# kNN-Embed: Locally Smoothed Embedding Mixtures For Multi-interest Candidate Retrieval

AHMED EL-KISHKY, Twitter Cortex, USA
THOMAS MARKOVICH, Twitter Cortex, USA
KENNY LEUNG, Twitter Cortex, USA
FRANK PORTMAN, Twitter Cortex, USA
ARIA HAGHIGHI†, Twitter Cortex, USA
YING XIAO†, Twitter Cortex, USA

Candidate generation is the first stage in recommendation systems, where a light-weight system is used to retrieve potentially relevant items for an input user. These candidate items are then ranked and pruned in later stages of recommender systems using a more complex ranking model. Since candidate generation is the top of the recommendation funnel, it is important to retrieve a high-recall candidate set to feed into downstream ranking models. A common approach for candidate generation is to leverage approximate nearest neighbor (ANN) search from a single dense query embedding; however, this approach this can yield a low-diversity result set with many near duplicates. As users often have multiple interests, candidate retrieval should ideally return a diverse set of candidates reflective of the user's multiple interests. To this end, we introduce *k*NN-Embed, a general approach to improving diversity in dense ANN-based retrieval. *k*NN-Embed represents each user as a smoothed mixture over learned item clusters that represent distinct 'interests' of the user. By querying each of a user's mixture component in proportion to their mixture weights, we retrieve a high-diversity set of candidates reflecting elements from each of a user's interests. We experimentally compare *k*NN-Embed to standard ANN candidate retrieval, and show significant improvements in overall recall and improved diversity across three datasets. Accompanying this work, we open source a large Twitter follow-graph dataset, to spur further research in graph-mining and representation learning for recommender systems.

## 1 INTRODUCTION

Recommendation systems for online services such as search engines or social networks present users with content and suggestions in the form of ranked lists of items [2, 7, 26]. Often, these item lists are constructed through a two-step process: (1) candidate generation, which efficiently retrieves a manageable subset of potentially relevant items, and (2) ranking, which applies a computationally-expensive ranking model to score, sort, and select the top-$k$ candidates to display to the user.

During candidate generation, we are primarily concerned with the *recall* of the system [17], as opposed to the ranking model which typically targets *precision*. Ensuring high recall for users with multiple interests is a challenging problem, which is exacerbated by the way we typically perform retrieval. The dominant paradigm for candidate generation

Authors' addresses: Ahmed El-Kishky, aelkishky@twitter.com, Twitter Cortex, USA; Thomas Markovich, tmarkovich@twitter.com, Twitter Cortex, USA; Kenny Leung, kennyleung@twitter.com, Twitter Cortex, USA; Frank Portman, fportman@twitter.com, Twitter Cortex, USA; Aria Haghighi†, ahaghighi@twitter.com, Twitter Cortex, USA; Ying Xiao†, yxiao@twitter.com, Twitter Cortex, USA.


Manuscript submitted to ACM

is to embed users and items in the same vector space, and then use approximate nearest-neighbor (ANN) search to retrieve candidates close to the user [7, 20]. However, ANN search will often return candidate pools that are highly intra-similar (e.g., all candidates pertain to one 'topic' only) [37]. A side effect of training embeddings to place users close to relevant items, is that similar items are also placed close to each other. During ANN-based candidate retrieval, this unfortunately leads to similar candidates that may not reflect a user's diverse multi-topic interests, and hence low recall; we demonstrate this in Section 7.

In this paper, we introduce *k*NN-Embed, a new strategy for retrieving a high-recall, diverse set of candidates reflecting a user's multiple interests. *k*NN-Embed captures multiple user interests by representing user preferences with a smoothed, mixture distribution. Our technique provides a turn-key way to increase recall and diversity while maintaining user relevance in any ANN-based candidate generation scheme. It does not require retraining the underlying user and item embeddings; instead, we build directly on top of pre-existing ANN systems. The underlying idea is to exploit the similarity of neighboring users to represent per-user interests as a mixture over learned high level clusters of item embeddings. Since user-item relevance signal is typically sparse, estimating the mixture weights introduces significance variance. Thus, we smooth the mixture weights with information from similar users. At retrieval time, we simply sample candidates from each cluster according to mixture weights. Within each cluster, we perform ANN search using a smoothed per-user per-cluster embedding.

Our contributions in this paper are (1) a principled method to retrieve a high-recall, diverse candidate set in ANN-based candidate generation systems and (2) a large-scale open-source dataset for studying graph-mining, recommendation, and retrieval over real-world graphs.

After related work, in Section 2, and preliminaries, in Section 3, we describe a simple embedding framework that learns unimodal embeddings by co-embedding users and items in the same space in Section 4. In Section 5 we outline *k*NN-Embed, our proposed approach to transform unimodal embeddings into smoothed mixtures of embeddings. In Section 6 we introduce Twitter-Follow, a large graph dataset where users follow Twitter content producers (items) on Twitter that we curate and open-source to the community. In Section 7, we thoroughly evaluate the proposed solution along multiple axes, including recall, diversity, and goodness-of-fit; additionally, we perform a careful hyper-parameter study. Finally, we conclude and discuss future work in Section 8.

## 2 RELATED WORKS

Traditionally, techniques for candidate generation have focused on large sparse vectors, and have relied on fast, scalable approaches to search for similar sparse vectors from large target collections [1, 4]. These approaches often apply innovative indexing and optimization strategies to scale similarity search.

One family of such approaches are cluster-based retrieval. These strategies generally build a static clustering of the entire collection of items; clusters are retrieved based on how well their centroids match the query [35]. These approaches have been applied to document and image retrieval [6, 27]. On a high level, these methods either (1) match the query against clusters of items and rank clusters based on similarity to query or (2) utilize clusters as a form of item smoothing. Both of these approaches differ from kNN-Embed which represents the query as a mixture of clusters and selects candidate items from each cluster - ensuring a diverse retrieved set.

