Electronic Preprint for Journal of Information Processing Vol.29

Regular Paper

Dynamic Hyperbolic Embeddings with Graph-Centralized Regularization for Recommender Systems

Kojiro Iizuka[1,†1,a)] Makoto P. Kato[2,b)] Yoshifumi Seki[1,c)]

Received: March 9, 2021, Accepted: July 3, 2021

Abstract: In this work, we propose two techniques for accurate and efficient hyperbolic embeddings for real-world recommender systems. The first technique is regularization. We found that the graphs of various recommendation datasets exhibit hierarchical or tree-like structures suitable for hyperbolic embeddings, while these structures are not well modeled by the original hyperbolic embeddings. Hence, we introduce a regularization term in the objective function of the hyperbolic embeddings for forcibly reflecting hierarchical or tree-like structures. The second technique is an efficient embedding method, which only updates the embedding of items that are recently added in a recommender system. In an offline evaluation with various recommendation datasets, we found that the regularization enforcing hierarchical or tree-like structures improved HR@10 up to +9% compared to hyperbolic embeddings without the regularization. Moreover, the evaluation result showed that our model update technique could achieve not only greater efficiency but also more robustness. Finally, we applied our proposed techniques to a million-scale news recommendation service and conducted an A/B test, which demonstrated that even *10-dimension* hyperbolic embeddings successfully increased the number of clicks by +3.7% and dwell time by +10%.

Keywords: recommender systems, hyperbolic embeddings, regularization, news service

1. Introduction

Although recent studies show that embedding-based recommender algorithms have achieved high levels of accuracy, how to learn embeddings in real-world recommender settings is one of the remaining challenges [12], [25]. In particular, there are special requirements for real-time stream recommender systems with a large number of new items added continuously in short intervals, such as news applications and social network services: 1) compact embeddings for avoiding performance degradation in response time [18], [25]; 2) online learning algorithms that can incrementally update the embeddings [15]; and 3) sufficiently high recommendation accuracy to allow users to find what they need within a large volume of information [17].

To satisfy the above requirements for a real-time stream recommender system, *hyperbolic embeddings* are considered as a reasonable solution, since they have recently been reported to be compact and able to produce highly accurate embeddings [23]. Hyperbolic embeddings are known to perform well when learning with hierarchical or tree-like structure data, even in low-dimensional spaces. However, there remain several questions in applying hyperbolic embeddings to a real recommender service: 1) whether the existing recommendation datasets have a suitable hierarchical or tree-like structure to embed an item in a low-dimensional space; 2) how online learning should be conducted without sacrificing the recommendation accuracy; and 3) whether hyperbolic embeddings actually lead to a better recommendation accuracy and user satisfaction when they are applied to a real recommendation service.

In this study, we first investigated whether the graph of recommendation datasets has a hierarchical or tree-like structure. While they were shown to form such a structure, we also found simple application of the hyperbolic embeddings did not allow us to learn embeddings that adequately reflected the hierarchical structure. Therefore, we introduced a regularization technique to promote embeddings that reflects a hierarchical structure using graph centrality in the loss function. Next, we proposed a model update technique for online hyperbolic learning. In this method, we only update embeddings of the most recently added items while keeping the previously added embeddings unchanged.

We conducted an offline experiment and A/B testing to evaluate the two proposed techniques. Throughout the offline experiment, we used both public datasets and a proprietary dataset generated by a million-scale news service. In the offline evaluation, we confirmed that regularization that leveraged the graph centrality improved the ranking metrics. Furthermore, we simulated real-time stream recommender systems and found that the dynamic model update improved the learning efficiency and robustness. Finally, we evaluated the performance of our method in a real-world news application by conducting A/B testing. The results showed that the number of clicks rose significantly (+3%), and dwell time also rose (+10%).

Our Contributions. The key contributions of our work are summarized as follows:

[1] Gunosy Inc., Minato, Tokyo 107–6016, Japan
[2] University of Tsukuba, Tsukuba, Ibaraki 305–8577, Japan
[†1] Presently with University of Tsukuba
[a)] kojiro.iizuka@gunosy.com
[b)] mpkato@acm.org
[c)] yoshifumi.seki@gunosy.com

- **Analysis**: We investigated the existence of a hierarchical and tree-like structure in the graphs of various recommendation datasets. All graphs of the recommendation datasets that we investigated showed high hierarchical or tree-like structure metrics.
- **Accuracy**: We proposed regularization using graph centrality for hyperbolic embeddings. By introducing this regularization, we improved the performance of recommender systems.
- **Efficiency**: We proposed a dynamic model update strategy for real-time stream recommender systems. The empirical simulation results showed that this method achieved not only greater efficiency but also greater robustness.
- **Application**: To the best of our knowledge, this is the first work to deploy a hyperbolic embedding method in a real-time stream recommender system. We deployed our method to a million-scale news recommendation service and confirmed that the number of clicks and user dwell time increased significantly.

The remainder of this paper is organized as follows. We describe related works in Section 2, and in Section 3, we report the results of our analysis of hierarchical or tree-like structure in the recommendation datasets. We propose a regularization technique in Section 4 and model update strategy in Section 5. In Sections 6 and 7, we present the results of the offline experiment and A/B testing, respectively. In Section 8, we conclude our work and suggest directions of future work.

2. Related Work

First, we introduce hyperbolic embeddings. Then, we describe regularization techniques. Finally, we review model update strategies for real-world recommender systems.

Hyperbolic space has recently been used to model complex networks. Nickel and Kiela applied hyperbolic distance to model taxonomic entities and graph nodes [23]. Following this work, various applications and generalization using hyperbolic embeddings have been proposed in the literature. In the information retrieval field, He et al. proposed a question-answering system using a hyperbolic space [29]. Hyperbolic models have been also used for recommendations [6], [11], [30]. Shimizu et al. generalize the fundamental components of neural networks in a single hyperbolic geometry model [28]. Nonetheless, there were no investigations into why hyperbolic models work well for recommendation datasets. Thus, in this work, we investigate whether recommendation datasets have a hierarchical or tree-like structure suitable for hyperbolic embeddings.

To improve recommender effectiveness, regularization techniques are frequently used. Matrix factorization (MF) models have many parameters, and to optimize them, a regularization term is added to prevent overfitting. The user and item factors learned through matrix factorization are usually regularized with the L2-norm. Liang et al. used the item–item co-occurrence matrix for regularization [21]. Rendle and Schmidt-Thieme used regularization to learn kernel matrix factorization models [27]. However, very few classical regularization methods are directly applicable or easily generalizable in the Poincaré ball model of hyperbolic space. For instance, the L2 regularization constraint degrades recommendation accuracy because it requires that all nodes remain close to the origin and breaks a hierarchical or tree-like structure [20]. In this paper, we use a graph property—specifically, graph centrality–to maintain hierarchical or tree-like structures for regularization to improve recommender accuracy.

