PinnerSage: Multi-Modal User Embedding Framework for Recommendations at Pinterest

Aditya Pal*, Chantat Eksombatchai*, Yitong Zhou*, Bo Zhao, Charles Rosenberg, Jure Leskovec
Pinterest Inc.
{apal,pong,yzhou,bozhao,crosenberg,jure}@pinterest.com

ABSTRACT

Latent user representations are widely adopted in the tech industry for powering personalized recommender systems. Most prior work infers a *single* high dimensional embedding to represent a user, which is a good starting point but falls short in delivering a full understanding of the user's interests. In this work, we introduce PinnerSage, an end-to-end recommender system that represents each user via multi-modal embeddings and leverages this rich representation of users to provides high quality personalized recommendations. PinnerSage achieves this by clustering users' actions into conceptually coherent clusters with the help of a hierarchical clustering method (Ward) and summarizes the clusters via representative pins (Medoids) for efficiency and interpretability. PinnerSage is deployed in production at Pinterest and we outline the several design decisions that makes it run seamlessly at a very large scale. We conduct several offline and online A/B experiments to show that our method significantly outperforms single embedding methods.

CCS CONCEPTS

• **Information systems** → **Recommender systems**; **Personalization**; *Clustering*; *Nearest-neighbor search.*

KEYWORDS

Personalized Recommender System; Multi-Modal User Embeddings

ACM Reference Format:

Aditya Pal, Chantat Eksombatchai, Yitong Zhou, Bo Zhao, Charles Rosenberg, Jure Leskovec. 2020. PinnerSage: Multi-Modal User Embedding Framework for Recommendations at Pinterest. In *Proceedings of the 26th ACM SIGKDD Conference on Knowledge Discovery & Data Mining (KDD '20), August 23–27, 2020, Virtual Event, CA, USA.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3394486.3403280

1 INTRODUCTION

Pinterest is a content discovery platform that allows 350M+ monthly active users to collect and interact with 2B+ visual bookmarks called *pins*. Each pin is an image item associated with contextual text, representing an idea that users can find and bookmark from around the world. Users can save pins on *boards* to keep them organized and easy to find. With billions of pins on Pinterest, it becomes crucial to help users find those ideas (Pins) which would spark inspiration. Personalized recommendations thus form an essential component of the Pinterest user experience and is pervasive in our products. The Pinterest recommender system spans a variety of algorithms that collectively define the experience on the homepage. Different algorithms are optimized for different objectives and include - (a) homefeed recommendations where a user can view an infinite recommendation feed on the homepage (as shown in Figure 1), (b) shopping product recommendations which link to 3rd party e-commerce sites, (c) personalized search results, (d) personalized ads, (e) personalized pin board recommendations, etc. Therefore, it becomes necessary to develop a universal, shareable and rich understanding of the user interests to power large-scale cross-functional use cases at Pinterest.

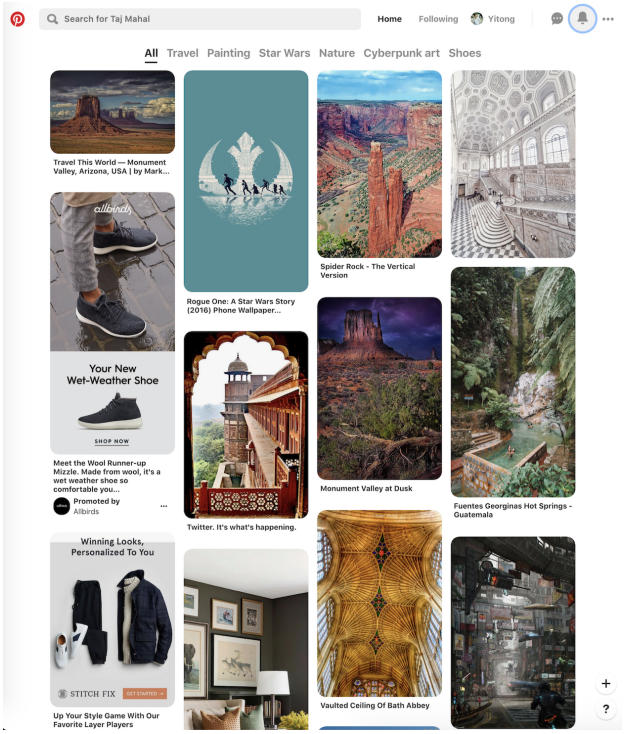

Figure 1: Pinterest Homepage.

Latent user representation methods have become increasingly important in advancing our understanding of users. They are shown to be effective at powering collaborative filtering techniques [15, 20], and serving as features in ranking models [6, 7, 26, 28]. Due to their efficacy, user embeddings are widely adopted in various industry settings. They are used to power YouTube and Google play

*These authors contributed equally.

KDD '20, August 23–27, 2020, Virtual Event, CA, USA

ACM ISBN 978-1-4503-7998-4/20/08.
https://doi.org/10.1145/3394486.3403280

recommendations [6, 7], personalize search ranking and similar listing recommendations at Airbnb [11], recommend news articles to users [19], connect similar users at Etsy [30], etc.

Building an effective user embedding system that provides personalized recommendations to hundreds of millions of users from a candidate pool of billions of items has several inherent challenges. *The foremost challenge is how to effectively encode multiple facets of a user?* A user can have a diverse set of interests with no obvious linkage between them. These interests can evolve, with some interests persisting long term while others spanning a short time period. Most of the prior work aims to capture the rich diversity of a user's actions and interests via a single high-dimensional embedding vector. Typically, items to be recommended are also represented in the same embedding space. This is initially satisfying, but as pointed by research work described in [3, 10, 16, 25], a good embedding must encode user's multiple tastes, interests, styles, etc., whereas an item (a video, an image, a news article, a house listing, a pin, etc) typically only has a single focus. Hence, an attention layer [29] or other context adapting approaches is needed to keep track of the evolving interests of the users.

One alternative that has shown promise is to represent a user with multiple embeddings, with each embedding capturing a specific aspect of the user. As shown by [25], multi-embedding user representations can deliver 25% improvement in YouTube video recommendations. [16] also shows reasonable gains on small benchmark datasets. However, multi-embedding models are not widely adopted in the industry due to several important questions and concerns that are not yet fully addressed by prior work:

- How many embeddings need to be considered per user?
- How would one run inference at scale for hundreds of millions of users and update the embeddings ?
- How to select the embeddings to generate personalized recommendations?
- Will the multi-embedding models provide any significant gains in a production setting?

Most of the prior multi-embedding work side-steps these challenges by either running only small-scale experiments and not deploying these techniques in production or by limiting a user to very few embeddings, thereby restricting the utility of such approaches.

