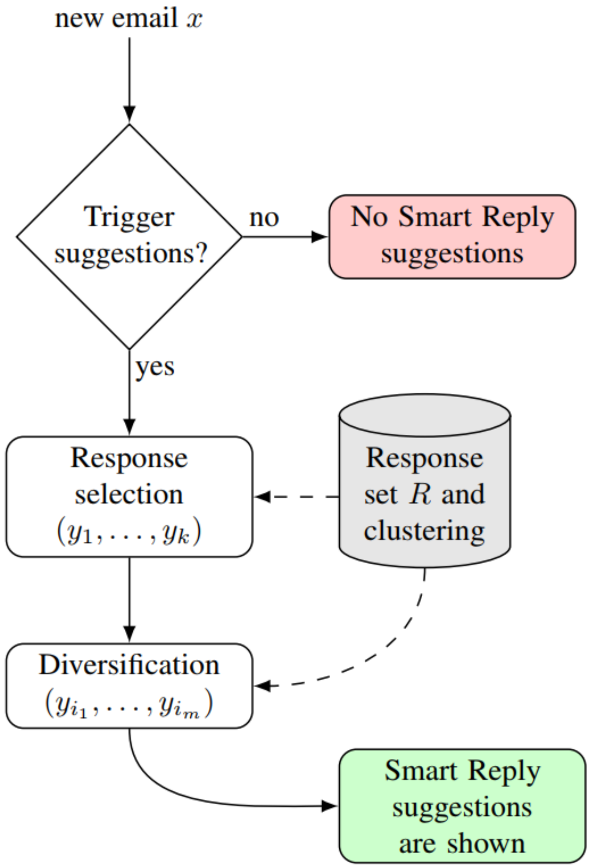
Early NLU systems parsed natural language with hand-crafted rules to explicit semantic representations, and used manually written state machines to generate specific responses from the output of parsing [18]. Such systems are generally limited to the situations imagined by the designer, and much of the development work involves writing more rules to improve the robustness of semantic

Efficient Natural Language Response Suggestion for Smart Reply

Goal: Automatically suggest common responses to e-mails



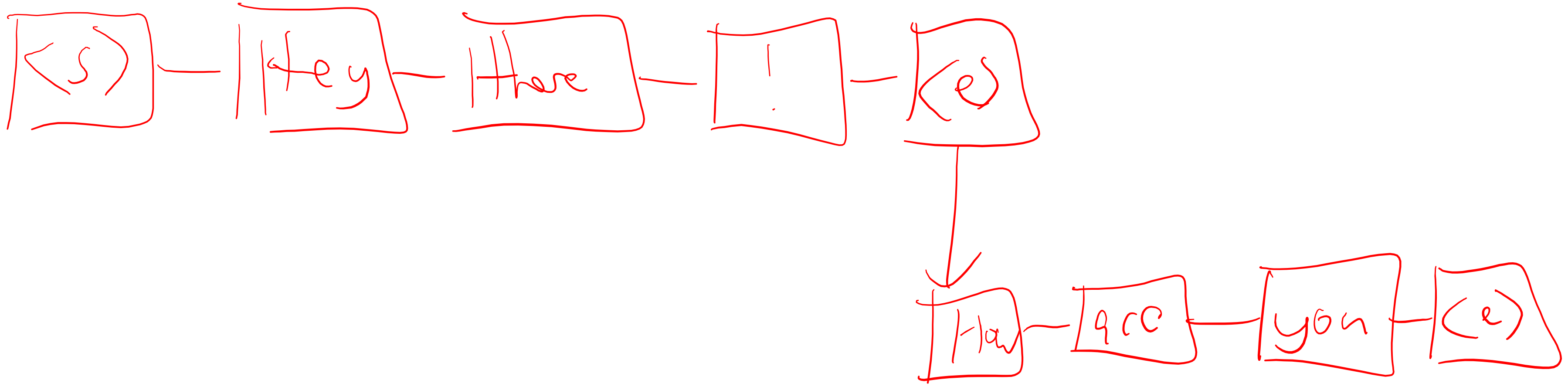
Basic setup


new email x
Trigger suggestions?
no
No Smart Reply suggestions
yes
Response selection $(y_1, \ldots, y_k)$
Response set R and clustering
Diversification $(y_{i_1}, \ldots, y_{i_m})$
Smart Reply suggestions are shown

Previous solution (KDD 2016)