Efficiency is one challenge for real-world recommender systems. An essential element of a practical recommender system is handling the dynamic nature of incoming data, for which timeliness is crucial consideration. Because it is prohibitive to retrain the full model online, various studies have developed incremental learning strategies for neighbor-based, graph-based, and probabilistic MF methods [15]. In this study, we propose dynamic model updates for hyperbolic embeddings rather than batch updates inspired by previous incremental learning strategies and aim to improve efficiency and robustness. We note that previous hyperbolic models [6], [11], [30] are difficult to apply to the real-time stream recommender system because of their computational resource limitations. Thus, we focused on building a hyperbolic model for efficient recommendations with dynamic updates.

3. Data Analysis

In this section, we investigate whether recommendation datasets have a hierarchical or tree-like structure.

3.1 Notation

To represent the user and item interactions in recommender datasets, we introduce a directed graph $G = (V, E)$, where V is a set of nodes that represents all of the items, and E is a set of edges that individually connect one item to another. $|V|$ and $|E|$ represent numbers of nodes and edges, respectively. In this work, we constructed edges based on users' interactions and connected items if they were adjacent in a user's history of interactions. More formally, letting $\mathbf{i} = (i_1, i_2, \ldots)$ be a sequence of items a user has interacted with, sorted by the interaction time, $E = \bigcup_i \{(i_k, i_{k+1}) | k = 1, 2, \ldots\}$. We use this graph definition throughout this paper.

3.2 Datasets

In this study, we took datasets from diverse domains, including movie, news, and e-commerce services, with details as follows:

- **MovieLens20M**: This is a widely used dataset from GroupLens Research for evaluating recommender algorithms. This dataset contains ratings of movies [*1]. For each user, item IDs with a rating more than 1.2 times the average value were extracted as positive feedback.
- **Online Retail** [8]: This is an e-commerce dataset that contains all the transactions occurring between Januaray 12, 2010 and September 12, 2011 for a UK-based and registered non-store online retail site [*2]. The company primarily sells unique all-occasion gifts. Many customers of the company are wholesalers.
- **Amazon Reviews** [22]: This dataset contains product reviews from Amazon. We selected five categories, *Sports*,

[*1] https://grouplens.org/datasets/movielens/
[*2] http://archive.ics.uci.edu/ml/datasets/Online+Retail.

Table 1 Dataset statistics.

	$\|V\|$	$\|E\|$	Degree	Density	C(k)
MovieLens	8,026	83,400	20.7	0.00258	0.0717
Online Retail	2,660	23,810	17.9	0.00673	0.0781
Sports	14,576	100,095	13.7	0.000942	0.0412
Cell Phones	26,386	169,920	12.9	0.000488	0.0364
Games	25,026	157,167	12.6	0.000501	0.0303
Automotive	24,771	167,236	13.5	0.000545	0.0334
Clothing	8,597	37,748	8.78	0.00102	0.0495
Lucra	986	23,282	47.2	0.0479	0.491

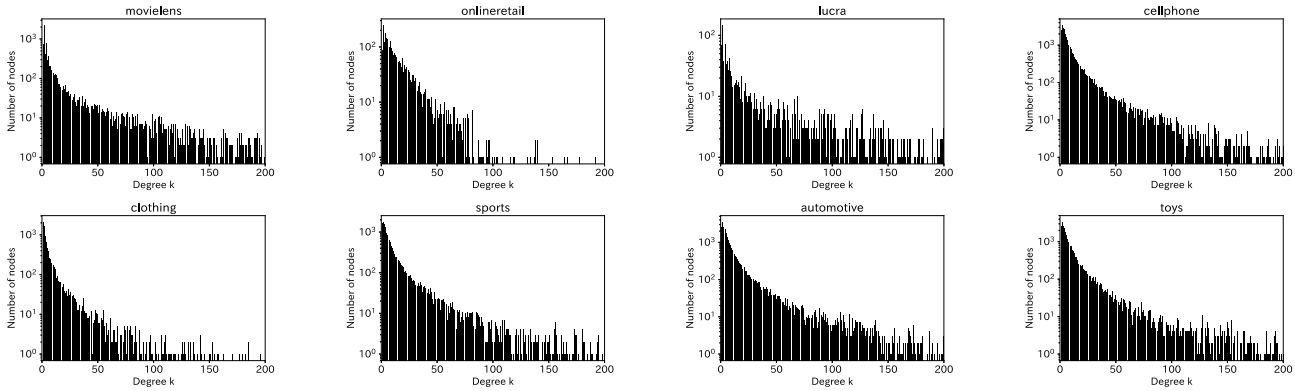

Fig. 1 Degree distributions.

Cell Phones, *Games*, *Automotive*, and *Clothing*, by considering the diversity of domains and dataset size [*3].

- **Lucra**: This is a news application service dataset. It is one of the most popular news application services among female users in a country served by Gunosy. The news application dataset is different from the other datasets in that the recommendation items are frequently changed.

We filtered out users who had interactions with items less than three times and only used each user's last ten interactions in the analysis. **Table 1** shows the statistics of the datasets.

3.3 Metrics

Here, we introduce the graph metrics related to hierarchical or tree-like structures.

In hierarchical networks, the degree of clustering can be used as an indicator of a hierarchical structure. *Clustering coefficient* [31] is a measure of the degree to which nodes in a graph tend to cluster together and is defined as follows: $C(k_i) = \frac{1}{|V|}\sum_{i=1}^{|V|}\frac{|\{e_{jk}:v_j,v_k\in N_i,e_{jk}\in E\}|}{k_i(k_i-1)}$, where N_i is a set of nodes that are connected to v_i, and $k_i = |N_i|$.

Gromov's hyperbolicity δ was used for the analysis of tree-like structures [1]. The smaller δ is, the closer the graph is to a tree, and it becomes a tree when $\delta = 0$ [7]. More formally, Gromov's hyperbolicity δ is defined as follows [1]. Let u, v, w, and x be nodes in the graph G. Consider the three distance sums, $dG(u,v) + dG(w,x)$, $dG(u,w) + dG(v,x)$, and $dG(u,x) + dG(v,w)$. We denote S_1, S_2, and S_3 that are sorted in non-decreasing order of the three distance sums: $S_1 \le S_2 \le S_3$ and denote the hyperbolicity of a quadruplet u, v, w and x as $\delta(u,v,w,x) = (S_3 - S_2)/2$. Then, δ-hyperbolicity in the graph G is the maximum hyperbolicity over all possible quadruplets $u,v,w,x \in V$, or $\delta(G) = \max\delta(u,v,w,x)$. As the computational time of the δ-hyperbolicity is $O(|V|^4)$, it is not easy to obtain the exact value of the δ-hyperbolicity. Thus, we approximately computed the δ-hyperbolicity by sampling 100,000 quadruplet nodes. Note that we ignored the δ value if a selected quadruplet node was not connected following to a previous work [7].

3.4 Results

Figure 1 shows the degree distributions, which reveal that all of the datasets had a power law distribution. A network with a degree distribution that follows a power law is called a scale-free network [2].