With the proliferation of embedding-based methods for recommender systems such as content-based recommendation [8] and collaborative-filtering recommender systems [40], there have been approaches developed for performing similarity search in large collections of dense targets. Initial approaches for candidate generation in dense-spaces have proposed hashing-based techniques such as mapping input and targets to discrete partitions and selecting targets from

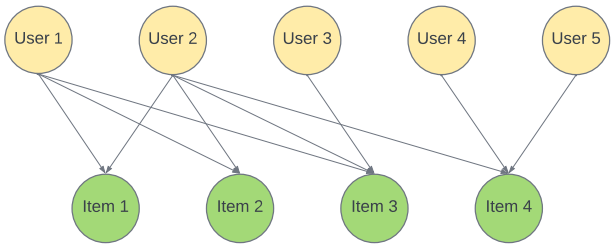



Fig. 1. Bipartite user-item engagement graph.

the same partitions as inputs [36]. With the advent of fast approximate nearest-neighbor search [19, 29, 33], dense nearest neighbor has been applied by recommender systems for candidate generation [7].

The application of scalable graph-based embedding methods has significantly improved recommender systems [12, 38]. With this improvement, there have been methods that have attempted to extend the embeddings used from single-mode to multiple mode representations [32]. These, and other [13], methods apply clustering over user actions to represent users with multiple embeddings. Our method extends upon this idea by incorporating nearest neighbor smoothing such that engagements of neighboring users can enhance the overall user representation. We believe this is crucial to address the sparsity problem of generating mixtures of embeddings for users with few engagements.

Smoothing via k-nearest-neighbor search has been applied for better language modeling [22] and machine translation [21]. Our approaches adopts a similar insight to smooth our mixture embeddings by leveraging engagements from similar users. We posit this is especially helpful for better representations for users with sparse engagement.

## 3 PRELIMINARIES

Let $\mathcal{U} = \{u_1, u_2, \dots u_n\}$ be the set of source entities (i.e., users in a recommender system) and $\mathcal{I} = \{i_1, i_2, \dots i_m\}$ be the set of target entities (i.e., items in a recommender system). Let $\mathcal{G}$ constitute a bipartite graph representing the engagements between users ($\mathcal{U}$) and items ($\mathcal{I}$) as shown in Figure 1. For each user and item, we define a 'relevance' variable in $\{0, 1\}$ indicating an item's relevance to a particular user. An item is considered relevant to a particular user if a user, presented with an item, will engage with said item.

Based on the engagements within $\mathcal{G}$, for each user, $u_j$, is associated with a $d$-dimensional embedding vector $\mathbf{u_j} \in \mathbb{R}^d$; similarly each target item $i_k$ is associated with an embedding vector $\mathbf{i_k} \in \mathbb{R}^d$. We will call these the *unimodal* embeddings, and assume that they model source-target relevance $p(\text{relevance}|u_j, i_k) = f(\mathbf{u_j}, \mathbf{i_k})$ for some suitable function $f$.

Given the input user-item engagement graph, our goal is to learn mixtures of embeddings representations of users. These mixture embeddings should better capture the multiple interests of a user as evidenced by higher recall in a candidate retrieval task.

## 4 LEARNING UNIMODAL USER AND ITEM REPRESENTATIONS

We apply a simple approach to co-embed users and items into the same space to learn our initial *unimodal* embeddings for each. To perform this embedding, we construct a bipartite graph $\mathcal{G}$ consisting of users and items, where an edge constitutes a form of engagement (e.g., citing paper cites cited paper or user follows content producer). We seek to learn an embedding vector (i.e., vector of learnable parameters) for each user ($u_j$) and item ($i_k$) in this bipartite graph;

we denote these learnable embeddings for users and items as $\mathbf{u_j}$ and $\mathbf{i_k}$ respectively. A user-item pair is scored with a scoring function of the form $f(\mathbf{u_j}, \mathbf{i_k})$. Our training objective seeks to learn $\mathbf{u}$ and $\mathbf{i}$ parameters that maximize a log-likelihood constructed from the scoring function for $(u, i) \in \mathcal{G}$ and minimize for $(u, i) \notin \mathcal{G}$.

For model simplicity, we apply a simple dot product comparator between user and item representations. For a user-item pair $e = (u_j, i)$, this operation is defined by:

$$f(e) = f(u_j, i_k) = \mathbf{u_j}^\top \mathbf{i_k} \tag{1}$$

As seen in Equation 1, we co-embed users and items by scoring their respective embedded representations via dot product. The task is then formulated as an edge (or link) prediction task. We consume the input bipartite graph $\mathcal{G}$ as a set of user-item pairs of the form $(u, i)$ which represent positive engagements between a user and item. The training objective of the embedding model is to find user and item representations that are useful for predicting which users and items are linked via an engagement. While a softmax is a natural formulation to predict a user-item engagement, it is impractical because of the prohibitive cost of computing the normalization over a large vocabulary of items. As such, following previous methods [15, 30], negative sampling, a simplification of noise-contrastive estimation, is used to learn the parameters $\mathbf{u}$ and $\mathbf{i}$. We maximize the following negative sampling objective:

$$\arg\max_{\mathbf{u},\mathbf{i}} \sum_{e \in \mathcal{G}} \left[ \log \sigma(f(e)) + \sum_{e' \in N(e)} \log \sigma(-f(e')) \right] \tag{2}$$

where: $N(u, i) = \{(u, i') : i' \in \mathcal{I}\} \cup \{(u', i) : u' \in \mathcal{U}\}$. Equation 2 represents the log-likelihood of predicting a binary "real" or "fake" label for the set of edges in the network (real) along with a set of the "fake" negatively sampled edges. To maximize the objective, we learn $\mathbf{u}$ and $\mathbf{i}$ parameters to differentiate positive edges from negative, unobserved edges. Negative edges are sampled by corrupting positive edges via replacing either the user or item in an edge pair with a negatively sampled user or item. As user-item interaction graphs are very sparse, randomly corrupting an edge in the graph is very likely to be a 'negative' edge absent from the graph. Following previous approaches, negative sampling is performed both uniformly and proportional to node prevalence in the training graph [5, 24]. Optimization is performed via Adagrad [10].

## 5 SMOOTHED MIXTURE OF EMBEDDINGS

In Section 4, we described a simple approach to co-embedding users and items to learn unimodal embeddings. In this section, we describe how *k*NN-Embed can be applied to transform unimodal embeddings into smoothed mixtures of embeddings. We then demonstrate an approach to candidate retrieval using these smoothed mixtures that leads to diverse, higher-recall candidate items.