Present Work. In this paper, we present an end-to-end system, called PinnerSage, that is deployed in production at Pinterest. PinnerSage is a highly scalable, flexible and extensible recommender system that internally represents each user with multiple PinSage [27] embeddings. It infers multiple embeddings via hierarchical clustering of users' actions into conceptual clusters and uses an efficient representation of the clusters via medoids. Then, it employs a highly efficient nearest neighbor system to power candidate generation for recommendations at scale. Finally, we evaluate PinnerSage extensively via offline and online experiments. We conduct several large scale A/B tests to show that PinnerSage based recommendations result in significant engagement gains for Pinterest's homefeed, and shopping product recommendations

2 PINNERSAGE DESIGN CHOICES

We begin by discussing important design choices of PinnerSage.

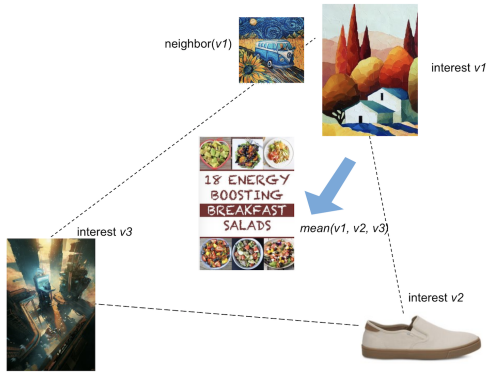

Figure 2: Pins of 256-dimensional embeddings visualized in 2D. These pins depict three different interests of a user.

Design Choice 1: Pin Embeddings are Fixed. Most prior work, jointly learns user and item embeddings (e.g. [7, 11]). This causes inherent problems in large-scale applications, such that it unnecessarily complicates the model, slows the inference computation, and brings in difficulty for real-time updates. Besides these, we argue that it can often lead to less desirable side-effects. Consider the toy example in Figure 2, where a user is interested in *painting*, *shoes*, and *sci-fi*. Jointly learnt user and pin embeddings would bring pin embeddings on these disparate topics closer, which is actually what we wish to *avoid*. Pin embeddings should only operate on the underlying principle of bringing similar pins closer while keeping the rest of the pins as far as possible. For this reason, we use PinSage [27], which precisely achieves this objective without any dilution. PinSage is a unified pin embedding model, which integrates visual signals, text annotations, and pin-board graph information to generate high quality pin embeddings. An additional advantage of this design choice is that it considerably simplifies our downstream systems and inference pipelines.

Design Choice 2: No Restriction on Number of Embeddings. Prior work either fixes the number of embeddings to a small number [25] or puts an upper bound on them [16]. Such restrictions at best hinders developing a full understanding of the users and at worst merges different concepts together leading to bad recommendations. For example, merging item embeddings, which is considered reasonable (see [7, 28]), could yield an embedding that lies in a very different region. Figure 2 shows that a merger of three disparate pin embeddings results in an embedding that is best represented by the concept *energy boosting breakfast*. Needless to say, recommendations based on such a merger can be problematic.

Our work allows a user to be represented by as many embeddings as their underlying data supports. This is achieved by clustering users' actions into conceptually coherent clusters via a hierarchical agglomerative clustering algorithm (Ward). A light user might get represented by 3-5 clusters, whereas a heavy user might get represented by 75-100 clusters.

Design Choice 3: Medoids based Representation of Clusters. Typically, clusters are represented by centroid, which requires storing an embedding. Additionally, centroid could be sensitive to outliers in the cluster. To compactly represent a cluster, we pick a

cluster member pin, called medoid. Medoid by definition is a member of the user's originally interacted pin set and hence avoids the pit-fall of topic drift and is robust to outliers. From a systems perspective, medoid is a concise way of representing a cluster as it only requires storage of medoid's pin id and also leads to cross-user and even cross-application cache sharing.

Design Choice 4: Medoid Sampling for Candidate Retrieval. PinnerSage provides a rich representation of a user via cluster medoids. However, in practice we cannot use all the medoids simultaneously for candidate retrieval due to cost concerns. Additionally, the user would be bombarded with too many different items. Due to cost concerns, we sample 3 medoids proportional to their importance scores (computation described in later sections) and recommend their nearest neighboring pins. The importance scores of medoids are updated daily and they can adapt with changing tastes of the user.

Design Choice 5: Two-pronged Approach for Handling Real-Time Updates. It is important for a recommender system to adapt to the current needs of their users. At the same time an accurate representation of users requires looking at their past 60-90 day activities. Sheer size of the data and the speed at which it grows makes it hard to consider both aspects together. Similar to [11], we address this issue by combining two methods: (a) a daily batch inference job that infers multiple medoids per user based on their long-term interaction history, and (b) an online version of the same model that infers medoids based on the users' interactions on the current day. As new activity comes in, only the online version is updated. At the end of the day, the batch version consumes the current day's activities and resolves any inconsistencies. This approach ensures that our system adapts quickly to the users' current needs and at the same time does not compromise on their long-term interests.

Design Choice 6: Approximate Nearest Neighbor System. To generate embeddings based recommendations, we employ an approximate nearest neighbor (ANN) system. Given a query (medoid), the ANN system fetches k pins closest to the query in the embedding space. We show how several improvements to the ANN system, such as filtering low quality pins, careful selection of different indexing techniques, caching of medoids; results in the final production version having 1/10 the cost of the original prototype.

3 OUR APPROACH

Notations. Let the set $\mathcal{P} = \{P_1, P_2, \ldots\}$ represent all the pins at Pinterest. The cardinality of $\mathcal{P}$ is in order of billions. Here, $P_i \in \mathbb{R}^d$ denotes the d-dimensional PinSage embedding of the i^{th} pin. Let $\mathcal{A}_u = \{a_1, a_2, \ldots\}$ be the sequence of action pins of user u, such that for each $a \in \mathcal{A}_u$, user either *repinned*, or *clicked* pin P_a at time $\mathcal{T}_u[a]$. For the sake of simplicity, we drop the subscript u as we formulate for a single user u, unless stated otherwise. We consider the action pins in $\mathcal{A}$ to be sorted based on action time, such that a_1 is the pin id of the first action of the user.

Main Assumption: Pin Embeddings are Fixed. As mentioned in our design choice 1 (Section 2), we consider pin embeddings to be fixed and generated by a black-box model. Within Pinterest, this model is PinSage [27] that is trained to place similar pins nearby in the embedding space with the objective of subsequent retrieval. This assumption is ideal in our setting as it considerably simplifies the complexity of our models. We also made a similar assumption in our prior work [28].