Figure 2 shows the clustering coefficient plots. When a graph has a scale-free property, it has been empirically shown that $C(k) \simeq k^{-\beta}$ if the graph is hierarchical [9], [19], [26]. In Fig. 2, most of our datasets had linear fittings in which the coefficient of determination was negative (excluding MovieLens and Lucra), and thus, the graphs had hierarchical structures.

Table 2 shows the relative frequency of δ-hyperbolicity values from sampled nodes, which were low for all of the datasets. More than 98% of the quadruplets had $\delta \le 1$, indicating that all of the graphs were metrically close to trees. In particular, MovieLens and Lucra had δ-hyperbolicity of 1, meaning that they had a chordal graph structure. The other two datasets had δ-hyperbolicity of 1.5. Because k-chordal graphs have hyperbolicity at most $k/4$, the other two datasets were 6-chordal graphs, which indicates the graph did not contain any chordless 7-cycles [32]. Thus, all the datasets were metrically close to trees.

In summary, the examined recommendation datasets were found to have hierarchical or tree-like structures in the graph suitable for hyperbolic embeddings.

[*3] Datasets were obtained from http://jmcauley.ucsd.edu/data/amazon/ using the five-core setting, with the domain names truncated in the interest of space.

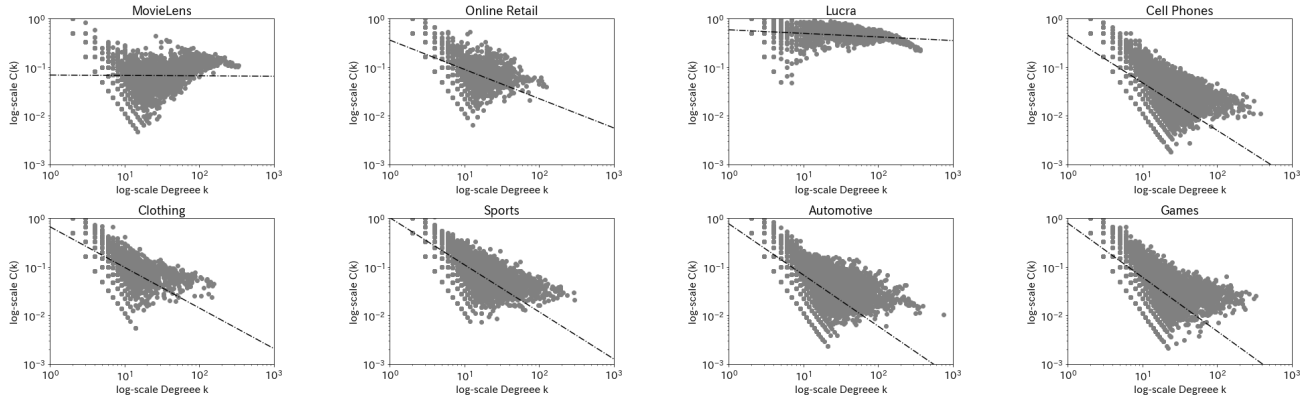

Fig. 2 Clustering coefficient plots.

Table 2 Relative frequency of δ-Hyperbolicity quadruplets in our graph datasets.

δ-Hyperbolicity	MovieLens	Online Retail	Sports	Cell Phones	Games	Automotive	Clothing	Lucra
0	6.38e-01	5.92e-01	6.06e-01	6.07e-01	6.09e-01	6.22e-01	6.19e-01	6.79e-01
0.5	3.55e-01	3.92e-01	3.81e-01	3.80e-01	3.79e-01	3.68e-01	3.70e-01	3.18e-01
1	6.58e-03	1.46e-02	1.18e-02	1.11e-02	1.14e-02	8.74e-03	1.04e-02	1.82e-03
1.5	-	1.00e-05	2.18e-05	2.01e-05	1.02e-05	1.02e-05	3.29e-05	-
max δ	1	1.5	1.5	1.5	1.5	1.5	1.5	1

4. Graph-Centralized Regularization

In this section, we propose a regularization technique for hyperbolic embedding using graph-central structures.

4.1 Hyperbolic Embeddings

The Poincaré ball model is the Riemannian manifold $\mathcal{P}^d = (\mathcal{B}^d, g_x)$, in which $\mathcal{B}^d = \{\mathbf{x} \in \mathbb{R}^d : \|x\| < 1\}$ is the open d-dimensional unit ball that is equipped with the following metric: $g_{\mathbf{x}} = \left(\frac{2}{1-\|\mathbf{x}\|}\right)^2 g^E$, where $\mathbf{x} \in \mathcal{B}^d$ and g^E denotes the Euclidean metric tensor. The distance between points $\mathbf{v}, \mathbf{u} \in \mathcal{B}^d$ is given by $d(\mathbf{u}, \mathbf{v}) = \operatorname{arcosh}\left(1 + 2\frac{\|\mathbf{u}-\mathbf{v}\|^2}{(1-\|\mathbf{u}\|^2)(1-\|\mathbf{v}\|^2)}\right)$.

Poincaré embedding determines the embedding $\Theta = \{\theta_i\}_{i=1}^n \in \mathbb{R}^{n\times d}$, where n is the number of all the nodes and d is the embedding size by solving the following optimization problem:

$$\arg\min_{\Theta} \mathcal{L}(\Theta) \text{ s.t. } \forall\theta_i \in \Theta : \|\theta_i\| < 1.$$

Here, θ_i corresponds to each node in V. We use the same loss function as Ref. [23], which uses the negative samples $\mathcal{N}(u) \subset \{v : (u,v) \notin E, v \neq u\}$ to maximize the distance between the embeddings of unrelated objects:

$$\mathcal{L}(\Theta) = -\sum_{(u,v)\in E} \log \frac{e^{-d(u,v)}}{\sum_{v'\in\mathcal{N}(u)} e^{-d(u,v')}}.$$

Hyperbolic embeddings work for the graphs of hierarchical or tree structure because of the following two properties. 1) Hyperbolic space expands as one moves from the center of the sphere to the boundary. 2) The number of nodes at the top of the hierarchical structure or near the tree's root is fewer than the nodes at the bottom of the hierarchical structure or the tree leaves. Thus, nodes higher in the hierarchy or closer to the tree's root are expected to be embedded near the center of the sphere, while nodes lower in the hierarchy or closer to the leaves of the tree are expected to be embedded near the boundary.

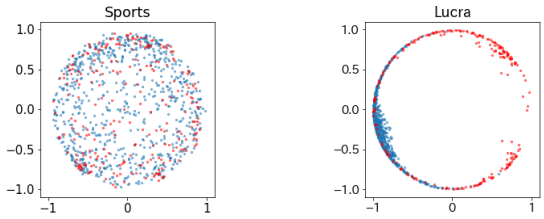

Fig. 3 2D item embedding visualization with and without regularization. The top 25% high-degree nodes are plotted as red dots and the remainder as blue dots.

