Web Mining and Recommender Systems

Advanced Recommender Systems
This week

Methodological papers
• Bayesian Personalized Ranking
• Factorizing Personalized Markov Chains
• Personalized Ranking Metric Embedding
Goals:
This week

Application papers
• Recommending Product Sizes to Customers
• Playlist Prediction via Metric Embedding
• Efficient Natural Language Response Suggestion for Smart Reply
This week

We (hopefully?) know enough by now to...
• Read academic papers on Recommender Systems
• Understand most of the models and evaluations used

See also – CSE291
BPR: Bayesian Personalized Ranking from Implicit Feedback

Steffen Rendle, Christoph Freudenthaler, Zeno Gantner and Lars Schmidt-Thieme
{rendle, freudenthaler, gantner, schmidt-thieme}@isml.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 prediction task, they have been used also for the ranking task. However, personalization 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

\[ i > u_j \]
Bayesian Personalized Ranking

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

![Matrix Diagram]

**But!** “0”s aren’t necessarily negative!
Bayesian Personalized Ranking

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

This suggests a possible solution based on ranking
Bayesian Personalized Ranking

**Defn:** AUC (for a user $u$)

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

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

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 $u$)

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

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

Summary: Goal is to count how many times we identified \(i\) as being more preferable than \(j\) for a user \(u\).
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:
Bayesian Personalized Ranking

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

\[
\text{BPR-Opt} := \ln p(\Theta | >_u) \\
= \ln p(>_u | \Theta) p(\Theta) \\
= \ln \prod_{(u,i,j) \in D_S} \sigma(\hat{x}_{uij}) p(\Theta) \\
= \sum_{(u,i,j) \in D_S} \ln \sigma(\hat{x}_{uij}) + \ln p(\Theta) \\
= \sum_{(u,i,j) \in D_S} \ln \sigma(\hat{x}_{uij}) - \lambda_\Theta \|\Theta\|^2
\]
Bayesian Personalized Ranking

**Idea:** Replace the counting function $\delta(\hat{x}_{uij} > 0)$ by a smooth function.
Experiments:
• RossMann (online drug store)
• Netflix (treated as a binary problem)
Bayesian Personalized Ranking

Experiments:

Online shopping: Rossmann

Video Rental: Netflix
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 $\hat{x}_{uij} = \gamma_u \cdot \gamma_i - \gamma_u \cdot \gamma_j$.
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.
Goal: build \textbf{temporal} models just by looking at the item the user purchased previously

\[ r(u, i \mid j) \]

(or \( p_u(i \mid j) \))
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.
Is this assumption realistic?
Data setup: Rossmann basket data
Factorizing Personalized Markov Chains for Next-Basket Recommendation

**Prediction task:**

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

\[
p(B_t | B_{t-1}) \propto \prod_{i \in B_t} p(i | B_{t-1})
\]
Could we try and compute such probabilities just by counting?

\[
\hat{a}_{l,i} = \hat{p}(i \in B_t | l \in B_{t-1}) = \frac{\hat{p}(i \in B_t \land l \in B_{t-1})}{\hat{p}(l \in B_{t-1})} = \\
= \frac{|\{(B_t, B_{t-1}) : i \in B_t \land l \in B_{t-1}\}|}{|\{(B_t, B_{t-1}) : l \in B_{t-1}\}|}
\]

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

But it’s not personalized
What if we try to personalize?

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

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

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)
What if we try to personalize?

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

\[ C \in \mathbb{R}^{k_U, k_L, k_I}, \quad V^U \in \mathbb{R}^{U \times k_U}, \]
\[ V^L \in \mathbb{R}^{I \times k_L}, \quad V^I \in \mathbb{R}^{I \times k_I} \]
Factorizing Personalized Markov Chains for Next-Basket Recommendation

What if we try to **personalize**?