- Based on a seq2seq method

$$
\begin{aligned}
P(y \mid x) &= P(y_1, \ldots, y_n \mid x_1, \ldots, x_m) \\
&= \textstyle\prod_{i=1}^{n} P_{\mathrm{LSTM}}(y_i \mid x_1, \ldots, x_m, y_1, \ldots, y_{i-1})
\end{aligned}
$$



Efficient Natural Language Response Suggestion for Smart Reply

Idea: Replace this (complex) solution with a simple multiclass classification-based solution

$$P(y \mid x) = \frac{P(x, y)}{\sum_k P(x, y_k)} \qquad P(x, y) \propto e^{S(x,y)}$$

$$P(\text{response } y \mid \text{email } x) = \frac{e^{S(x,y)}}{\sum_{y'} e^{S(x,y')}}$$

Efficient Natural Language Response Suggestion for Smart Reply

Idea: Replace this (complex) solution with a simple multiclass classification-based solution

$$P_{\text{approx}}(y \mid x) = \frac{e^{S(x,y)}}{\sum_{k=1}^{K} e^{S(x,y_k)}}$$

1 true reply

99 non replies

Efficient Natural Language Response Suggestion for Smart Reply

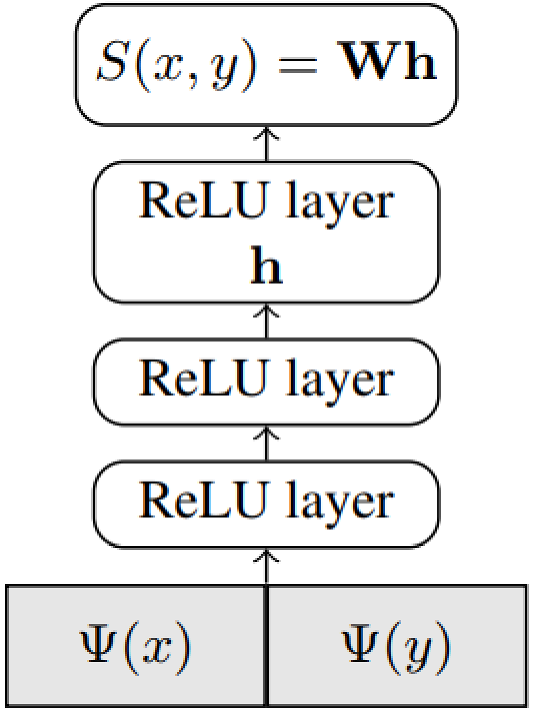
Model: S(x,y)

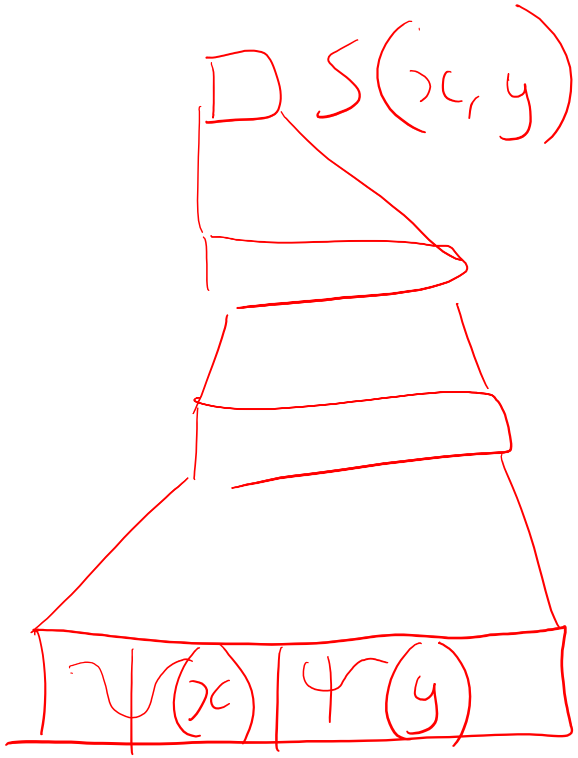
$$\Psi(x) \in \mathbb{R}^d$$

508	12	15	.	.	.	.	.	.			