Table 1: Accuracy lift of models on predicting the next user action. The lifts are computed w.r.t. last pin model.

Models	Accuracy Lift
Last pin	0%
Decay average	25%
Kmeans Oracle	98%
Oracle (*practical upper-bound*)	**140%**

Main Goal. Our main goal is to infer multiple embeddings for each user, $\mathcal{E} = \{e_1, e_2, \ldots\}$, where $e_i \in \mathbb{R}^d$ for all i, given a user's actions $\mathcal{A}$ and pins embeddings $\mathcal{P}$. Since pin embeddings are fixed and hence *not* jointly inferred, we seek to learn each e_i compatible with pin embeddings – *specifically for the purpose of retrieving similar pins to* e_i. For different users, the number of embeddings can be different, i.e. $|\mathcal{E}_u|$ need *not* be same as $|\mathcal{E}_v|$. However, for our approach to be practically feasible, we *require the number of embeddings to be in order of tens to hundreds* ($|\mathcal{E}| << |\mathcal{P}|$).

To show the promise of the clustering-based approach, we consider a task of predicting the next user action. We have access to the user's past actions $\{a_1, a_2, \ldots, a_i\}$ and our goal is to predict the next action a_{i+1} that the user is going to interact with from a corpus of billions of pins. To simplify this challenge, we measure the performance by asking how often is the distance between the user-embedding and the pin embedding $P_{a_{i+1}}$ is above a cosine threshold of 0.8. We compare four single embedding approaches:

(1) *Last pin*: User representation is the embedding of the last action pin (P_{a_i}).
(2) *Decay average*: User representation is a time-decayed average of embeddings of their action pins. Specifically, decay average embedding $\propto \sum_{a \in \mathcal{A}} e^{-\lambda(\mathcal{T}_{now} - \mathcal{T}[a])} \cdot P_a$.
(3) *Oracle*: Oracle can "look into the future" and pick as the user representation the past action pin of the user that is closest to a_{i+1}. This measures the upper bound on accuracy of a method that would predict future engagements based on past engagements.
(4) *Kmeans Oracle*: User is represented via k-means clustering ($k = 3$) over their past action pins. Again, the Oracle gets to see pin a_{i+1} and picks as the user representation the cluster centroid closest to it.

Table 1 shows the results. Oracle model provides substantial accuracy gains, deservedly so as it can look at the future pin. However, its superior performance is only possible because it is able to recall the embedding of all the pins (obviously not practical from a systems point-of-view). Interestingly, a clustering based Oracle that only has to recall 3 cluster centroid embeddings improves over the baselines by a large margin. This result is not entirely surprising because users have multiple interests and they often switch between those interests. Figure 3 depicts such an example, which is replete in our setting. We note that none of the past 5 pins correlate

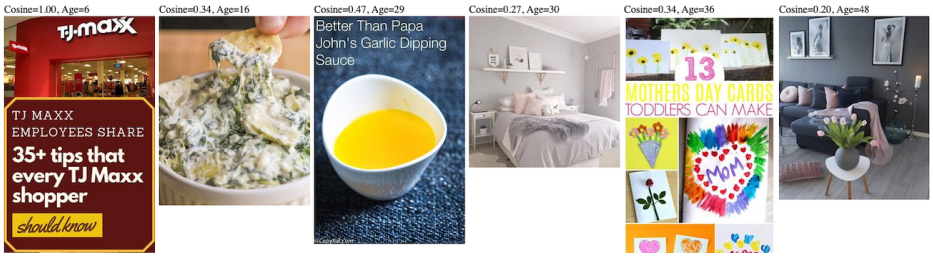

Figure 3: Snapshot of action pins (*repins* or *clicks*) of a random user. Cosine score for a pin is the cosine similarity between its embedding and the latest pin. Age of a pin is the number of days elapsed from the action date to the data collection date.

with the latest pin and one has to look further back to find stronger correlations. Hence, single embedding models with limited memory fail at this challenge.

3.1 PinnerSage

We draw two key insights from the previous task: (i) It is too limiting to represent a user with a single embedding, and (ii) Clustering based methods can provide a reasonable trade-off between accuracy and storage requirement. These two observations underpin our approach, which has the following three components.

(1) Take users' action pins from the last 90 days and cluster them into a small number of clusters.
(2) Compute a medoid based representation for each cluster.
(3) Compute an importance score of each cluster to the user.

3.1.1 ***Step 1: Cluster User Actions.*** We pose two main constraints on our choice of clustering methods.

- The clusters should *only* combine conceptually similar pins.
- It should automatically determine the number of clusters to account for the varying number of interests of a user.

The above two constraints are satisfied by Ward [24], which is a hierarchical agglomerative clustering method, that is based on a minimum variance criteria (satisfying our constraint 1). Additionally, the number of clusters is automatically determined based on the distances (satisfying our constraint 2). In our experiments (sec. 5), it performed better than K-means and complete linkage methods [8]. Several other benchmark tests also establish the superior performance of Ward over other clustering methods [1].

Our implementation of Ward is adapted from the Lance-Williams algorithm [14] which provided an efficient way to do the clustering. Initially, Algorithm 1 assigns each pin to its own cluster. At each subsequent step, two clusters that lead to a minimum increase in within-cluster variance are merged together. Suppose after some iterations, we have clusters $\{C_1, C_2, \ldots\}$ with distance between clusters C_i and C_j represented as d_{ij}. Then if two clusters C_i and C_j are merged, the distances are updated as follows:

$$d(C_i \cup C_j, C_k) = \frac{(n_i + n_k)\, d_{ik} + (n_j + n_k)\, d_{jk} - n_k\, d_{ij}}{n_i + n_j + n_k} \quad (1)$$

where $n_i = |C_i|$ is the number of pins in cluster i.