### 5.1 Proposed Method: *k*NN-Embed

To use embeddings for candidate generation, we need a method of selecting relevant items given the input user. Ideally, we would like to construct a full distribution over all items for each user $p(i_k|u_j)$ and draw samples from it. The sheer number of items makes this difficult to do efficiently, especially when candidate generation strategies are meant to be light-weight. In practice, the most common method is to greedily select the top few most relevant items using an ANN search with the unimodal user embedding as query. A significant weakness of this greedy selection is that, by its nature, ANN search will return items that are similar not only to the user embedding, but also to each other; this drastically reduces the *diversity* of the returned items. This reduction in diversity is a side-effect of the way embeddings

are trained – typically, the goal of training embeddings is to put users and relevant items close in Euclidean space; however, this also places similar users close in space, as well as similar items. We will repeatedly exploit this 'locality implies similarity' property of embeddings in this paper to resolve this diversity issue.

We summarize our approach before providing a richer description. We model the distribution of user preferences for items, $p(i|u)$, using a mixture over learned item clusters denoted by $c$. In order to generate $n$ candidates for a user $u$, we take the following steps:

- Using Equation 5, compute cluster distribution $p_{\text{smoothed}}(c|u)$.
- Sample $n$ times with replacement from $p_{\text{smoothed}}(c|u)$ to obtain cluster counts $z_c$ for each cluster.
- For each item cluster $c$, compute the cluster-specific query embedding $\mathbf{u}^c$ using Equation 7 and retrieve $z_c$ items using ANN. The union of these cluster samples is the candidate pool.

**Clustering Items** Since neighboring items are similar in the embedding space, if we apply a distance-based clustering to items, we can arrive at groupings that represent individual user preferences well. As such, we first cluster items using spherical k-means [9]. Under this scheme, each embedding vector is first normalized to unit norm, and, after each epoch, cluster centroids are similarly normalized. As such, all cluster centroids are placed on a high-dimensional sphere with radius one.

Given these item clusters, instead of immediately collapsing the distribution $p(i_k|u_j)$ to a few items as ANN search does, we can write the full distribution $p(i_k|u_j)$ as a mixture over item clusters:

$$p(i_k|u_j) = \sum_c p(c|u_j) \cdot p(i_k|u_j, c)$$

where in each cluster, we now want to learn a separate distribution over the items in the cluster $p(i_k|u_j, c)$. Thus, we are modeling each user's higher level interests $p(c|u)$, and then within each interest $c$, we can apply an efficient ANN-search strategy as before. In effect, we are interpolating between sampling the full preference distribution $p(i_k|u_j)$ and greedily selecting a few items in an ANN.

**Mixture of Embeddings via Cluster Engagements** After clustering target entities, we need to learn $p(c|u_j)$. The natural method is the maximum likelihood estimator (MLE) given by:

$$p_{\text{mle}}(c|u_j) = \frac{\text{count}(u_i, c)}{\sum_{c' \in \mathcal{M}_j} \text{count}(u_j, c')}, \tag{3}$$

where, $\text{count}(u_j, c)$ is the number of times $u_j$ has a relevant item in cluster $c$. For computational efficiency, we take $\mathcal{M}_j$ to be $u_j$'s top $m$ most relevant clusters. We normalize these counts to obtain a proper cluster-relevance distribution.

**Nearest Neighbor Smoothing** Unfortunately, we typically have little user-item engagement data on a per-user basis; thus, while the MLE is unbiased and asymptotically efficient, it can also be high variance. To this end, we introduce a smoothing technique that once again exploits locality in the ANN search, this time for users.

In Figure 2, we illustrate our scheme whereby we identify $k$ nearest-neighbors to the query user, and leverage information from the neighbors' cluster engagements to augment the user's cluster relevance. To do this, we query the nearest-neighbor index to retrieve $u_j$'s $k$-nearest user neighbors $\mathcal{K}_j$. We then compute a distribution over item clusters

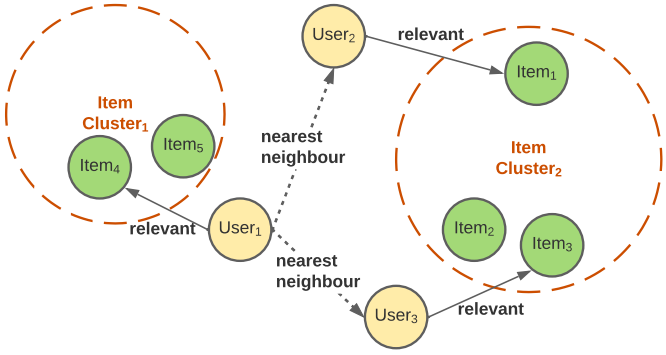



Fig. 2. Example of retrieving two candidates. In an ANN, items 4 and 5 would be deterministically returned for user 1. In our proposed $k$NN-Embed, even though the distances to cluster 2 are larger, smoothing means that we will sometimes return items from that cluster, yielding more diverse items. Note in this case, we don't even require that user 1 has previously relevant items in cluster 2.

by averaging the MLE probability for each nearest neighbor (item clusters that are not engaged with by a retrieved neighbor have zero probability).

$$p_{kNN}(c|u_j) = \frac{1}{|\mathcal{K}_j|} \sum_{u' \in \mathcal{K}_j} p_{\text{mle}}(c|u') \tag{4}$$

While Dirichlet smoothing is a natural Bayesian approach to smoothing our multinomials, the lack of a closed form estimate makes the approach impractical at scale [31, 39]. As an alternative, we apply Jelinik-Mercer smoothing to interpolate between a user's MLE distribution with the aggregated nearest neighbor distribution [16, 18, 22].

$$p_{\text{smoothed}}(c|u_j) = (1-\lambda)p_{\text{mle}}(c|u_j) + \lambda p_{kNN}(c|u_j), \tag{5}$$

where $\lambda \in [0, 1]$ represents how much smoothing is applied. It can be manually set or tuned on a downstream extrinsic task.