$$\Psi(y) \in \mathbb{R}^d$$

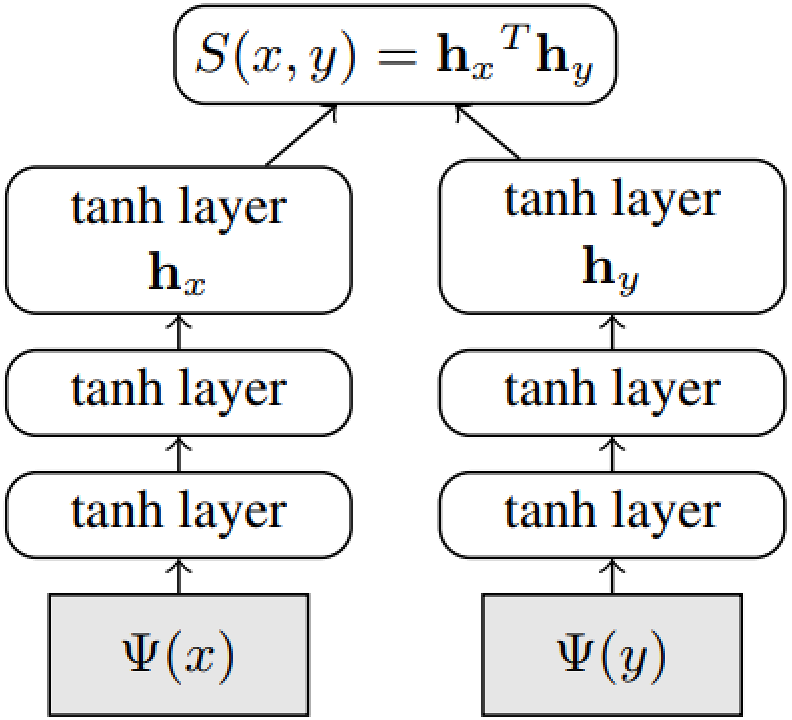
Model: Architecture v1





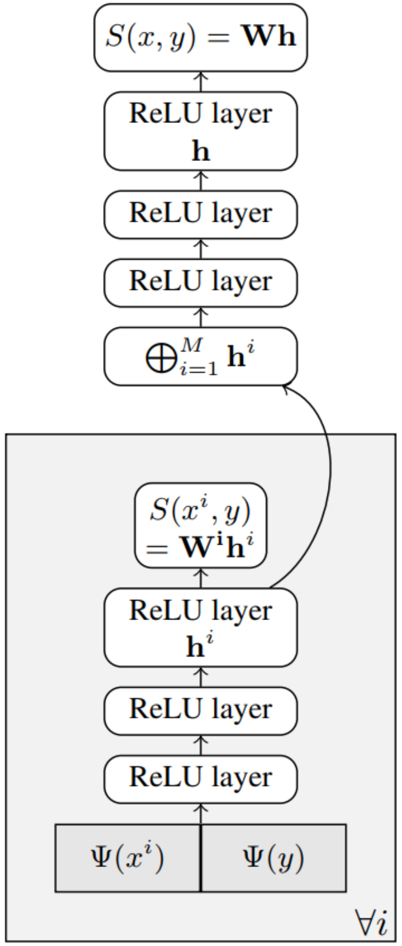
Efficient Natural Language Response Suggestion for Smart Reply

Model: Architecture v2



Efficient Natural Language Response Suggestion for Smart Reply

Model: Extensions



Model: Extensions

Message: Did you manage to print the document?	
With response bias	**Without response bias**
– Yes, I did.	– It's printed.
– Yes, it's done.	– I have printed it.
– No, I didn't.	– Yes, all done.

more popular

more compatible

diversity

Efficient Natural Language Response Suggestion for Smart Reply

Experiments: (offline)

Batch Size	Scoring Model	P@1
25	Joint	49%
25	Dot-product	48%
50	Dot-product	52%

prec@1

Efficient Natural Language Response Suggestion for Smart Reply

Experiments: (online)

System		**Experiment**	**Conversion rate relative to Seq2Seq**	**Latency relative to Seq2Seq**
Exhaustive search	(1)	Use a joint scoring model to score all responses in R.	–	500%
Two pass	(2)	Two passes: dot-product then joint scoring.	67%	10%
	(3)	Include response bias.	88%	10%
	(4)	Improve sampling of dataset, and use multi-loss structure.	104%	10%
Single pass	(5)	Remove second pass.	104%	2%
	(6)	Use hierarchical quantization for search.	104%	1%

Efficient Natural Language Response Suggestion for Smart Reply

Morals:

- Even a seemingly complex problem like natural-language response generation can be cast as a multiclass classification problem!
- Even a simple bag-of-words model proved to be sufficient, no need to handle “grammar” etc.
- Also, no personalization (though to what extent would this be possible with the data available?)

Overview

Morals:

- State-of-the-art recommender systems (whether from academia or industry) are not so far from what we learned in class
- All of them depended on some kind of maximum-likelihood expression, along with gradient ascent/descent!