Computational Complexity of Ward Clustering Algorithm. The computational complexity of Ward clustering is $\mathbb{O}(m^2)$, where

[1] https://jmonlong.github.io/Hippocamplus/2018/02/13/tsne-and-clustering/

Algorithm 1: Ward($\mathcal{A} = \{a_1, a_2, \ldots\}, \alpha$)

Input : $\mathcal{A}$ - action pins of a user
α - cluster merging threshold
Output: Grouping of input pins into clusters
// Initial set up: each pin belongs to its own cluster
Set $C_i \leftarrow \{i\}, \forall i \in \mathcal{A}$
Set $d_{ij} \leftarrow ||P_i - P_j||_2^2, \forall i, j \in \mathcal{A}$
$merge\_history = []$
$stack = []$
while $|\mathcal{A}| > 1$ **do**
  // put first pin from $\mathcal{A}$ (without removing it) to the stack
  $stack.add(\mathcal{A}.first())$
  while $|stack| > 0$ **do**
    $i \leftarrow stack.top()$
    $J \leftarrow \{j : d_{ij} = \min_{j \neq i, j \in \mathcal{A}}\{d_{ij}\}\}$
    $merged = False$
    if $|stack| \geq 2$ **then**
      $j = stack.second\_from\_top()$
      if $j \in J$ **then**
        // merge clusters i and j
        $stack.pop()$; stack.pop(); //remove i and j
        merge_history.add($C_i \leftarrow C_j, d_{ij}$)
        $\mathcal{A} \leftarrow \mathcal{A} - \{j\}$
        Set $d_{ik} \leftarrow d(C_i \cup C_j, C_k), \forall k \in \mathcal{A}, k \neq i$
        $C_i \leftarrow C_i \cup C_j$
        $merged = True$
    if $merged = False$ **then**
      // push first element of J in the stack
      $stack.push(J.first())$
Sort tuples in $merge\_history$ in decreasing order of d_{ij}
Set $\mathbb{C} \leftarrow \{\}$
foreach $(C_i \leftarrow C_j, d_{ij}) \in merge\_history$ **do**
  if $d_{ij} \leq \alpha$ *and* $\{C_i \cup C_j\} \cap C = \emptyset, \forall C \in \mathbb{C}$ **then**
    // add $C_i \cup C_j$ to the set of clusters
    Set $\mathbb{C} \leftarrow \mathbb{C} \cup \{C_i \cup C_j\}$
return $\mathbb{C}$

$m = |\mathcal{A}|^2$. To see this, we note that in every outer while loop a cluster is added to the *empty* stack. Now since a cluster cannot be added twice to the stack (see Appendix, Lemma 8.2), the algorithm has to start merging the clusters once it cycles through all the m

Algorithm 2: PinnerSage ($\mathcal{A}, \alpha, \lambda$)

Set $\mathbb{C} \leftarrow \text{Ward}(\mathcal{A}, \alpha)$
foreach $C \in \mathbb{C}$ **do**
  Set $medoid_C \leftarrow \arg\min_{m \in C} \sum_{j \in C} ||P_m - P_j||_2^2$
  Set $importance_C \leftarrow \sum_{i \in C} e^{-\lambda(\mathcal{T}_{now} - \mathcal{T}[i])}$
return $\{medoid_C, importance_C : \forall C \in \mathbb{C}\}$

clusters (worst case). The step that leads to addition of a cluster to the stack or merging of two clusters has a computational cost of $\mathbb{O}(m)$. The algorithm operates with m *initial clusters* and then $m-1$ *intermediate merged clusters* as it progresses, leading to the total computational complexity of $\mathbb{O}(m^2)$.

3.1.2 ***Step 2: Medoid based Cluster Representation.*** After a set of pins are assigned to a cluster, we seek a compact representation for that cluster. A typical approach is to consider cluster centroid, time decay average model or a more complex sequence models such as LSTM, GRU, etc. However one problem with the aforementioned techniques is that the embedding inferred by them could lie in a very different region in the d-dimensional space. This is particularly true if there are outlier pins assigned to the cluster, which could lead to large with-in cluster variance. The side-effect of such an embedding would be retrieval of non-relevant candidates for recommendation as highlighted by Figure 2.

We chose a more robust technique that selects a cluster member pin called as *medoid* to represent the cluster. We select the pin that minimizes the sum of squared distances with the other cluster members. Unlike centroid or embeddings obtained by other complex models, medoid by definition is a point in the d-dimensional space that coincides with one of the cluster members. Formally,

$$embedding(C) \leftarrow P_m, \text{ where } m = \arg\min_{m \in C} \sum_{j \in C} ||P_m - P_j||_2^2 \quad (2)$$

An additional benefit of medoid is that we only need to store the index m of the medoid pin as its embedding can be fetched on demand from an auxiliary key-value store.

3.1.3 ***Step 3: Cluster Importance.*** Even though the number of clusters for a user is small, it can still be in order of tens to few hundreds. Due to infra-costs, we cannot utilize all of them to query the nearest neighbor system; making it essential to identify the relative importance of clusters to the user so we can sample the clusters by their importance score. We consider a time decay average model for this purpose:

$$Importance(C, \lambda) = \sum_{i \in C} e^{-\lambda(\mathcal{T}_{now} - \mathcal{T}[i])} \quad (3)$$

where $\mathcal{T}[i]$ is the time of action on pin i by the user. A cluster that has been interacted with frequently and recently will have higher importance than others. Setting $\lambda = 0$ puts more emphasis on the frequent interests of the user, whereas $\lambda = 0.1$ puts more emphasis on the recent interests of the user. We found $\lambda = 0.01$ to be a good balance between these two aspects.

Algorithm 2 provides an end-to-end overview of PinnerSage model. We note that our model operates independently for each user and hence it can be implemented quite *efficiently in parallel* on a MapReduce based framework. We also maintain an online version of PinnerSage that is run on the most recent activities of the user. The output of the batch version and the online version are combined together and used for generating the recommendations.

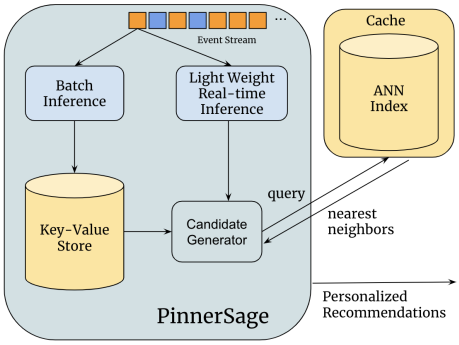

Figure 4: PinnerSage Recommendation System.

4 PINNERSAGE RECOMMENDATION SYSTEM

PinnerSage can infer as many medoids for a user as the underlying data supports. This is great from a user representation point of view, however not all medoids can be used simultaneously at any given time for generating recommendations. For this purpose, we consider importance sampling. We sample a maximum of 3 medoids per user at any given time. The sampled medoids are then used to retrieve candidates from our nearest-neighbor system. Figure 4 provides an overview of PinnerSage recommendation system.

4.1 Approximate Nearest Neighbor Retrieval

Within Pinterest, we have an approximate nearest neighbor retrieval system (ANN) that maintains an efficient index of pin embeddings enabling us to retrieve similar pins to an input query embedding. Since it indexes billions of pins, ensuring that its infrastructure cost and latency is within the internally prescribed limits is an engineering challenge. We discuss a few tricks that have helped ANN become a first-class citizen alongside other candidate retrieval frameworks, such as Pixie [9].