**Sampling within Clusters** Within each cluster there are many ways to retrieve items on a per user basis. A simple, but appealing, strategy is to represent each user as a normalized centroid of their relevant items in that cluster:

$$\text{centroid}(c, u_j) = \frac{\sum_{m \in R(c,u_j)} \mathbf{i_m}}{\| \sum_{m \in R(c,u_j)} \mathbf{i_m} \|}, \tag{6}$$

where $R(c, u_j)$ is the set of relevant items for user $u_j$ in cluster $c$. However, since we are applying smoothing to the cluster probabilities $p(c|u_j)$, it may be case that $u_j$ has zero relevant items in a given cluster. Hence, we smooth the user centroid by using information from the neighbors to obtain the final user representation $\mathbf{u_j^c}$:

$$\mathbf{u_j^c} = \frac{(1-\lambda)\,\text{centroid}(c, u_j) + \frac{\lambda}{|\mathcal{K}_j|} \sum_{u' \in \mathcal{K}_j} p_{\text{mle}}(c|u')\,\text{centroid}(c, u')}{\|(1-\lambda)\,\text{centroid}(c, u_j) + \frac{\lambda}{|\mathcal{K}_j|} \sum_{u' \in \mathcal{K}_j} p_{\text{mle}}(c|u')\,\text{centroid}(c, u')\|} \tag{7}$$

Equation 7 shows the $kNN$-smoothed user-specific embedding for cluster $c$. This embedding takes the user-specific cluster representations from Equation 6, and performs a weighted averaging proportionate to each user's contribution to $p_{\text{smoothed}}(c|u_j)$. The final centroid vector is once again normalized to unit norm.

## 6 EVALUATION DATASETS AND METRICS

In this section we describe the datasets we evaluate on, including *Twitter-Follow*, a Twitter Follow graph that we curated, and are open-sourcing alongside our work. Additionally, we introduce three metrics that we use to evaluate the quality of embedding-based candidate retrieval methodologies.

### 6.1 Datasets

We evaluate *k*NN-Embed on three datasets with their associated candidate retrieval tasks, which we describe below:

**HEP-TH Citation Graph:** Our first dataset we evaluate on is *arXiv HEP-TH* – a high energy physics theory academic paper citation graph [14]. This paper citation network is collected from Arxiv preprints from the High Energy Physics category. The dataset consists of: 34,546 papers and 421,578 citations. The task is to retrieve candidate papers that a user may cite for a given source paper.

**DBLP Citation Graph:** The second dataset we evaluate on is the *DBLP citation network* – a paper citation network extracted from DBLP [34]. The dataset consists of 5,354,309 papers and 48,227,950 citation relationships. Once again, the task is to retrieve candidate papers that a user may cite for a given source paper.

**Twitter Follow Graph:** As an accompanying resource to this paper, we created this dataset by querying Twitter follows for each user (available via API) and heavily sub-sampling this user *follows* user graph. Each edge is directed, and the presence of symmetric edges indicates that both users follow one another. We construct this graph by first selecting a number of 'highly-followed' users that we refer to as 'content producers'; these content producers serve as 'items' in our recommender systems terminology. We then sampled users that follow these content producer accounts. All users are anonymized with no other personally identifiable information (e.g., demographic features) present. Additionally, the timestamp of each follow edge was mapped to an integer that respects date ordering, but does not provide any information about the date that follow occurred. In total, we have $261M$ edges and $15.5M$ vertices, with a max-degree of $900K$ and a min-degree of 5. We hope that this dataset will be of useful to the community as a test-bed for large-scale retrieval and embedding research.

### 6.2 Metrics

We evaluate *k*NN-Embed on three aspects: (1) the recall of retrieved candidates as measured on a held-out set of items (2) the diversity of retrieved candidates and (3) the goodness of fit, namely how well a multi-interest mixture of embeddings models the user's interests as evidenced through a held-out set of items. Below, we formalize these metrics.

**Recall@K:** The most natural (and perhaps most important) metric for computing the efficacy of various candidate retrieval strategies is *Recall@K*. This metric is given by considering a fixed number of top candidates yield by a retrieval system (up to size $K$) and measuring what percent of these candidates are held-out relevant candidates. The purpose of most candidate retrieval systems is to collect a high-recall pool of items for further ranking, and thus recall is a relevant metric to consider. Additionally, recall provides an indirect way to measure diversity – to achieve high recall, one is obliged to return a large fraction of *all* relevant documents, which simple greedy ANN searches can struggle with.

**Diversity:** To evaluate the diversity among the retrieved candidates, we measure the spread in the embeddings of the retrieved candidates by calculating the average distance retrieved candidates are from their centroid. The underlying idea is that in for embeddings, 'locality implies similarity'; as a corollary, if candidates are *further* in Euclidean distance, then they are likely to be different. As such, for a given set of candidates $C$, we compute diversity $D$ as follows:

$$D(C) = \frac{1}{|C|} \sum_{i_k \in C} \|\mathbf{i_k} - \hat{\mathbf{i}}\| \tag{8}$$

where $C$ denotes the set of retrieved candidates and $\hat{\mathbf{i}} = \sum_{i_k \in C} \mathbf{i_k}/|C|$ is the mean of the unimodal embeddings of the retrieved candidates. We could also have used the square norm here, which gives largely similar results. Since our embeddings are normalized, square norms tend to be very small and quite difficult to interpret.

**Goodness of Fit:** In addition to diversity of retrieved items, we need to ensure that a user's mixture representation is an accurate model of their interests – that is the mixture of embeddings identifies points in the embedding space where relevant items lie. Thus, we compare held out relevant items to the user's mixture representation we use to query. We measure this "goodness of fit" by computing the Earth Mover's Distance (EMD) [25] between a uniform distribution over a user's relevant items and the user's mixture distribution.

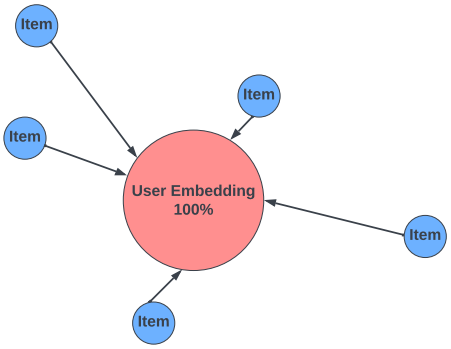



Fig. 3. Goodness of fit of unimodal embedding.