4.2 Regularization

As we confirmed that the graphs of recommendation datasets had hierarchical or tree-like structures in Section 3, we investigated how each node was embedded by the hyperbolic embeddings method. It can be seen that nodes with high graph-centrality are embedded near the boundary as well as those with a low degree in **Fig. 3**.

Inspired from the related work [24], we hypothesize that a node that is similar to many other nodes should be embedded near the center of the ball. In this recommendation dataset, a node that is similar to many other nodes is the same as the node degree is large. The graph in the recommendation dataset is constructed by connecting the nodes interacted by a user, i.e., nodes are connected according to the similarity of the user's interests. However, without the regularization, the large degree nodes are embedded near the boundary in Fig. 3.

Thus, we introduce the L2-based regularization term $\mathcal{L}_R$ to embed the large degree nodes near the center of the ball:

$$\mathcal{L}_R(\Theta) = \sum_{u\in V} w_u\|\theta_u\|^2,$$

where w_u represents the graph-centrality metric of node u. We used the L2-based regularization because the embedding accuracy might be decayed if we use the geometrically defined regularization, such as the distance in the hyperbolic space. Due to the structure of the Poincaré ball model, the distance of points in-

creases exponentially the closer they are to the boundary. The ideal embedding in hyperbolic space is a situation where the nodes at the top of the hierarchical structure are placed at the center of the space, and the nodes at the bottom of the hierarchical structure are placed near the boundary away from the origin [23]. When we use the geometrically defined regularization, all nodes might be closer to the origin rather than the boundary because the penalty of nodes at the boundary is too high. Then, the hierarchical structure can not be fully reflected that the recommendation dataset has. We note that the standard L2 regularization is not also enough to improve embedding accuracy because all nodes are enforced to be close to the origin and can impair the benefit of hierarchical structures. On the contrary, the weighted L2 regularization term increases the penalty for distance from the center of the sphere for nodes with higher graph centrality. In other words, nodes with a high graph-centrality move closer to the origin, and nodes with a low graph-centrality move closer to the boundary.

The overall loss function is $\hat{\mathcal{L}}(\Theta) = \mathcal{L}(\Theta) + h\mathcal{L}_R(\Theta)$, where h is a hyper-parameter for regularization.

Although there are multiple graph-centrality metrics, such as closeness, degree, and eigenvector centrality, we used the degree centrality. Because real-world recommender systems have large graphs, the computational cost for calculating graph centrality needs to be low. Another reason is the following. The ideal embedding in hyperbolic space is that the highly general nodes at the top of the hierarchy are placed near the origin, while the less general nodes at the bottom of the hierarchy are placed near the boundary [24]. For example, in the case of a graph embedding for the sending relationship of employees' e-mails, the engineering manager is embedded near the origin, and the staff engineers are placed near the boundary. In this paper, we approximate the generality in the recommendation dataset by the node degree. For example, it is natural to think that a movie with a large degree that many users watched (e.x., Titanic) has a high generality, while a niche movie with a small degree has a low generality.

Additionally, by using the degree for weight $w = w_{in} + w_{out}$ (with w_{in} denoting the in degree and w_{out} the out degree), a single embedding update for each edge $(u, v) \in E$ can be calculated without w_u by using the following relationship:

$$\begin{aligned}
\mathcal{L}_R(\Theta) &= \sum_{u \in V} w_u \|\theta_u\|^2 \\
&= \sum_{u \in V} (w_{u,in} + w_{u,out}) \|\theta_u\|^2 \\
&= \sum_{u \in V} (|\{i : (i,u) \in E\}| + |\{i : (u,i) \in E\}|) \|\theta_u\|^2 \\
&= \sum_{(u,v) \in E} (\|\theta_u\|^2 + \|\theta_v\|^2)
\end{aligned}$$

The transformation from the third to the fourth line of the formula is calculated by rounding out the common edges that appear in the first and second terms.

5. Dynamic Model Updates

In this section, we introduce a dynamic model update technique to apply the hyperbolic embedding method to real-time stream recommender systems.

5.1 Problem Definition

In this problem setting, new items and their relationships are provided at every time step. Letting $G_t = (V_t, E_t)$ represent the items and relations that appear at time t, our problem is to learn the node embedding $\Theta_t \in \mathbb{R}^{n \times d}$ based on the data available up to time t. Our goal is to find an efficient and robust solution to this problem.

A straightforward approach for this problem is to learn the embedding $\Omega_t \in \mathbb{R}^{n \times d}$ by using the complete set of data up to time t (i.e., $G_1, G_2, \cdots, G_t$). We call this approach the *oracle*. This approach can be effective but could be infeasible for real-time stream recommender systems that regularly update the embeddings.

In order to clarify the goal of this work, we introduce an effectiveness measure for embedding, $f : \mathbb{R}^{n \times d} \to \mathbb{R}$. This function takes the embedding and measures its effectiveness at a specific task. For example, it can be nDCG@10 in a recommendation task. The robustness of the solution can be considered the closeness to the effectiveness of the *oracle* embedding even when the entire set of data is not used for the embedding learning. Hence, the robustness is measured by $1/(1 + f(\Omega_t) - f(\Theta_t))$. In the following subsections, we propose techniques to close the gap between Ω_t and Θ_t and increase robustness.

5.2 Dynamic Model Update

We propose a dynamic model update strategy for real-time stream recommender systems. Since new items are added regularly in real-time stream recommender systems, it is necessary to obtain an embedding of new items as soon as they arrive. However, it is infeasible to use all of the data up to time t (i.e., $G_1, G_2, \cdots, G_t$), at every time step (i.e., the *oracle* embedding), but it can be less effective than *oracle* to use only the data obtained at time t (i.e., G_t) for training the embedding of V_t. This *batch learning* approach using only G_t can result in a considerable inconsistency of embeddings between different time steps; for example, similar items that appear at different time steps may be embedded very differently. These problems imply low effectiveness of batch learning relative to *oracle* embedding, and this accounts for its low robustness.

In contrast, our dynamic model update strategy utilizes only the latest data from multiple time steps and keeps updating recently learned embeddings until they become stable. The key idea of our solution is to update only the embeddings of items that appear within window size τ by using items and their relationships within the same window, as well as those necessary to learn the embeddings for the new items. More precisely, at time t, we only update the embeddings of $V_{t-\tau}, V_{t-\tau+1}, \cdots, V_t$, with the data obtained between time $t - \tau$ and t. In addition, we incorporate existing embeddings of nodes connected by the relationships within the time window. These nodes and edges are formally defined as $G_{t-\tau:t} = (V_{t-\tau:t}, E_{t-\tau:t})$, where $V_{t-\tau:t} = \cup_{t'=t-\tau}^{t} V_{t'}$ and $E_{t-\tau:t}$ is a set of edges that appear within the time window $[t - \tau, t]$ and consist of at least a node from $V_{t-\tau:t}$. In other words, $E_{t-\tau:t} = \{(u, v) \in \cup_{t'=t-\tau}^{t} E_{t'} : u \in V_{t-\tau:t} \vee v \in V_{t-\tau:t}\}$. By these means, we ensure the consistency of embeddings between new items and existing items.