4.1.1 ***Indexing Scheme.*** Many different embedding indexing schemes (see [12]) were evaluated, such as LSH Orthoplex [1, 21], Product Quantization [2], HNSW [17], etc. We found HNSW to perform best on cost, latency, and recall. Table 2 shows that HNSW leads to a significant cost reduction over LSH Orthoplex. Superior performance of HNSW has been reported on several other benchmark datasets as well [2].

Candidate Pool Refinement. A full index over billions of pins would result in retrieving many near-duplicates. These near duplicates are not desirable from recommendation purposes as there is limited value in presenting them to the user. Furthermore, some pins can have intrinsically lower quality due to their aesthetics (low resolution or large amount of text in the image). We filter out near duplicates and lower quality pins via specialized in-house models.

[2] https://erikbern.com/2018/06/17/new-approximate-nearest-neighbor-benchmarks.html

Table 2: Relative Cost Benefits of Optimization Techniques.

Optimization Technique			Cost
LSH Orthoplex	→	HNSW	-60%
Full Index	→	Index Refinement	-50%
Cluster Centroid	→	Medoid	-75%

Table 2 shows that index refinement leads to a significant reduction in serving cost.

Caching Framework. All queries to the ANN system are formulated in pin embedding space. These embeddings are represented as an array of d floating point values that are not well suited for caching. On the other-hand, medoid's pin id is easy to cache and can reduce repeated calls to the ANN system. This is particularly true for popular pins that appear as medoids for multiple users. Table 2 shows the cost reduction of using medoids over centroids.

4.2 Model Serving

The main goal of PinnerSage is to recommend relevant content to the users based on their past engagement history. At the same time, we wish to provide recommendations that are relevant to actions that a user is performing in the real-time. One way to do this is by feeding all the user data to PinnerSage and run it as soon as user takes an action. However this is practically not feasible due to cost and latency concerns: We consider a two pronged approach:

(1) *Daily Batch Inference*: PinnerSage is run daily over the last 90 day actions of a user on a MapReduce cluster. The output of the daily inference job (list of medoids and their importance) are served online in key-value store.
(2) *Lightweight Online Inference*: We collect the most recent 20 actions of each user on the latest day (after the last update to the entry in the key-value store) for online inference. PinnerSage uses a real-time event-based streaming service to consume action events and update the clusters initiated from the key-value store.

In practice, the system optimization plays a critical role in enabling the productionization of PinnerSage. Table 2 shows a rough estimation of cost reduction observed during implementation. While certain limitations are imposed in the PinnerSage framework, such as a two pronged update strategy, the architecture allows for easier improvements to each component independently.

5 EXPERIMENT

Here we evaluate PinnerSage and empirically validate its performance. We start with qualitative assessment of PinnerSage, followed by A/B experiments and followed by extensive offline evaluation.

5.1 PinnerSage Visualization

Figure 5 is a visualization of PinnerSage clusters for a given user. As can be seen, PinnerSage does a great job at generating conceptually consistent clusters by grouping only contextually similar pins together. Figure 6 provides an illustration of candidates retrieved by PinnerSage. The recommendation set is a healthy mix of pins that are relevant to the top three interests of the user: *shoes*, *gadgets*, and *food*. Since this user has interacted with these topics in the past, they are likely to find this diverse recommendation set interesting and relevant.

Table 3: A/B experiments across Pinterest surfaces. Engagement gain of PinnerSage vs current production system.

Experiment	Volume	Propensity
Homefeed	+4%	+2%
Shopping	+20%	+8%

5.2 Large Scale A/B Experiments

We ran large scale A/B experiments where users are randomly assigned either in control or experiment groups. Users assigned to the experiment group experience PinnerSage recommendations, while users in control get recommendations from the single embedding model (decay average embedding of action pins). Users across the two groups are shown equal number of recommendations. Table 3 shows that PinnerSage provides significant engagement gains on increasing overall engagement volume (repins and clicks) as well as increasing engagement propensity (repins and clicks per user). Any gain can be directly attributed to increased quality and diversity of PinnerSage recommendations.

5.3 Offline Experiments

We conduct extensive offline experiments to evaluate PinnerSage and its variants w.r.t. baseline methods. We sampled a large set of users (*tens of millions*) and collected their past activities (*actions* and *impressions*) in the last 90 days. All activities before the day d are marked for training and from d onward for testing.

Baselines. We compare PinnerSage with the following baselines: (a) single embedding models, such as last pin, decay avg with several choices of λ $(0, 0.01, 0.1, 0.25)$, LSTM, GRU, and HierTCN[28], (b) multi-embedding variants of PinnerSage with different choices of (i) clustering algorithm, (ii) cluster embedding computation methods, and (iii) parameter λ for cluster importance selection.

Similar to [28], baseline models are trained with the objective of ranking user actions over impressions with several loss functions (l2, hinge, log-loss, etc). Additionally, we trained baselines with several types of negatives besides impressions, such as random pins, popular pins, and hard negatives by selecting pins that are similar to action pins.

Evaluation Method. The embeddings inferred by a model for a given user are evaluated on future actions of that user. Test batches are processed in chronological order, first day d, then day $d+1$, and so on. Once evaluation over a test batch is completed, that batch is used to update the models; mimicking a daily batch update strategy.

5.3.1 ***Results on Candidate Retrieval Task.*** Our main use-case of user embeddings is to retrieve relevant candidates for recommendation out of a very large candidate pool (billions of pins). The candidate retrieval set is generated as follows: Suppose a model outputs e embeddings for a user, then $\lfloor \frac{400}{e} \rfloor$ nearest-neighbor pins are retrieved per embedding, and finally the retrieved pins are combined to create a recommendation set of size ≤ 400 (due to overlap