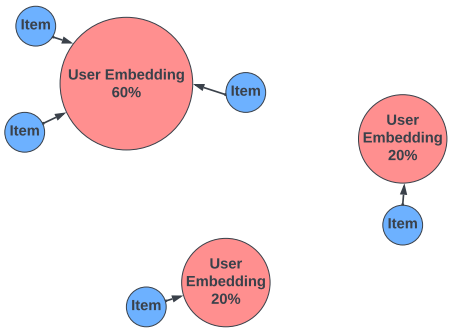



Fig. 4. Goodness of fit of mixture of embeddings.

The EMD is a metric that measures the distance between two probability distributions over a metric space. In our case, we measure the distance between a user's cluster distribution (e.g., Equation 3 and Equation 4), to a uniform distribution over a held-out set of relevant items: $p(i|u_j)$. Following previous works, distance is measured over the Euclidean embedding space [11, 23]. In our case, the EMD is computed by soft assigning all held-out relevant items of a user to clusters, minimizing the sum of item-cluster distances, with the constraint that the sum over soft assignments matches $p(c|u_j)$. As seen in Figure 3, with standard unimodal representations, a single embedding vector is compared to the held-out items and the goodness of fit is the distance between the item embeddings and the singular user embedding. In comparison, Figure 4, shows our mixture embeddings, under this model, we model each user with multiple user embeddings who each have fractional probability mass that in total sums to 1. The goodness of fit is then the distance achieved by allocating the mass in each item to the closest user embedding cluster with available probability mass. Observing Fig. 3, a single unimodal embedding is situated in the embedding space and compared to held-out relevant

items. As shown, some held-out items are close to the unimodal embedding, while others are further away. In contrast, Fig. 4, multiple user-embeddings represent each user where each embedding lies close to a cluster of relevant items. The intuition is that if a user has multiple item clusters they are interested in, multiple embeddings can better capture these interests.

We compute this goodness of fit EMD between held-out items and a user representation by solving the linear program:

$$\text{EMD}(p(c|u_j), p(i|u_j)) = \min_{T \geq 0} \sum_{c \in \mathcal{M}_j} \sum_{i_k \in \mathcal{R}_j} T_{c,k} \cdot ||\mathbf{u_j^c} - \mathbf{i_k}||$$

subject to:

$$\forall c \sum_{i_k \in \mathcal{R}_j} T_{c,k} = p(c|u_j)$$

$$\forall k \sum_{c \in \mathcal{M}_j} T_{c,k} = p(i_k|u_j)$$

Where $T \in R^{|C| \times |\mathcal{I}|}$ is a non-negative matrix, where each $T_{c,k}$ denotes how much of cluster $c$ for user $j$ is assigned to item $k$ for user $j$, and constraints ensure the flow of the probability mass for an item or cluster cannot exceed its allocated mass. Specifically, this formulation ensures the the entire outgoing flow from cluster $c$ equals $p(c|u_j)$, i.e. $\sum_k T_{c,k} = p(c|u_j)$. Additionally, the amount of incoming flow to item $k$ must match $p(i_k|u_j)$, i.e., $\sum_c T_{c,k} = p(i_k|u_j)$.

## 7 EXPERIMENTS

In this section, we evaluate the performance of *k*NN-Embed and compare it against baseline unimodal embeddings upon which it's built. Additionally, we perform ablation experiments to isolate the improvement due to smoothing.

### 7.1 Experimental Setup

For our underlying ANN-based candidate generation system, we start by creating a bipartite graph between source entities and target entities for each dataset, with each edge representing explicit relevance between items (e.g., citing paper *cites* cited paper or user *follows* content producer). Given each bipartite graph, we learn unimodal embeddings for users and items by follows the steps as described in Section 4. We learn 100-dimensional user and item embeddings and train our embeddings over 20 epochs. For the clustering algorithm, we apply the spherical $k$-means algorithm for 20 epochs to cluster items based on their unimodal embedding vectors [3, 28].

**Evaluation Task:** We evaluate three candidate generation strategies – baseline ANN with unimodal embeddings (which is how most ANN-based candidate generation systems work), mixture of embeddings with no smoothing (as is done in [32]), and mixture of embeddings with smoothing (as described in Section 5). For each strategy, we compute the *Recall@K*, diversity, and fit in a link prediction task i.e., predicting which additional papers to cite, or which accounts to follow, given the training set. Recall provides a natural way to measure diversity – to achieve high recall, one is obliged to return a large fraction of *all* relevant documents, which simple greedy ANN searches are unable to achieve. In addition, explicitly measuring diversity via average deviation of retrieved candidates and goodness of fit via EMD metrics shows that a mixture of embeddings is naturally a superior paradigm to model user-item engagements for the purpose of multi-interest candidate generation.

**Research Hypotheses**: We aim to explore two research hypotheses (as well as achieve some understanding of the hyperparameters):

(1) Unimodal embeddings miss many relevant items due to the similarity of retrieved items. Mixture representations can yield more diverse and higher recall candidates.
(2) Smoothing, by using information from neighboring users, further improves the recall of retrieved items.

### 7.2 Recall

Table 1. HEP-TH Citation Prediction. $\lambda = 0.8$, 2000 clusters, 5 embeddings for multi-querying.

| Approach | R@10 | R@20 | R@50 |
|---|---|---|---|
| Unimodal | 20.0% | 30.0% | 45.7% |
| Mixture | 22.7% | 33.4% | 49.3% |
| *k*NN-Embed | **25.8%** | **37.4%** | **52.5%** |

Table 2. DBLP Citation Prediction. $\lambda = 0.8$, 10000 clusters, 5 embeddings for multi-querying.

| Approach | R@10 | R@20 | R@50 |
|---|---|---|---|
| Unimodal | 9.4% | 13.9% | 21.6% |
| Mixture | 10.9% | 16.1% | 25.1% |
| *k*NN-Embed | **12.7%** | **18.8%** | **28.3%** |

Table 3. Twitter Follow Prediction. $\lambda = 0.8$, 40000 clusters, 5 embeddings for multi-querying.

| Approach | R@10 | R@20 | R@50 |
|---|---|---|---|
| Unimodal | 0.58% | 1.02% | 2.06% |
| Mixture | 3.70% | 5.53% | 8.79% |
| *k*NN-Embed | **4.13%** | **6.21%** | **9.77%** |

Experiments comparing candidate generation recall with a single embedding, vs mixture of embeddings, vs smoothed mixtures (*k*NN-Embed). Higher recall is better.