Algorithm 1: Dynamic Model Updates

Input: $G_t = (V_t, E_t)$ for $t = 1, 2, \ldots$.
1 Randomly initialize Θ_0
2 $V_0 \longleftarrow \emptyset$, $E_0 \longleftarrow \emptyset$
3 $V_{0:\tau} \longleftarrow \emptyset$, $E_{0:\tau} \longleftarrow \emptyset$
4 **for** $t = \tau + 1, \tau + 2, \ldots$ **do**
5 $\quad V_{t-\tau:t} \longleftarrow (V_{t-\tau-1:t-1} \setminus V_{t-\tau-1}) \cup V_t$
6 $\quad E'_t \longleftarrow \{(u, v) \in E_t : u \in V_{t-\tau:t} \vee v \in V_{t-\tau:t}\}$
7 $\quad E_{t-\tau:t} \longleftarrow (E_{t-\tau-1:t-1} \setminus E_{t-\tau-1}) \cup E'_t$
8 $\quad \Theta_t \longleftarrow \Theta_{t-1}$
9 $\quad \Theta_t \longleftarrow \mathrm{UpdateEmbedding}(G_{t-\tau:t}, \Theta_t)$
end

In practice, the embeddings are learned through Algorithm 1. Lines 1–3 initialize the variables used in this algorithm. Lines 5–7 update nodes $V_{t-\tau:t}$ and edges $E_{t-\tau:t}$ to learn the embeddings at time t. Line 8 copies the embedding from the previous time step to that at time t for initialization. Line 9 learns the embeddings with $G_{t-\tau:t} = (V_{t-\tau:t}, E_{t-\tau:t})$ based on the initialized embedding Θ_t. In the embedding learning process, nodes in $V_{t-\tau:t}$ are updated based on their initialized embedding and edges $E_{t-\tau:t}$, while nodes that are not in $V_{t-\tau:t}$ remain unchanged.

6. Offline Experiments

In this section, we describe how we conducted experiments to answer the following research questions:

RQ1 Is our regularization effective at improving evaluation accuracy of recommendation?

RQ2 How efficient and robust is our dynamic embedding?

We called the first task answering **RQ1** the *accuracy evaluation* and the second task answer **RQ2** the *efficiency evaluation*.

6.1 Metrics

We used the normalized discounted cumulative gain (nDCG@10) and hit ratio (HR@10) as the primary evaluation metrics. Both are well-established ranking metrics for recommendation. We evaluate the methods with user sessions. A session in this context means an event that a user visits a recommendation service and clicks on one of the items in the recommended list. Let S be the set of sessions for the evaluation, $c_{s,i}$ be 1 when the item at position i is clicked, and 0 otherwise. Each metric is formulated as:

$$\mathrm{nDCG@}k = \frac{1}{|S|} \sum_{s \in S} \frac{\sum_{i=1}^{k} c_{s,i} / \log_2(i+1)}{\max_{\pi} \sum_{i=1}^{k} c_{s,\pi(i)} / \log_2(\pi(i)+1)}$$

$$\mathrm{HR@}k = \frac{1}{|S|} \sum_{s \in S} \min(|\{i | c_{s,i} = 1, i \le k\}|, 1)$$

where π is an arbitrary permutation of positions.

We constructed a ranking for a user by selecting a positive sample item that the user clicked on and 20 negative sample items that the user did not click on.

6.2 Accuracy Evaluation

6.2.1 Settings

For all the datasets, the test sets were composed of the last item on which the user clicked, and these items were excluded from the train and validation sets for each user. We split the dataset into train : validation : test = 80 : 10 : 10. The models were trained with the training set for 20 epochs and then evaluated with the test set. We used the following common parameter settings: vector dimension (10, 100, and 300), learning rate (0.001, 0.01, 0.1, and 1.0), batch size (128, 256, and 512), and negative sampling size (5, 10, 15, and 20). Hyper-parameters were tuned with the validation set.

We compared the results using models as follows:

- **Item-KNN** [4]: A traditional collaborative filtering (CF) approach based on k-nearest-neighbor (KNN) and item–item similarities. Regularization was also included to avoid coincidental high similarities of rarely visited items. This baseline is one of the most common item-to-item solutions in practical systems. This model recommends items using previous item interactions of users. The method contains the following parameters: neighborhood size (50, 100, and 200) and regularization term (16, 32, and 64) that discounts the similarity of rare items [10].
- **Item2vec** [3]: A model derived from word2vec. This model uses the interaction of items as word co-occurrences. We used the skip-gram negative sampling model for training, which was conducted in Euclidean space. The method contains the following parameters: vector dimension, learning rate, negative sampling size, and window size (3, 7, 12, and 15) [5].
- **GRU4REC** [16]: A sequence prediction model can predict the next item using recurrent neural networks when given the items a user has interacted with in the past. This model can capture long-term user preferences, unlike the other baseline models. The method contains the following parameters: the parameter vector dimension, learning rate, and batch size [16].
- **Neural collaborative filtering (NCF)** [14]: A well-known recommendation algorithm that generalizes the matrix factorization problem with a multi-layer perceptron. The method contains the following parameters: vector dimension, hidden layers for MLP, learning rate, and batch size. We set low vector dimensions (8 and 16) to avoid overfitting and employed three hidden layers for MLP ([32, 16, 8] and [64, 32, 16]) as described in Ref. [14].
- **Light graph convolution network (LightGCN)** [13]: A simple, linear, and neat graph convolution network (GCN) model for recommendation. The method contains the following parameters: vector dimension, layers (1, 2, and 3), learning rate, and batch size.
- **Poincaré** [23]: We used the original poincaré model to get item embeddings. Because of the time and computational resource limitations of real-time stream recommender systems, we cannot embed users when there are much more users than items. Thus, we did not use the other hyperbolic embedding models [11], [30]. We ranked the items by the distance from the second latest item the user previously clicked. The method contains the following parameters: vector dimension, learning rate, negative sampling size. We set these parameters the same as common parameter settings. We used the burn-in process for five epochs at the beginning

Table 3 nDCG@10 and HR@10. In this table, L2 means the the L2 regularization and GL2 means the graph-centralized L2 regularization. Poincaré-10 means a 10-dimension Poincaré embedding.