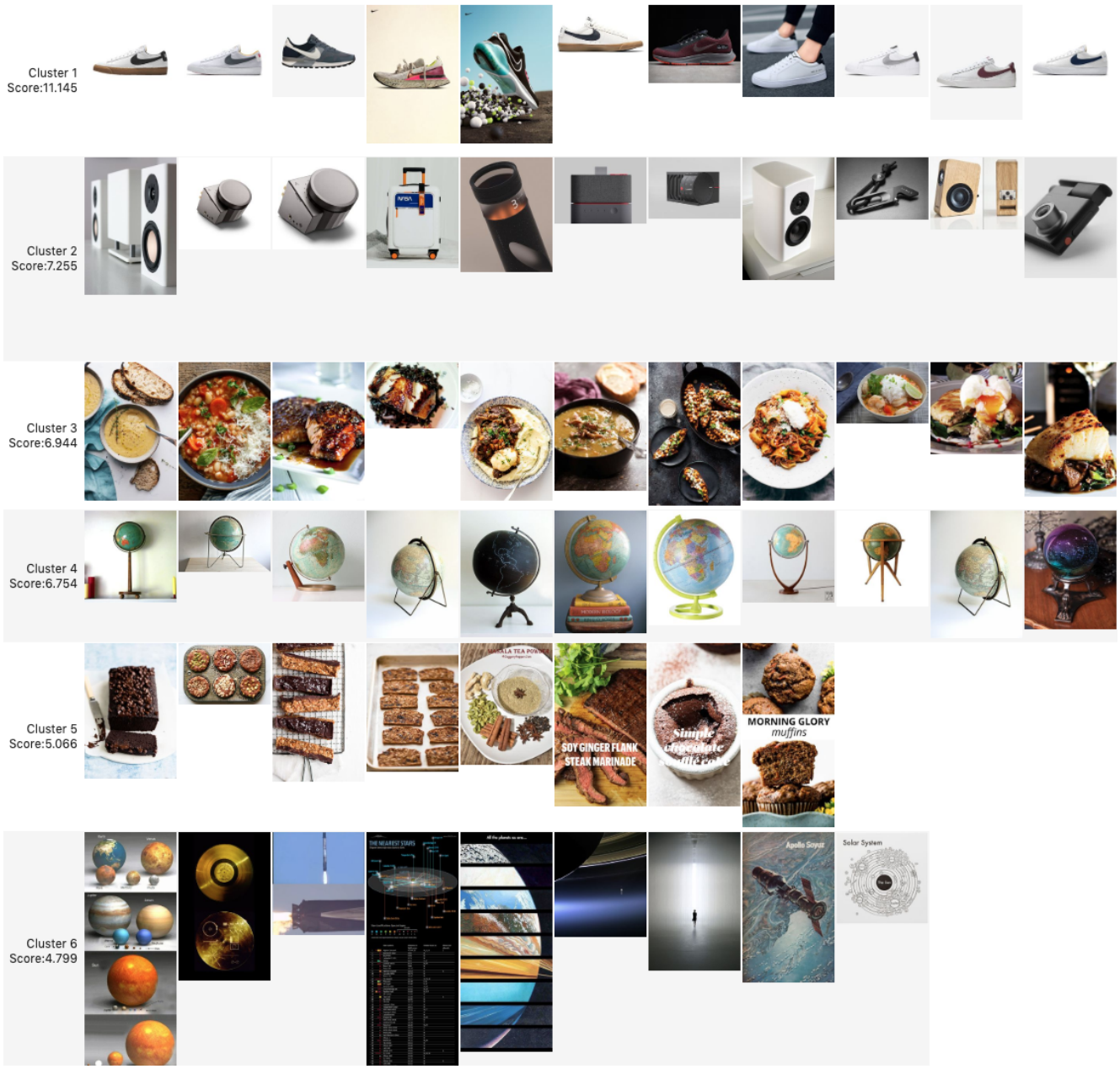

Figure 5: PinnerSage clusters of an anonymous user.

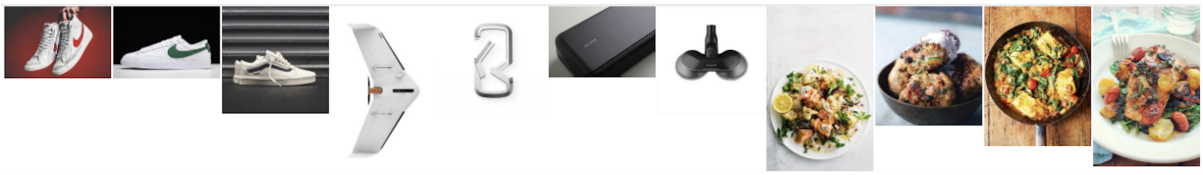

Figure 6: Sample recommendations generated by PinnerSage for the top 3 clusters 5.

Table 4: Lift relative to *last pin model* for retrieval task.

	Rel.	***Recall***
Last pin model	0%	0%
Decay avg. model ($\lambda = 0.01$)	28%	14%
Sequence models (HierTCN)	31%	16%
PinnerSage (sample 1 embedding)	33%	18%
PinnerSage (K-means(k=5))	91%	68%
PinnerSage (Complete Linkage)	88%	65%
PinnerSage (embedding = Centroid)	105%	81%
PinnerSage (embedding = HierTCN)	**110%**	**88%**
PinnerSage (importance $\lambda = 0$)	97%	72%
PinnerSage (importance $\lambda = 0.1$)	94%	69%
PinnerSage (Ward, Medoid, $\lambda = 0.01$)	**110%**	**88%**

Table 5: Lift relative to *last pin model* for ranking task.

	R-Prec.	***Rec. Rank***
Last pin model	0%	0%
Decay avg. model ($\lambda = 0.01$	8%	7%
Sequence models (HierTCN)	21%	16%
PinnerSage (sample 1 embedding)	24%	18%
PinnerSage (Kmeans(k=5))	32%	24%
PinnerSage (Complete Linkage)	29%	22%
PinnerSage (embedding = Centroid)	**37%**	**28%**
PinnerSage (embedding = HierTCN)	**37%**	**28%**
PinnerSage (importance $\lambda = 0$)	31%	24%
PinnerSage (importance $\lambda = 0.1$)	30%	24%
PinnerSage (Ward, Medoid, $\lambda = 0.01$)	**37%**	**28%**

it can be less than 400). The recommendation set is evaluated with the observed user actions from the test batch on the following two metrics:

(1) **Relevance (*Rel.*)** is the proportion of observed action pins that have high cosine similarity ($\geq$ 0.8) with any recommended pin. Higher relevance values would increase the chance of user finding the recommended set useful.
(2) ***Recall*** is the proportion of action pins that are found in the recommendation set.

Table 4 shows that PinnerSage is more effective at retrieving relevant candidates across all baselines. In particular, the single embedding version of PinnerSage is better than the state-of-art single embedding sequence methods.

Amongst PinnerSage variants, we note that Ward performs better than K-means and complete link methods. For cluster embedding computation, both sequence models and medoid selection have similar performances, hence we chose medoid as it is easier to store and has better caching properties. Cluster importance with $\lambda = 0$, which is same as counting the number of pins in a cluster, performs worse than $\lambda = 0.01$ (our choice). Intuitively this makes sense as higher value of λ incorporates recency alongside frequency. However, if λ is too high then it over-emphasize recent interests, which can compromise on long-term interests leading to a drop in model performance ($\lambda = 0.1$ vs $\lambda = 0.01$).