In Table 1 and Table 2, we report results when evaluating recall on citation prediction tasks for the HEP-TH and DBLP citation networks. Results support the first hypothesis that unimodal embeddings may miss relevant items if they don't lie close to the source entity in the shared embedding space. When we utilize a mixture of embeddings with no smoothing, we see a 14% relative improvement in $R$@10 for for HEP-TH, and 16% relative improvement for DBLP. Our second hypothesis (2) posits that data sparsity can lead to sub-optimal mixtures of embeddings, and that nearest-neighbor smoothing can mitigate this. Our experiments support this hypothesis, as we see a 25% relative improvement for HEP-TH in $R$@10, and 35% for DBLP and when using *k*NN-Embed. We see similar significant improvements over baselines in $R$@20 and $R$@50.

In Table 3, we report results for *k*NN-Embed and baselines for our new Twitter follow prediction task (see Section 6). In this case, the improvements in recall are dramatic – 534% in relative terms going from unimodal embeddings to a mixture of embeddings in $R$@10. We suspect this significant improvement is because Twitter-Follows simulatenously has a much higher average degree than HEP-TH and DBLP and the number of unique nodes is much larger. It is a more difficult task to embed so many items, from many different interest clusters, in close proximity to a user. As such, we see a massive improvement by explicitly querying from each user's interest clusters. Applying smoothing provides an additional 74% in relative terms, and similar behaviours are observed in $R$@20 and $R$@50.

### 7.3 Diversity

We apply Equation 8 to retrieved candidates, and measure the spread of the distances between each retrieved candidate's embedding vector to the centroid of the retrieved candidates. As seen in Tables 4, 5, and 6, we notice that candidates generated via unimodal retrieval are less diverse than candidates generated via multi-interest. This verifies our first research hypothesis that unimodal embeddings may retrieve many items that are clustered closely together as a by-product of ANN retrieval (i.e., diversity is low, and so is recall). However, querying multiple times from mixtures of

Table 4. HEP-TH diversity

| **Approach** | D@10 | D@20 | D@50 |
|---|---|---|---|
| Unimodal | 0.49 | 0.54 | 0.61 |
| Mixture | **0.58** | **0.63** | **0.68** |
| *k*NN-Embed | 0.54 | 0.60 | 0.66 |

Table 5. DBLP diversity

| **Approach** | D@10 | D@20 | D@50 |
|---|---|---|---|
| Unimodal | 0.43 | 0.46 | 0.51 |
| Mixture | **0.51** | **0.56** | **0.60** |
| *k*NN-Embed | 0.46 | 0.52 | 0.57 |

Table 6. Twitter-Follow diversity

| **Approach** | D@10 | D@20 | D@50 |
|---|---|---|---|
| Unimodal | 0.38 | 0.40 | 0.43 |
| Mixture | **0.56** | **0.54** | **0.58** |
| *k*NN-Embed | 0.47 | 0.52 | 0.55 |

Diversity of retrieved candidates as measured by spread of retrieved candidates.

embeddings broadens the search spatially. The retrieved items are from different clusters, which are more spread out from each other.

Interestingly we notice that *k*NN-Embed (i.e., smooth mixture retrieval) results in slightly less diverse candidates than unsmoothed mixture retrieval. We posit that this is due to the high-variance of the maximum likelihood estimator of the $p_{\text{mle}}(c|u_j)$ multinomial (Equation 3). While this high-variance may yield more diverse candidates, we believe variance results in less accurate candidates retrieved; this is verified by cross-referencing diversity results with recall as seen in Tables 1, 2, and 3 where *k*NN-Embed consistently yields better recall than unsmoothed mixture retrieval. While high diversity is necessary for high recall, it is insufficient on its own (e.g., returning uniformly random candidates is high diversity, and low recall), and is not unconditionally desirable in candidate generation.

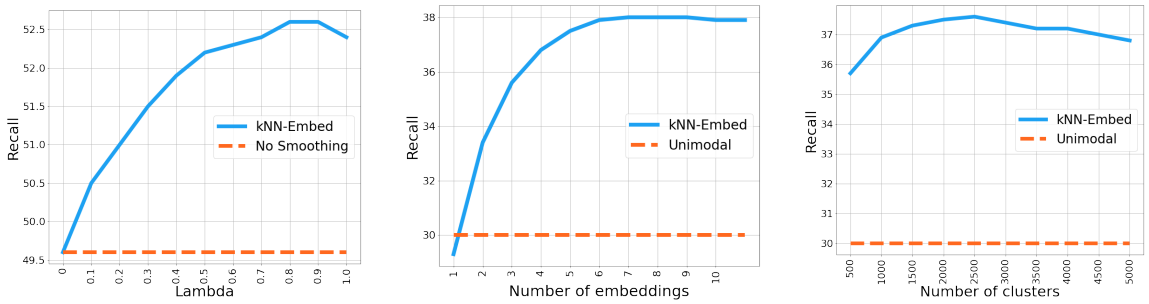
### 7.4 Goodness of Fit

In addition to measuring the diversity of retrieved candidates, we also seek to evaluate how well modeling users via unimodal, mixture, and smoothed mixture embeddings fits a user's interest. To facilitate this, we evaluate the goodness of fit between user representations used for retrieval to a set of held-out relevant items for each user. The main idea is that the better fit a user representation is, the closer it will be to the distribution of held out relevant items for that user.

Table 7. Goodness of fit between user representation and held out items as measured by earth mover's distance over the Euclidean embedding space. Lower EMD is better.

| **Approach** | HEP-TH Citation | DBLP Citation | Twitter-Follow |
|---|---|---|---|
| Unimodal | 0.897 | 0.889 | 1.018 |
| Mixture | 0.838 | 0.830 | 0.952 |
| *k*NN-Embed | **0.811** | **0.808** | **0.940** |

As seen in Table 7, the results validate the idea that unimodal user embeddings do not model user interets as well as mixtures over multiple embeddings. These additional embeddings yield a significant EMD improvement over a single embedding vector when evaluated on held-out items. Additionally, smoothing also further decreases the EMD; we posit that, once again, this is due to the smoothed embedding mixtures being lower-variance estimates as they leverage engagement data from similar users in constructing the representations. Thus, the results in this subsection hint at an underlying reason as to why smoothed mixtures may have higher recall than unsmoothed mixtures or unimodal embeddings – smoothed mixtures have simply learned the user preferences better.