	nDCG@10							
	MovieLens	Online Retail	Sports	Cell Phones	Games	Automotive	Clothing	Lucra
Item-KNN	0.427	0.462	0.293	0.290	0.275	0.304	0.658	0.600
Item2vec	0.383	0.337	0.539	0.294	0.335	0.326	0.659	0.674
GRU4REC	0.333	0.271	0.371	0.330	0.350	0.345	0.538	0.627
NCF	0.378	0.364	0.328	0.293	0.309	0.318	0.603	0.759
LightGCN	0.341	0.269	0.533	0.395	0.431	0.424	0.678	0.502
Poincaré	0.459	0.481	0.587	0.418	0.415	0.438	0.674	0.764
Poincaré L2	0.467	0.485	0.587	0.422	0.416	0.442	0.675	0.765
Poincaré-10 GL2	0.462	**0.490**	0.607	0.434	0.425	0.450	0.686	0.776
Poincaré GL2	**0.470**	**0.490**	**0.608**	**0.435**	**0.433**	**0.457**	**0.696**	**0.779**
	HR@10							
	MovieLens	Online Retail	Sports	Cell Phones	Games	Automotive	Clothing	Lucra
Item-KNN	0.652	0.660	0.533	0.527	0.516	0.542	0.776	0.809
Item2vec	0.640	0.639	0.689	0.584	0.591	0.589	0.769	0.888
GRU4REC	0.644	0.535	0.687	0.623	0.660	0.657	0.806	0.937
NCF	0.694	0.658	0.625	0.625	0.575	0.620	0.773	0.954
LightGCN	0.628	0.537	0.773	0.676	0.685	0.685	0.827	0.780
Poincaré	0.744	0.721	0.761	0.680	0.639	0.664	0.784	0.943
Poincaré L2	0.749	0.726	0.761	0.685	0.644	0.668	0.786	0.942
Poincaré-10 GL2	0.759	0.726	0.804	**0.734**	0.690	0.710	0.827	0.967
Poincaré GL2	**0.769**	**0.733**	**0.812**	**0.734**	**0.699**	**0.714**	**0.834**	**0.968**

of the learning phase. This burn-in technique was introduced in Ref. [23] to get a good initial angular layout that can be helpful to find good embeddings.

- **Poincaré with regularization**: We utilize the regularization to the Poincaré method in addition to the baseline settings. Hyper-parameter of proposed graph-centralized regularization term h is $\in \{0.5, 0.75, 1\}$ in Poincaré embedding method.

6.2.2 Results

Table 3 shows the evaluation accuracy of nDCG@10 and HR@10. For each of the obtained results shown in Table 3, the best result is in boldface, while the best in the baselines is underlined. Overall, the hyperbolic embedding model with our regularization outperformed the other baseline methods in both nDCG@10 and HR@10. Regarding the baselines, the hyperbolic method without our regularization outperformed the other methods in nDCG@10. LightGCN also performed well in Amazon datasets. In contrast to the LightGCN results, NCF worked well for Lucra with a dense graph structure. Item-KNN performed well in nDCG@10 about the MovieLens and Online Retail dataset. As reported in Ref. [10], a deep neural network model does not always have more permanence, and a tuned item-KNN model worked well.

To break down the regularization effects, we also show the results of the standard L2 regularization and graph-centralized L2 regularization in Table 3. We can see that the graph-centralized regularization leveraging hierarchical graph structures had more uplift in accuracy up to 8.5%, than the standard L2 regularization and had 9% more uplift than without regularization in HR@10. In the beginning of the learning phase, we used the burn-in technique [23] to improve the angular layout without moving too far towards the boundary. This technique might have a regularization effect. Even though we used the burn-in process, the evaluation accuracy could be further improved by the regularization. This means that our proposed regularization had another effect on accuracy in addition to the burn-in technique.

To see the effectiveness of extremely low-dimensional hyperbolic embedding, we also explicitly describe the result of 10-dimensional hyperbolic embedding in Table 3. Despite the low embedding dimension (10 dimensions), graph-centralized regularization can maintain the hierarchical structure in the embedding space and showed that high accuracy compared with other baselines.

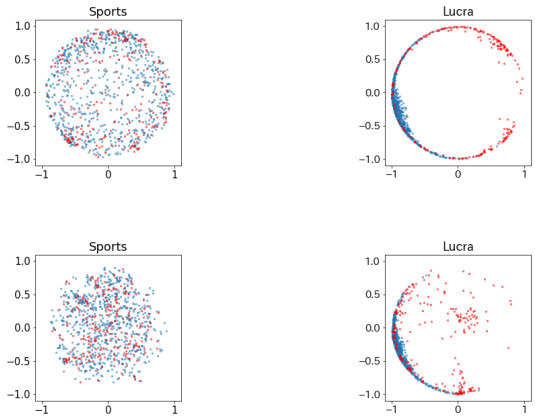

Fig. 4 2D item embedding visualization with and without regularization. The upper side of each dataset shows the results without any regularization. The lower side shows results with graph-centralized regularization.

Figure 4 shows the 2D-item embedding visualization with and without regularization. We plotted the top 25% high-degree nodes as red dots and the remainder as blue dots. Using graph-centralized regularization, the high-degree nodes tended to be closer to the origin, while the low-degree nodes tended to be far from the origin. Without regularization, we see that the items tended to be embedded in the Poincaré ball's boundary. These results demonstrate that we can maintain the graph structure when we use regularization.

Regarding **RQ1**, the hyperbolic embedding model with our regularization outperformed the other baseline methods in both nDCG@10 and HR@10 and effect was more positive than standard L2 regularization.

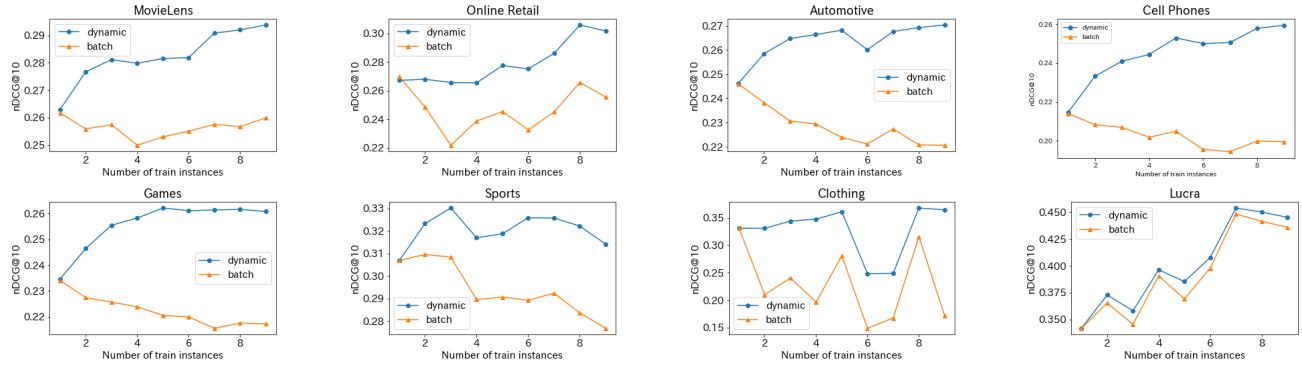

Fig. 5 Efficiency of the dynamic and batch model updates in stream settings.