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

$$\arg\max_{\Theta} \prod_{u \in U} \prod_{B_t \in B^u} p(>_{u,t} \mid \Theta) \prod_{u \in U} \prod_{B_t \in B^u} \prod_{i \in B_t} \prod_{j \notin B_t} p(i >_{u,t} j \mid \Theta)$$
Factorizing Personalized Markov Chains for Next-Basket Recommendation

F@5

FMC: not personalized

MF: personalized, but not sequentially-aware
Morals of the story:

• Can improve performance by modeling \textit{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 \textit{tensor decomposition} techniques
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 Yuan2
1Interdisciplinary Graduate School, Nanyang Technological University, Singapore, sfeng003@e.ntu.edu.sg
2School of Computer Engineering, Nanyang Technological University, Singapore, {liutao,gaocong,quyan}@ntu.edu.sg
3School of Computing, Teesside University, UK, y.Zeng@tees.ac.uk
4School 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-GN method to enhance the personalized ranking with the sequential information 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
Goal: Can we build better sequential recommendation models by using the idea of metric embeddings

$$\gamma_u \cdot \gamma_i \quad \text{vs.} \quad d(\gamma_u, \gamma_i)$$
Why would we expect this to work (or not)?
Otherwise, goal is the same as the previous paper:

\[ p_u(i|j) \]
## Data

<table>
<thead>
<tr>
<th>Dataset</th>
<th>#User</th>
<th>#POI</th>
<th>#Check-in</th>
<th>Time range</th>
</tr>
</thead>
<tbody>
<tr>
<td>FourSquare</td>
<td>1917</td>
<td>2675</td>
<td>155365</td>
<td>08/2010-07/2011</td>
</tr>
<tr>
<td>Gowalla</td>
<td>4996</td>
<td>6871</td>
<td>245157</td>
<td>11/2009-10/2010</td>
</tr>
</tbody>
</table>
Personalized Ranking Metric Embedding for Next New POI Recommendation

Qualitative analysis

![Graph showing the ratio of new POIs over the number of days for FourSquare and Gowalla](image-url)
Personalized Ranking Metric Embedding for Next New POI Recommendation

Qualitative analysis

- Graph 1: Probability distribution of Pr(X<\alpha) against Hour (0-240).
  - Red: FourSquare
  - Blue: Gowalla

- Graph 2: Probability distribution of Pr(X<\alpha) against Distance (0-50 km).
  - Red: FourSquare
  - Blue: Gowalla
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)

\[ l_i >_{l^c} l_j \iff \hat{P}(l_i|l^c) > \hat{P}(l_j|l^c) \]
Personalized Ranking Metric Embedding for Next New POI Recommendation

Personalized version

\[ D_{u,l,c,l} = \alpha D_{u,l}^P + (1 - \alpha) D_{l,c,l}^S \]
Personalized Ranking Metric Embedding for Next New POI Recommendation

Personalized version

\[ D_{u,l^c,l} = \begin{cases} D_{u,l}^P & \text{if } \Delta(l, l^c) > \tau \\ \alpha D_{u,l}^P + (1 - \alpha) D_{l^c,l}^S & \text{otherwise} \end{cases} \]
Personalized Ranking Metric Embedding for Next New POI Recommendation

Learning

\[ P(>_{u,l^c} | \Theta) = P \left( (D_{u,l^c,l_j} - D_{u,l^c,l_i}) > 0 | \Theta \right) \]

\[ = \sigma(D_{u,l^c,l_j} - D_{u,l^c,l_i}) \]
Personalized Ranking Metric Embedding for Next New POI Recommendation

Results

(a) Precision on FourSquare
(b) Recall on FourSquare
(c) Precision on Gowalla
(d) Recall on Gowalla
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?)
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 to Customers

Vivek Sembium  
Amazon Development Center India  
viveksem@amazon.com

Atul Saroop  
Amazon Development Center India  
asaroop@amazon.com

Rajeev Rastogi  
Amazon Development Center India  
rastogi@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 customers.
- **Low diversity**: Users’ preferences are highly skewed, and it’s challenging to recommend products for customers with rare or unobserved preferences.
- **Large scale**: The sheer volume and variety of products and customers makes it infeasible to have an exhaustive database for every product.