(a) Varying lambda – R@50. (b) Varying mixture size – R@20. (c) Varying number of clusters – R@20.

Fig. 5. We analyze the effect of three important hyper-parameters: (1) the $\lambda$ smoothing (2) the number of embeddings in the mixture (3) the number of clusters for candidate generation in the HEP-TH dataset.

### 7.5 Hyper-parameter Ablation

To better understand how hyper-parameters affect the performance of *k*NN-Embed, we perform a study on the HEP-TH dataset. In this section, we focus on recall as the *sine qua non* of candidate retrieval problems. In Figure 5a, we vary the smoothing parameter $\lambda$ – for simplicity, we use the same parameter for smoothing the both the mixture probabilities and the cluster centroids. We see that for our HEP-TH dataset, using heavy smoothing improves performance significantly. This likely stems from the sparsity of HEP-TH where most papers have only a few citations.

Next, in Figure 5b, we vary the number of embeddings (i.e., the mixture size). We find that performance improves markedly until saturating at six mixture components. Out of all the hyperparameters, this seems to be the critical one in achieving high recall. In practice, latency constraints can be considered when selecting the number of embeddings per user, explicitly making the trade-off between diversity and latency. One interesting aspect is that *k*NN-Embed with a single embedding performs only slightly worse than the original unimodal embedding. The comparable performance supports our first hypothesis that a unimodal user embedding is often placed next to the single item cluster it has the most engagement to, leading to the lack of diversity in candidates.

Finally, in Figure 5c, we vary the number of clusters in the *k*NN clustering of items. For this particular dataset, recall peaks at $k = 2500$ and then decreases. HEP-TH is a small dataset with only 34,546 items; it is likely that generating a very large number of clusters leads to excessively fine-grained and noisy sub-divisions of the items.

## 8 CONCLUSIONS AND FUTURE WORK

In this work, we present *k*NN-Embed, a method of transforming a single user dense embedding, into a mixture of embeddings, with the goal of increasing retrieval recall and diversity. This multi-embedding scheme effectively represents a source entity with multiple distinct topical affinities by globally clustering items and aggregating the source entity's engagements with clusters. Recognizing that user-item engagements may often be sparse, we propose a novel nearest-neighbor smoothing to enrich an entities mixture representation from similar entities. Our proposed smoothed mixture representation better models user preferences retrieving a diverse set of candidate items reflective of a user's multiple interests. This significantly improves recall on a candidate generation retrieval task on three datasets: DBLP citation prediction, ArXiv HEP-TH citation prediction, and Twitter user account suggestion. In conjunction with this work, we open-source our curated Twitter Follow Graph dataset as a resource to the information retrieval community.

There are many follow-up areas of investigation left to future work. For example, in our investigations we utilized a global smoothing parameter; however a per-user smoothing parameter may be more appropriate. Variable smoothing per user can provide more smoothing when a user's engagements are sparse, and back off to less smoothing when engagements are dense. Further work can investigate utilizing mixtures of embeddings for additional recommender systems tasks like in supervised ranking models. We posit that mixture of embeddings representation of users, mixture of embeddings of items can be effectively utilized by modern attention-based transformer architectures for superior ranking.

## REFERENCES


[1] Alexandr Andoni and Piotr Indyk. 2006. Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. In *2006 47th annual IEEE symposium on foundations of computer science (FOCS'06)*. IEEE, 459–468.

[2] Vito Walter Anelli, Saikishore Kalloori, Bruce Ferwerda, Luca Belli, Alykhan Tejani, Frank Portman, Alexandre Lung-Yut-Fong, Ben Chamberlain, Yuanpu Xie, Jonathan Hunt, et al. 2021. RecSys 2021 Challenge Workshop: Fairness-aware engagement prediction at scale on Twitter's Home Timeline. In *Fifteenth ACM Conference on Recommender Systems*. 819–824.

[3] David Arthur and Sergei Vassilvitskii. 2006. *k-means++: The advantages of careful seeding*. Technical Report. Stanford.

[4] Roberto J Bayardo, Yiming Ma, and Ramakrishnan Srikant. 2007. Scaling up all pairs similarity search. In *Proceedings of the 16th international conference on World Wide Web*. 131–140.

[5] Antoine Bordes, Nicolas Usunier, Alberto Garcia-Duran, Jason Weston, and Oksana Yakhnenko. 2013. Translating embeddings for modeling multi-relational data. *Advances in neural information processing systems* 26 (2013).

[6] Yixin Chen, James Ze Wang, and Robert Krovetz. 2005. Clue: Cluster-based retrieval of images by unsupervised learning. *IEEE transactions on Image Processing* 14, 8 (2005), 1187–1201.

[7] Paul Covington, Jay Adams, and Emre Sargin. 2016. Deep neural networks for youtube recommendations. In *Proceedings of the 10th ACM conference on recommender systems*. 191–198.

[8] Marco De Gemmis, Pasquale Lops, Cataldo Musto, Fedelucio Narducci, and Giovanni Semeraro. 2015. Semantics-aware content-based recommender systems. In *Recommender systems handbook*. Springer, 119–159.

[9] Inderjit S Dhillon and Dharmendra S Modha. 2001. Concept decompositions for large sparse text data using clustering. *Machine learning* 42, 1 (2001), 143–175.

[10] J. Duchi, E. Hazan, and Y. Singer. 2011. Adaptive subgradient methods for online learning and stochastic optimization. *JMLR* 12, 7 (2011).

[11] Ahmed El-Kishky and Francisco Guzmán. 2020. Massively Multilingual Document Alignment with Cross-lingual Sentence-Mover's Distance. *arXiv preprint arXiv:2002.00761* (2020).

[12] Ahmed El-Kishky, Thomas Markovich, Serim Park, Chetan Verma, Baekjin Kim, Ramy Eskander, Yury Malkov, Frank Portman, Sofia Samaniego, Ying Xiao, et al. 2022. TwHIN: Embedding the Twitter Heterogeneous Information Network for Personalized Recommendation. *arXiv preprint arXiv:2202.05387* (2022).