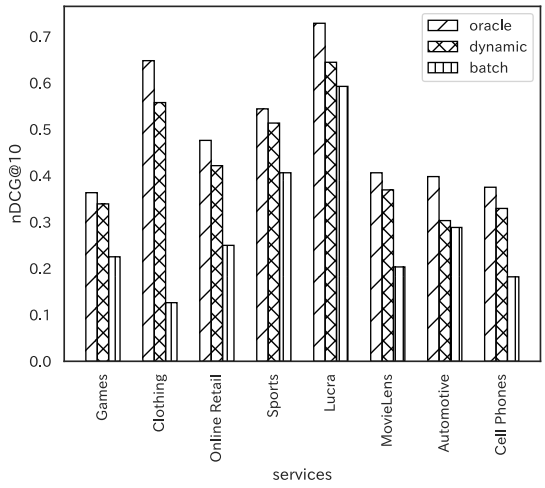

Fig. 6 Robustness of *oracle*, batch, and dynamic model updates in setting ($T = 20$).

6.3 Efficiency Evaluation

6.3.1 Settings

To simulate a real-time stream recommender setting, we generated a dynamic data stream. We first sorted all of the data by timestamp and split items into 20 sets. Next, we used each split set for training. Assuming many new items are added sequentially at short intervals, we trained the model by one epoch for each data stream. Thus, we did not use the burn-in process [23] in the Poincaré embedding in this efficiency evaluation setting. We set the window size τ to 10, started building trained data from time $t = 1$, and trained from time $t = 10$ to $t = 20$. In each testing phase for time $t = 10$ to $t = 20$, we filtered the ground truth items to be included in the current data stream. The ground truth was re-ranked with 20 negative sampling items; the performance was judged based on the ranked list. We measured nDCG@10 for ranking metrics averaged over 20 evaluations.

6.3.2 Results

Figure 5 shows the performance of the dynamic update model strategy, where the x-axis represents the number of input stream datasets, and the y-axis represents the nDCG@10. The dynamic model outperformed batch training on all datasets. By using dynamic updates, the training proceeded more efficiently and achieved higher accuracy. The difference in improvement was small in Lucra probably because of the density of the Lucra, which tended to have more edges in the streaming data, and one epoch was enough to learn the embeddings.

Figure 6 shows the results for robustness. The *oracle* model used all the data to train the embeddings, while the dynamic and batch models trained the embeddings in a window size of $\tau = 10$. After finishing all the training phases, we used the latest embeddings to measure the nDCG@10. The *oracle* model was the highest-scoring, followed by the dynamic model. The dynamic model had much better accuracy than the batch model due to its dynamic update embeddings and successfully maintained the embeddings' consistency even if they were not contained in the training data. We note that the *oracle* mode needed t/τ times the computational cost to get the results for each time t.

Therefore, **RQ2** can be answered as follows: The dynamic model update could achieve much better efficiency and robustness than the batch training.

7. A/B Testing

This section reports the results of the A/B testing. We assessed the impact on a real-world news service when the proposed method was deployed. We aimed to improve the online metrics by building compact hyperbolic embedding into the current recommender system.

7.1 Experimental Setup

From February 5, 2020, to February 10, 2020, we conducted A/B testing on the news service Lucra. We applied 5% traffic to the proposed method so as not to degrade the user experience.

We deployed our method to a *related article component*. This component displays related articles after a user reads an article. The existing logic for finding related articles uses title similarity and the click-through rate of the article. In this testing, we referred to the existing logic as the control and the logic using similarity of Poincaré embedding in addition to existing logic as the treatment.

We trained 15 epochs every 15 minutes while the user interactions accumulated sequentially with the 10-dimension Poincaré embedding. We set τ at three days to retain embeddings in the model. In Lucra, approximately 10,000 articles emerged in one day, and all of the articles from the three days were candidates for the recommendation.

7.2 Online Metrics

We used four online metrics to evaluate the impact of the pro-

Table 4 Treatment vs. Control.

Metrics	Percent Lift
Clicks	+3.79%
Click-Through Rate	+2.58%
Clicks/Unique Users	+3.92%
Dwell Time/Unique Users	+10.2%

posed methods [25].

Clicks The total number of user clicks.

Click-Through Rate The total number of user clicks divided by the number of user article views. If this value is low, the user is more engaged in searching articles than browsing articles.

Clicks/Unique Users The total number of user clicks divided by the number of unique users. It is divided by the number of unique users to remove the bias of the number of users.

Dwell Time/Unique Users Total reading time of the user's related articles divided by the number of unique users. The article viewing total time refers to the time spent reading an article after clicking on a related article.

7.3 Results

Table 4 provides a summary of the A/B test results, showing that the treatment bucket value was significantly improved for all metrics. Not only did the number of clicks increase, but dwell time per user also improved significantly. The difference came from whether we used CF-based Poincaré embeddings. The control bucket used only the title and click-through rate, while in the treatment bucket with CF-based Poincaré embeddings, there was an increase in average dwell time. Due to the Poincaré embeddings, the articles that did not contain similar titles but would be pertinent for the user became easier to recommend. As a result, the average article dwell time increased. In the treatment bucket, 96.34% of responses could be returned with trained Poincaré embeddings. In this experiment, the Poincaré embedding was 10-dimensional. Therefore, there was no degradation of system performance when we conducted A/B testing. Our response time was kept less than 50 milliseconds. Thus, Poincaré embedding is promising as a recommender component that requires few resources.

8. Conclusion

One of the key challenges of a real-time stream recommender system is how to learn compact and accurate embedding representations efficiently. In this work, we first determined that the graphs of various recommendation datasets had a hierarchical or tree-like structure suitable for hyperbolic embedding. Then, we proposed a regularization method that uses graph-centrality. Next, we proposed a model update technique for learning hyperbolic embeddings by online learning rather than by batch learning.

In the offline evaluation, we confirmed that the regularization leveraging of the hierarchical or tree-like structures improved HR@10 up to 9% when compared with a hyperbolic embedding method without regularization. In the offline experiments in real-time stream recommender settings, we confirmed that the proposed model update technique achieved more efficient and robust updates than batch updates. Finally, we deployed our method to a million-scale news recommendation service for A/B testing. The results showed that the number of clicks improved (+3.7%) and the dwell time also improved (+10%) when *10-dimension* hyperbolic embeddings were used.

In this study, we focused on CF-based Poincaré embedding employing user interactions. In the future, we plan to realize the content and a CF-based hybrid recommender system using compact hyperbolic embeddings.

References

[1] Abu-Ata, M. and Dragan, F.F.: Metric tree-like structures in real-world networks: An empirical study, *Netw*, Vol.67, No.1, pp.49–68 (2016).

[2] Barabási, A.L. and Bonabeau, E.: Scale-free networks, *Scientific American*, Vol.288, No.5, pp.60–69 (2003).

[3] Barkan, O. and Koenigstein, N.: Item2vec: Neural item embedding for collaborative filtering, *2016 IEEE 26th International Workshop on Machine Learning for Signal Processing* (*MLSP*), pp.1–6, IEEE (2016).

[4] Breese, J.S., Heckerman, D. and Kadie, C.: Empirical analysis of predictive algorithms for collaborative filtering, *Proc. 14th Conference on Uncertainty in Artificial Intelligence*, pp.43–52 (1998).