The objective is to provide recommendations to customers in real-time.
**Goal:** Build a recommender system that predicts whether an item will “fit”:

\[(u, i) \rightarrow \{\text{small, fit, large}\}\]
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
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:**

\[
L(y_{ij}, f_w(s_i, t_j)) = \begin{cases} 
L^{bin}(+1, f_w(s_i, t_j) - b_2) & \text{if } y_{ij} = \text{Small} \\
L^{bin}(-1, f_w(s_i, t_j) - b_2) & \text{if } y_{ij} = \text{Fit} \\
L^{bin}(+1, f_w(s_i, t_j) - b_1) & \text{if } y_{ij} = \text{Large} \\
L^{bin}(-1, f_w(s_i, t_j) - b_1) & \text{if } y_{ij} = \text{Large} 
\end{cases}
\]
Recommending product sizes to customers

**Loss function:**

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

\[
L_i = \sum_{(i,j,y_{ij}) \in D \land y_{ij} = \text{Small}} \max\{0, 1 - f_w(s_i, t_j) + b_2\}
+ \sum_{(i,j,y_{ij}) \in D \land y_{ij} = \text{Fit}} \left(\max\{0, 1 + f_w(s_i, t_j) - b_2\}
+ \max\{0, 1 - f_w(s_i, t_j) + b_1\}\right)
+ \sum_{(i,j,y_{ij}) \in D \land y_{ij} = \text{Large}} \max\{0, 1 + f_w(s_i, t_j) - b_1\}
\]
Recommending product sizes to customers

Model fitting:
Recommendating 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**
Recommending product sizes to customers

### Experiments:

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Baseline RF</th>
<th>Baseline Linear</th>
<th>Baseline RF</th>
<th>Baseline Linear</th>
<th>Algorithm 1 RF</th>
<th>Algorithm 1 Linear</th>
<th>Algorithm 1 RF</th>
<th>Algorithm 1 Linear</th>
<th>Algorithm 1 RF</th>
<th>Algorithm 1 Linear</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0.6%</td>
<td>0%</td>
<td>0%</td>
<td>0%</td>
<td>16.4%</td>
<td>17.2%</td>
<td>17.9%</td>
<td>18.1%</td>
<td></td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>2.1%</td>
<td>0.4%</td>
<td>2.1%</td>
<td>20.3%</td>
<td>21.3%</td>
<td>21.9%</td>
<td>20.5%</td>
<td>21.3%</td>
<td></td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>3.8%</td>
<td>1.9%</td>
<td>1.9%</td>
<td>15.7%</td>
<td>16.1%</td>
<td>18.4%</td>
<td>17.8%</td>
<td>21.3%</td>
<td></td>
<td></td>
</tr>
<tr>
<td>D</td>
<td>2.7%</td>
<td>1.2%</td>
<td>1.6%</td>
<td>20.0%</td>
<td>20.2%</td>
<td>18.4%</td>
<td>17.4%</td>
<td>17.4%</td>
<td></td>
<td></td>
</tr>
<tr>
<td>E</td>
<td>1.5%</td>
<td>0.2%</td>
<td>0.6%</td>
<td>15.8%</td>
<td>15.6%</td>
<td>17.4%</td>
<td>17.4%</td>
<td>17.4%</td>
<td></td>
<td></td>
</tr>
<tr>
<td>F</td>
<td>2.5%</td>
<td>2.7%</td>
<td>2.3%</td>
<td>18.1%</td>
<td>17.3%</td>
<td>18.5%</td>
<td>17.3%</td>
<td>17.3%</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
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!
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 in 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
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)
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 last lecture)
Basic model ("single point"): 
Playlist prediction via Metric Embedding

“Dual-point” model

\[ ||X(s) - X(s')||_2 \]

\[ ||V(s) - U(s')||_2 \]
Extensions:

- Popularity biases

\[
Pr(p_i | p_{i-1}) = \frac{e^{-\Delta(p_i, p_{i-1})^2 + b_i}}{\sum_j e^{-\Delta(s_j, p_{i-1})^2 + b_j}}
\]
Playlist prediction via Metric Embedding