5.3.2 ***Results on Candidate Ranking Task.*** A user embedding is often used as a feature in a ranking model especially to rank candidate pins. The candidate set is composed of action and impression pins from the test batch. To ensure that every test batch is weighted equally, we randomly sample 20 impressions per action. In the case when there are less than 20 impressions in a given test batch, we add random samples to maintain the 1:20 ratio of actions to impressions. Finally the candidate pins are ranked based on the decreasing order of their maximum cosine similarity with any user embedding. A better embedding model should be able to rank actions above impressions. This intuition is captured via the following two metrics:

(1) **R-Precision (*R-Prec.*)** is the proportion of action pins in top-k, where k is the number of actions considered for ranking against impressions. RP is a measure of signal-to-noise ratio amongst the top-k ranked items.
(2) **Reciprocal Rank (*Rec. Rank*)** is the average reciprocal rank of action pins. It measures how high up in the ranking are the action pins placed.

Table 5 shows that PinnerSage significantly outperforms the baselines, indicating the efficacy of user embeddings generated by it as a stand-alone feature. With regards to single embedding models, we make similar observations as for the retrieval task: single embedding PinnerSage infers a better embedding. Amongst PinnerSage variants, we note that the ranking task is less sensitive to embedding computation and hence centroid, medoid and sequence models have similar performances as the embeddings are only used to order pins. However it is sensitive to cluster importance scores as that determines which 3 user embeddings are picked for ranking.

5.3.3 ***Diversity Relevance Tradeoff.*** Recommender systems often have to trade between relevance and diversity [5]. This is particularly true for single embedding models that have limited focus. On the other-hand a multi-embedding model offers flexibility of covering disparate user interests simultaneously. We define diversity to be average pair-wise *cosine distance* between the recommended set.

Figure 7 shows the diversity/relevance lift w.r.t. last pin model. We note that by increasing e, we increase both relevance and diversity. This intuitively makes sense as for larger e, the recommendation set is composed of relevant pins that span multiple interests of the user. For $e > 3$ the relevance gains tapers off as users activities

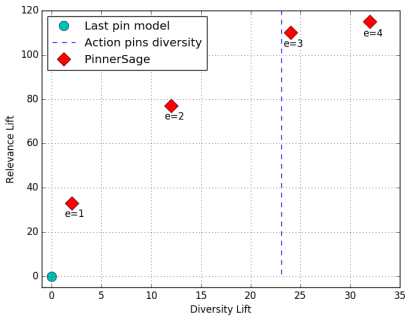

Figure 7: Diversity relevance tradeoff when different number of embeddings (e) are selected for candidate retrieval.

do not vary wildly in a given day (on average). Infact, for $e = 3$, the recommendation diversity achieved by PinnerSage matches closely with action pins diversity, which we consider as a sweet-spot.

6 RELATED WORK

There is an extensive research work focused towards learning embeddings for users and items, for e.g., [6, 7, 23, 26, 28]. Much of this work is fueled by the models proposed to learn word representations, such as Word2Vec model [18] that is a highly scalable continuous bag-of-words (CBOW) and skip-gram (SG) language models. Researchers from several domains have adopted word representation learning models for several problems such as for recommendation candidate ranking in various settings, for example for movie, music, job, pin recommendations [4, 13, 22, 28]. There are some recent papers that have also focused on candidate retrieval. [6] mentioned that the candidate retrieval can be handled by a combination of machine-learned models and human-defined rules; [23] considers large scale candidate generation from billions of users and items, and proposed a solution that pre-computes hundreds of similar items for each embedding offline. [7] has discussed a candidate generation and retrieval system in production based on a single user embedding.

Several prior works [3, 16, 25] consider modeling users with multiple embeddings. [3] uses multiple time-sensitive contextual profile to capture user's changing interests. [25] considers max function based non-linearity in factorization model, equivalently uses multiple vectors to represent a single user, and shows an improvement in 25% improvement in YouTube recommendations. [16] uses polysemous embeddings (embeddings with multiple meanings) to improve node representation, but it relies on an estimation of the occurrence probability of each embedding for inference. Both [16, 25] report results on offline evaluation datasets. Our work complements prior work and builds upon them to show to operationalize a rich multi-embedding model in a production setting.

7 CONCLUSION

In this work, we presented an end-to-end system, called PinnerSage, that powers personalized recommendation at Pinterest. In contrast to prior production systems that are based on a single embedding based user representation, PinnerSage proposes a multi-embedding based user representation scheme. Our proposed clustering scheme ensures that we get full insight into the needs of a user and understand them better. To make this happen, we adopt several design choices that allows our system to run efficiently and effectively, such as medoid based cluster representation and importance sampling of medoids. Our offline experiments show that our approach leads to significant relevance gains for the *retrieval task*, as well as delivers improvement in *reciprocal rank* for the *ranking task*. Our large A/B tests show that PinnerSage provides significant real world online gains. Much of the improvements delivered by our model can be attributed to its better understanding of user interests and its quick response to their needs. There are several promising areas that we consider for future work, such as selection of multiple medoids per clusters and a more systematic reward based framework to incorporate implicit feedback in estimating cluster importance.

8 ACKNOWLEDGEMENTS

We would like to extend our appreciation to Homefeed and Shopping teams for helping in setting up online A/B experiments. Our special thanks to the embedding infrastructure team for powering embedding nearest neighbor search.