[5] Caselles-Dupré, H., Lesaint, F. and Royo-Letelier, J.: Word2vec applied to recommendation: Hyperparameters matter, *Proc. 2018 ACM Conference on Recommender Systems*, pp.352–356 (2018).

[6] Chamberlain, B.P., Hardwick, S.R., Wardrope, D.R., Dzogang, F., Daolio, F. and Vargas, S.: Scalable hyperbolic recommender systems, arXiv:1902.08648 (2019).

[7] Chami, I., Ying, Z., Ré, C. and Leskovec, J.: Hyperbolic graph convolutional neural networks, *Advances in Neural Information Processing Systems*, pp.4869–4880 (2019).

[8] Chen, D., Sain, S.L. and Guo, K.: Data mining for the online retail industry: A case study of rfm model-based customer segmentation using data mining, *Journal of Database Marketing & Customer Strategy Management*, Vol.19, No.3, pp.197–208 (2012).

[9] Chen, W., Fang, W., Hu, G. and Mahoney, M.W.: On the hyperbolicity of small-world and treelike random graphs, *Internet Mathematics*, Vol.9, No.4, pp.434–491 (2013).

[10] Dacrema, M.F., Cremonesi, P. and Jannach, D.: Are we really making much progress? a worrying analysis of recent neural recommendation approaches, *Proc. 2019 ACM Conference on Recommender Systems*, pp.101–109 (2019).

[11] Feng, S., Tran, L.V., Cong, G., Chen, L., Li, J. and Li, F.: Hme: A hyperbolic metric embedding approach for next-poi recommendation, *Proc. 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp.1429–1438 (2020).

[12] Grbovic, M. and Cheng, H.: Real-time personalization using embeddings for search ranking at airbnb, *Proc. 2018 ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp.311–320 (2018).

[13] He, X., Deng, K., Wang, X., Li, Y., Zhang, Y. and Wang, M.: Lightgcn: Simplifying and powering graph convolution network for recommendation, *Proc. 2020 International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp.639–648 (2020).

[14] He, X., Liao, L., Zhang, H., Nie, L., Hu, X. and Chua, T.S.: Neural collaborative filtering, *Proc. 2017 International Conference on World Wide Web*, pp.173–182 (2017).

[15] He, X., Zhang, H., Kan, M.Y. and Chua, T.S.: Fast matrix factorization for online recommendation with implicit feedback, *Proc. 2016 International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp.549–558 (2016).

[16] Hidasi, B., Karatzoglou, A., Baltrunas, L. and Tikk, D.: Session-based recommendations with recurrent neural networks, arXiv:1511.06939 (2015).

[17] Huang, Y., Cui, B., Zhang, W., Jiang, J. and Xu, Y.: Tencentrec: Real-time stream recommendation in practice, *Proc. 2015 ACM SIGMOD International Conference on Management of Data*, pp.227–238 (2015).

[18] Jiang, J.Y., Chen, P.H., Hsieh, C.J. and Wang, W.: Clustering and constructing user coresets to accelerate large-scale top-k recommender systems, *Proc. Web Conference 2020*, *WWW '20*, pp.2177–2187, Association for Computing Machinery (2020).

[19] Klemm, K. and Eguiluz, V.M.: Highly clustered scale-free networks, *Physical Review E*, Vol.65, No.3, 036123 (2002).

[20] Kolyvakis, P., Kalousis, A. and Kiritsis, D.: Hyperbolic knowledge graph embeddings for knowledge base completion, arXiv:1908.04895

(2019).
[21] Liang, D., Altosaar, J., Charlin, L. and Blei, D.M.: Factorization meets the item embedding: Regularizing matrix factorization with item co-occurrence, *Proc. 2016 ACM Conference on Recommender Systems*, pp.59–66 (2016).
[22] Ni, J., Li, J. and McAuley, J.: Justifying recommendations using distantly-labeled reviews and fine-grained aspects, *Proc. 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing*, pp.188–197, Association for Computational Linguistics (Nov. 2019).
[23] Nickel, M. and Kiela, D.: Poincaré embeddings for learning hierarchical representations, *Advances in Neural Information Processing Systems*, pp.6338–6347 (2017).
[24] Nickel, M. and Kiela, D.: Learning continuous hierarchies in the lorentz model of hyperbolic geometry, *International Conference on Machine Learning*, pp.3779–3788, PMLR (2018).
[25] Okura, S., Tagami, Y., Ono, S. and Tajima, A.: Embedding-based news recommendation for millions of users, *Proc. 2017 ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp.1933–1942 (2017).
[26] Ravasz, E. and Barabási, A.L.: Hierarchical organization in complex networks, *Physical Review E*, Vol.67, No.2, 026112 (2003).
[27] Rendle, S. and Schmidt-Thieme, L.: Online-updating regularized kernel matrix factorization models for large-scale recommender systems, *Proc. 2008 ACM Conference on Recommender Systems*, pp.251–258 (2008).
[28] Shimizu, R., Mukuta, Y. and Harada, T.: Hyperbolic neural networks++, arXiv preprint arXiv:2006.08210 (2020).
[29] Tay, Y., Tuan, L.A. and Hui, S.C.: Hyperbolic representation learning for fast and efficient neural question answering, *Proc. 11th ACM International Conference on Web Search and Data Mining*, pp.583–591 (2018).
[30] Vinh Tran, L., Tay, Y., Zhang, S., Cong, G. and Li, X.: Hyperml: A boosting metric learning approach in hyperbolic space for recommender systems, *Proc. 2020 International Conference on Web Search and Data Mining*, pp.609–617 (2020).
[31] Watts, D.J. and Strogatz, S.H.: Collective dynamics of 'small-world' networks, *Nature*, Vol.393, No.6684, p.440 (1998).
[32] Wu, Y. and Zhang, C.: Hyperbolicity and chordality of a graph, *The Electronic Journal of Combinatorics*, p.43 (2011).

Kojiro Iizuka received his B.S. and M.S. degrees in Computer Science from University of Tsukuba. He works at Gunosy Inc. and is a Ph.D. student at University of Tsukuba. His research interests include Recommender Systems and Information Retrieval, especially in User Behavior Analysis and Online Evaluation.

Makoto P. Kato received his B.S., M.S., and Ph.D. degrees in Informatics from Kyoto University, Japan, in 2008, 2009, and 2012, respectively. Since 2019, he has been an associate professor of Faculty of Library, Information and Media Science, University of Tsukuba. His research interests include Information Retrieval, Web Mining, and Machine Learning.

Yoshifumi Seki is co-founder of Gunosy Inc. He co-developed Gunosy in 2011 when he was a postgraduate student and eventually co-founded the company in 2012. This company was listed in Tokyo Stock Exchange Market in 2015. In parallel, he received his Ph.D. in Engineering from the University of Tokyo in 2017. He's been responsible for news delivery algorithms at the company and has deep expertise in Web Mining, focusing on Recommender Systems and application of Machine Learning. Currently, he is engaged in research and development focusing on Recommender Systems, User Behavior Analysis.

(Editor in Charge: *Masayoshi Aritsugi*)