Extensions:

• Personalization

\[
Pr(p[i]|p[i-1], u) = \frac{e^{-\Delta(p[i], p[i-1])^2 + A(p[i])^T B(u)}}{\sum_j e^{-\Delta(s_j, p[i-1])^2 + A(s_j)^T B(u)}}
\]
Playlist prediction via Metric Embedding

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

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

Experiments:

Yes.com playlists
- Dec 2010 – May 2011

“Small” dataset:
- 3,168 songs
- 134,431 + 1,191,279 transitions

“Large” dataset
- 9,775 songs
- 172,510 transitions + 1,602,079 transitions
Playlist prediction via Metric Embedding

Experiments:
Playlist prediction via Metric Embedding

Experiments:

**Small**

**Big**

![Graphs showing experiment results for small and big datasets](image)
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
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 parsing and the coverage of the state machine. The systems used by early commercial products [31]
Goal: Automatically suggest common responses to e-mails
Efficient Natural Language Response Suggestion for Smart Reply

Basic setup

1. **new email** $x$
   - **Trigger suggestions?**
     - **no** → **No Smart Reply suggestions**
     - **yes** → **Response selection** $(y_1, \ldots, y_k)$
       - **Response set** $R$ and clustering
       - **Diversification** $(y_{i1}, \ldots, y_{im})$
       - **Smart Reply suggestions are shown**
Previous solution (KDD 2016)

- Based on a seq2seq method

\[
P(y \mid x) = P(y_1, \ldots, y_n \mid x_1, \ldots, x_m) \\
= \prod_{i=1}^n P_{\text{LSTM}}(y_i \mid x_1, \ldots, x_m, y_1, \ldots, y_{i-1})
\]
Idea: Replace this (complex) solution with a simple multiclass classification-based solution

\[
P(y \mid x) = \frac{P(x, y)}{\sum_k P(x, y_k)} \quad P(x, y) \propto e^{S(x, y)}
\]
Idea: Replace this (complex) solution with a simple multiclass classification-based solution

\[ P_{\text{approx}}(y \mid x) = \frac{e^{S(x,y)}}{\sum_{k=1}^{K} e^{S(x,y_k)}} \]
Efficient Natural Language Response Suggestion for Smart Reply

Model: $S(x,y)$

$\Psi(x) \in \mathbb{R}^d$
Efficient Natural Language Response
Suggestion for Smart Reply

**Model:** Architecture v1

\[ S(x, y) = Wh \]

- ReLU layer
- \( h \)
- ReLU layer
- ReLU layer
- \( \Psi(x) \) and \( \Psi(y) \)
Model: Architecture v2
**Model:** Extensions

\[ S(x, y) = Wh \]

ReLU layer

\[ h \]

ReLU layer

ReLU layer

ReLU layer

\[ \bigoplus_{i=1}^{M} h^i \]

\[ S(x^i, y) = W'h^i \]

ReLU layer

\[ h^i \]

ReLU layer

ReLU layer

ReLU layer

\[ \Psi(x^i) \quad \Psi(y) \]

\( \forall i \)
**Model:** Extensions

<table>
<thead>
<tr>
<th>Message: Did you manage to print the document?</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>With response bias</strong></td>
</tr>
<tr>
<td>– Yes, I did.</td>
</tr>
<tr>
<td>– Yes, it’s done.</td>
</tr>
<tr>
<td>– No, I didn’t.</td>
</tr>
</tbody>
</table>
**Experiments:** (offline)

<table>
<thead>
<tr>
<th>Batch Size</th>
<th>Scoring Model</th>
<th>P@1</th>
</tr>
</thead>
<tbody>
<tr>
<td>25</td>
<td>Joint</td>
<td>49%</td>
</tr>
<tr>
<td>25</td>
<td>Dot-product</td>
<td>48%</td>
</tr>
<tr>
<td>50</td>
<td>Dot-product</td>
<td>52%</td>
</tr>
</tbody>
</table>
## Experiments: (online)

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