REFERENCES

[1] V. Athitsos, M. Potamias, P. Papapetrou, and G. Kollios. Nearest neighbor retrieval using distance-based hashing. In *ICDE*, 2008.
[2] A. Babenko and V. Lempitsky. Efficient indexing of billion-scale datasets of deep descriptors. In *CVPR*, 2016.
[3] L. Baltrunas and X. Amatriain. Towards time-dependant recommendation based on implicit feedback. In *Workshop on context-aware recommender systems*, 2009.
[4] O. Barkan and N. Koenigstein. ITEM2VEC: neural item embedding for collaborative filtering. In *Workshop on Machine Learning for Signal Processing*, 2016.
[5] J. G. Carbonell and J. Goldstein. The use of mmr, diversity-based reranking for reordering documents and producing summaries. In *SIGIR*, 1998.
[6] H. Cheng, L. Koc, J. Harmsen, T. Shaked, T. Chandra, H. Aradhye, G. Anderson, G. Corrado, W. Chai, M. Ispir, R. Anil, Z. Haque, L. Hong, V. Jain, X. Liu, and H. Shah. Wide & deep learning for recommender systems. In *DLRS@RecSys*, 2016.
[7] P. Covington, J. Adams, and E. Sargin. Deep neural networks for youtube recommendations. In *RecSys*, pages 191–198, 2016.
[8] D. Defays. An efficient algorithm for a complete link method. *The Computer Journal*, 20(4):364–366, 01 1977.
[9] C. Eksombatchai, P. Jindal, J. Z. Liu, Y. Liu, R. Sharma, C. Sugnet, M. Ulrich, and J. Leskovec. Pixie: A system for recommending 3+ billion items to 200+ million users in real-time. In *WWW*, 2018.
[10] A. Epasto and B. Perozzi. Is a single embedding enough? learning node representations that capture multiple social contexts. In *WWW*, 2019.
[11] M. Grbovic and H. Cheng. Real-time personalization using embeddings for search ranking at airbnb. In *KDD*, 2018.
[12] J. Johnson, M. Douze, and H. Jégou. Billion-scale similarity search with gpus. *CoRR*, abs/1702.08734, 2017.
[13] K. Kenthapadi, B. Le, and G. Venkataraman. Personalized job recommendation system at linkedin: Practical challenges and lessons learned. In *RecSys*, 2017.
[14] G. N. Lance and W. T. Williams. A general theory of classificatory sorting strategies 1. Hierarchical systems. *Computer Journal*, 9(4):373–380, Feb. 1967.
[15] G. Linden, B. Smith, and J. York. Amazon.com recommendations: Item-to-item collaborative filtering. *IEEE Internet computing*, 7(1):76–80, 2003.
[16] N. Liu, Q. Tan, Y. Li, H. Yang, J. Zhou, and X. Hu. Is a single vector enough?: Exploring node polysemy for network embedding. In *KDD*, pages 932–940, 2019.
[17] Y. A. Malkov and D. A. Yashunin. Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs. *PAMI*, 2018.
[18] T. Mikolov, I. Sutskever, K. Chen, G. Corrado, and J. Dean. Distributed representations of words and phrases and their compositionality. In *NIPS*, 2013.
[19] S. Okura, Y. Tagami, S. Ono, and A. Tajima. Embedding-based news recommendation for millions of users. In *KDD*, 2017.
[20] B. Sarwar, G. Karypis, J. Konstan, and J. Riedl. Item-based collaborative filtering recommendation algorithms. In *WWW*, page 285âĂŞ295, 2001.
[21] K. Terasawa and Y. Tanaka. Spherical lsh for approximate nearest neighbor search on unit hypersphere. In *Workshop on Algorithms and Data Structures*, 2007.
[22] D. Wang, S. Deng, X. Zhang, and G. Xu. Learning to embed music and metadata for context-aware music recommendation. *World Wide Web*, 21(5):1399–1423, 2018.
[23] J. Wang, P. Huang, H. Zhao, Z. Zhang, B. Zhao, and D. L. Lee. Billion-scale commodity embedding for e-commerce recommendation in alibaba. In *KDD*, 2018.
[24] J. J. H. Ward. Hierarchical grouping to optimize an objective function. 1963.
[25] J. Weston, R. J. Weiss, and H. Yee. Nonlinear latent factorization by embedding multiple user interests. In *RecSys*, pages 65–68, 2013.
[26] L. Y. Wu, A. Fisch, S. Chopra, K. Adams, A. Bordes, and J. Weston. Starspace: Embed all the things! In *AAAI*, 2018.
[27] R. Ying, R. He, K. Chen, P. Eksombatchai, W. L. Hamilton, and J. Leskovec. Graph convolutional neural networks for web-scale recommender systems. In *KDD*, 2018.
[28] J. You, Y. Wang, A. Pal, P. Eksombatchai, C. Rosenberg, and J. Leskovec. Hierarchical temporal convolutional networks for dynamic recommender systems. In *WWW*, 2019.
[29] N. Zhang, S. Deng, Z. Sun, X. Chen, W. Zhang, and H. Chen. Attention-based capsule networks with dynamic routing for relation extraction. *arXiv preprint arXiv:1812.11321*, 2018.
[30] X. Zhao, R. Louca, D. Hu, and L. Hong. Learning item-interaction embeddings for user recommendations. 2018.

REPRODUCIBILITY SUPPLEMENTARY MATERIALS

APPENDIX A: Convergence proof of Ward clustering algorithm

Lemma 8.1. *In Algo. 1, a merged cluster $C_i \cup C_j$ cannot have distance lower to another cluster C_k than the lowest distance of its children clusters to that cluster, i.e., $d(C_i \cup C_j, C_k) \geq \min\{d_{ik}, d_{jk}\}$.*

Proof. For clusters i and j to merge, the following two conditions must be met: $d_{ij} \leq d_{ik}$ and $d_{ij} \leq d_{jk}$. Without loss of generality, let $d_{ij} = d$ and $d_{ik} = d + \gamma$ and $d_{jk} = d + \gamma + \delta$, where $\gamma \geq 0, \delta \geq 0$. We can simplify eq. 1 as follows:

$$d(C_i \cup C_j, C_k) = \frac{(n_i + n_k)(d + \gamma) + (n_j + n_k)(d + \gamma + \delta) - n_k\, d}{n_i + n_j + n_k}$$
$$= d + \gamma + \frac{n_k\, \gamma + (n_j + n_k)\, \delta}{n_i + n_j + n_k} \geq d + \gamma \tag{4}$$

$d(C_i \cup C_j, C_k) \geq d + \gamma$ implies $d(C_i \cup C_j, C_k) \geq \min\{d_{ik}, d_{jk}\}$. □

Lemma 8.2. *A cluster cannot be added twice to the stack in Ward clustering (algo. 1).*

Proof by contradiction. Let the state of stack at a particular time be $[\ldots, i, j, k, i]$. Since j is added after i in stack, this implies that $d_{ij} \leq d_{ik}$ (condition 1). Similarly from subsequent additions to the stack, we get $d_{jk} \leq d_{ji}$ (condition 2) and $d_{ki} \leq d_{kj}$ (condition 3). We also note that by symmetry $d_{xy} = d_{yx}$. Combining condition 2 and 3 leads to $d_{ik} \leq d_{ij}$, which would contradict condition 1 unless $d_{ji} = d_{jk}$. Since i is the second element in the stack after addition of j, j cannot add k given $d_{ji} = d_{jk}$. Hence i cannot be added twice to the stack.

Additionally, a merger of clusters since the first addition of i to the stack, cannot add i again. This is because its distance to i is greater than or equal to the smallest distance of its child clusters to i by Lemma 8.1. Since the child cluster closest to i cannot add i, so can't the merged cluster. □