[13] Weihao Gao, Xiangjun Fan, Jiankai Sun, Kai Jia, Wenzhi Xiao, Chong Wang, and Xiaobing Liu. 2020. Deep retrieval: An end-to-end learnable structure model for large-scale recommendations. *arXiv preprint arXiv:2007.07203* (2020).

[14] Johannes Gehrke, Paul Ginsparg, and Jon Kleinberg. 2003. Overview of the 2003 KDD Cup. *Acm Sigkdd Explorations Newsletter* 5, 2 (2003), 149–151.

[15] Y. Goldberg and O. Levy. 2014. word2vec Explained: deriving Mikolov et al.'s negative-sampling word-embedding method. *arXiv preprint arXiv:1402.3722* (2014).

[16] Edouard Grave, Moustapha Cissé, and Armand Joulin. 2017. Unbounded cache model for online language modeling with open vocabulary. *arXiv preprint arXiv:1711.02604* (2017).

[17] Jui-Ting Huang, Ashish Sharma, Shuying Sun, Li Xia, David Zhang, Philip Pronin, Janani Padmanabhan, Giuseppe Ottaviano, and Linjun Yang. 2020. Embedding-based retrieval in facebook search. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 2553–2561.

[18] Frederick Jelinek. 1980. Interpolated estimation of Markov source parameters from sparse data. In *Proc. Workshop on Pattern Recognition in Practice, 1980*.

[19] Jeff Johnson, Matthijs Douze, and Hervé Jégou. 2019. Billion-scale similarity search with gpus. *IEEE Transactions on Big Data* (2019).

[20] Wang-Cheng Kang and Julian McAuley. 2019. Candidate generation with binary codes for large-scale top-n recommendation. In *Proceedings of the 28th ACM international conference on information and knowledge management*. 1523–1532.

[21] Urvashi Khandelwal, Angela Fan, Dan Jurafsky, Luke Zettlemoyer, and Mike Lewis. 2020. Nearest neighbor machine translation. *arXiv preprint arXiv:2010.00710* (2020).

[22] Urvashi Khandelwal, Omer Levy, Dan Jurafsky, Luke Zettlemoyer, and Mike Lewis. 2019. Generalization through memorization: Nearest neighbor language models. *arXiv preprint arXiv:1911.00172* (2019).

[23] Matt Kusner, Yu Sun, Nicholas Kolkin, and Kilian Weinberger. 2015. From word embeddings to document distances. In *International conference on machine learning*. 957–966.

[24] Adam Lerer, Ledell Wu, Jiajun Shen, Timothee Lacroix, Luca Wehrstedt, Abhijit Bose, and Alex Peysakhovich. 2019. Pytorch-biggraph: A large-scale graph embedding system. *arXiv preprint arXiv:1903.12287* (2019).

[25] Elizaveta Levina and Peter Bickel. 2001. The earth mover's distance is the mallows distance: Some insights from statistics. In *Proceedings Eighth IEEE International Conference on Computer Vision. ICCV 2001*, Vol. 2. IEEE, 251–256.

[26] David C Liu, Stephanie Rogers, Raymond Shiau, Dmitry Kislyuk, Kevin C Ma, Zhigang Zhong, Jenny Liu, and Yushi Jing. 2017. Related pins at pinterest: The evolution of a real-world recommender system. In *Proceedings of the 26th international conference on world wide web companion*. 583–592.

[27] Xiaoyong Liu and W Bruce Croft. 2004. Cluster-based retrieval using language models. In *Proceedings of the 27th annual international ACM SIGIR conference on Research and development in information retrieval*. 186–193.

[28] Stuart Lloyd. 1982. Least squares quantization in PCM. *IEEE transactions on information theory* 28, 2 (1982), 129–137.

[29] Yu A Malkov and Dmitry A Yashunin. 2018. Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs. *IEEE transactions on pattern analysis and machine intelligence* 42, 4 (2018), 824–836.

[30] T. Mikolov, I. Sutskever, K. Chen, G. Corrado, and J. Dean. 2013. Distributed representations of words and phrases and their compositionality. *NeurIPS* 26 (2013).

[31] Thomas Minka. 2000. Estimating a Dirichlet distribution.

[32] Aditya Pal, Chantat Eksombatchai, Yitong Zhou, Bo Zhao, Charles Rosenberg, and Jure Leskovec. 2020. PinnerSage: multi-modal user embedding framework for recommendations at pinterest. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 2311–2320.

[33] Anshumali Shrivastava and Ping Li. 2014. Asymmetric LSH (ALSH) for sublinear time maximum inner product search (MIPS). *Advances in neural information processing systems* 27 (2014).

[34] Jie Tang, Jing Zhang, Limin Yao, Juanzi Li, Li Zhang, and Zhong Su. 2008. Arnetminer: extraction and mining of academic social networks. In *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*. 990–998.

[35] Cornelis Joost Van Rijsbergen and W Bruce Croft. 1975. Document clustering: An evaluation of some experiments with the Cranfield 1400 collection. *Information Processing & Management* 11, 5-7 (1975), 171–182.

[36] Jason Weston, Ameesh Makadia, and Hector Yee. 2013. Label partitioning for sublinear ranking. In *International conference on machine learning*. PMLR, 181–189.

[37] Mark Wilhelm, Ajith Ramanathan, Alexander Bonomo, Sagar Jain, Ed H Chi, and Jennifer Gillenwater. 2018. Practical diversified recommendations on youtube with determinantal point processes. In *Proceedings of the 27th ACM International Conference on Information and Knowledge Management*. 2165–2173.

[38] Rex Ying, Ruining He, Kaifeng Chen, Pong Eksombatchai, William L Hamilton, and Jure Leskovec. 2018. Graph convolutional neural networks for web-scale recommender systems. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 974–983.

[39] Chengxiang Zhai and John Lafferty. 2017. A study of smoothing methods for language models applied to ad hoc information retrieval. In *ACM SIGIR Forum*, Vol. 51. ACM New York, NY, USA, 268–276.

[40] Fuzheng Zhang, Nicholas Jing Yuan, Defu Lian, Xing Xie, and Wei-Ying Ma. 2016. Collaborative knowledge base embedding for recommender systems. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*. 353–362.