Recommending items with attributes

Anonymous Author(s)

ABSTRACT

Recommendation models suggest items to users. We study a variant of this task where each item has a set of attributes, such as tags on an image, user reactions to a post, or foods in a meal. We propose RANKFROMSETS, a flexible and scalable model for recommending items with attributes. RANKFROMSETS treats item attributes as side information and learns embeddings to discriminate items a user will consume from items a user is unlikely to consume. We develop theory connecting the RANKFROMSETS loss to optimal recall and show that the learnable class of models for RANKFROMSETS is a superset of several previously-proposed models. To scale RANKFROMSETS to large datasets, we propose a stochastic optimization framework based on negative sampling. Within this framework, we suggest two specific approaches that trade off computation for accuracy. In experiments on two real-world datasets, RANKFROMSETS outperforms competing approaches while also learning embeddings that reveal interpretable structure in user behavior.

1 INTRODUCTION

Classical recommender system datasets contain a matrix where each row is a user and each column is an item. Each entry in the matrix indicates whether or not the user consumed the item. Modern applications often gather rich side information about items in the form of a set of attributes or tags. Intuitively, item attributes provide valuable side information for recommender systems. When the number of items is large or the user-item matrix is sparse, attribute information is crucial to achieving good performance.

Modeling item attributes in recommenders is not straightforward. Popular ways to use item attributes like multiple matrix factorization [8, 32] struggle when the attribute vocabulary or number of items is large. Simple models are computationally tractable. But oversimplifying risks losing the ability to capture nonlinear patterns of user consumption. For instance, a user may enjoy items tagged with attributes A and B, B and C, or A and C, but not all three. Finding the right balance between scalability and flexibility is therefore a primary goal.

Even when a model can be scaled, it is not always clear how the training procedure connects to the recommender system evaluation metric. A model with optimal training loss may not lead to optimal recommender system performance. For example, a matrix factorization method may minimize mean squared error but the recommender system may be evaluated on recall. While it is plausible that minimizing error will improve recall, the connection between the two is implicitly assumed in many methods. Ideally, a well-designed recommender system should have an optimization loss that matches its evaluation metric.

This paper proposes RANKFROMSETS (RFS), a scalable model for recommending items with attributes. RFS casts the recommendation problem as binary classification. Given an item and a user, RFS treats attributes as features and classifies whether or not the item is likely to be consumed by the user. RFS learns embeddings for each user and attribute; each item is represented as the mean of its attribute embeddings. To scale to large datasets, RFS is trained using a stochastic optimization procedure that randomly samples items that are unlikely to be consumed.

RFS enjoys two benefits from framing the recommendation problem as classification. First, the RFS classification objective function is directly tied to recommender recall, and we show that a classifier with zero worst-case error achieves maximum recall. Second, the model is provably flexible enough to learn any class of recommendation model based on set-valued side information. RFS generalizes other models.

We study RFS on two datasets. We consider a dataset of 65k users clicking on 637k papers posted to the arXiv; the attributes of a paper are the unique words in its abstract. RFS outperforms a state-of-the-art content-based matrix factorization model [8] on this dataset, and the learned document embeddings capture the structure of human-labeled topics. Second, we consider a large dataset of 55k users logging 16M meals using the LoseIt! diet tracking app. Again, RFS outperforms other baseline models, and the learned meal embeddings reveal a diverse set of recommendations that capture the notion of taste.

2 BACKGROUND

We highlight two themes in research on recommendation models. We describe recommendation models that incorporate side information and models that optimize proxies of ranking metrics, and summarize this related work in Table 1.

Recommendation with side information. Side information is included in recommendation models in several ways; we focus on deep learning and matrix factorization approaches. Item side information can be modeled with deep representations [1, 5, 7, 16, 18, 34, 39, 41] or can be included in content-based matrix factorization models as an additional matrix [2, 8, 20, 27, 32, 40]. Some deep learning based approaches scale to large datasets, but may not have loss functions tied to evaluation metrics, or require data besides user-item interactions. Content-based matrix factorization methods require learning parameters for every item, and do not scale to data with large numbers of items.

Learning to rank. Recommendation models can be trained on loss functions that approximate ranking-based evaluation metrics [17, 24, 28, 36], and these models may include side information [4, 23, 25, 35, 37]. Such approaches may require data in addition to the user-item matrix, per-item parameters, or use models where the output depends on the ordering of item attributes.

Recommendation model	Uses attributes	Only user-item interactions	Scalable	Permutation-invariant	Loss tied to evaluation
RANKFROMSETS	✔	✔	✔	✔	✔
Wang and Blei [32]	✔	✔		✔	
Gopalan et al. [8]	✔	✔		✔	
Dong et al. [7]	✔			✔	
Chen et al. [5]	✔		✔		
Bansal et al. [1]	✔	✔			
Xu et al. [34]	✔		✔	✔	
Rendle et al. [24]		✔			✔
Shi et al. [25]	✔	✔		✔	✔
Wu et al. [33]	✔		✔	✔	
Kula [14]	✔	✔		✔	
Shi et al. [26]		✔			✔
Chen and de Rijke [6]	✔	✔		✔	
Liu et al. [19]	✔			✔	✔
Cao et al. [4]	✔				✔
Okura et al. [23]	✔		✔		✔

Table 1: RANKFROMSETS is a scalable recommendation model that recommends items using attributes, and is trained on an objective function connected to an evaluation metric. Most methods we highlight use attributes; some require data in addition to the user-item matrix of observations. Some models are invariant to permutation of the attributes, and may use a loss function that is connected to a recommendation performance metric. Few methods are scalable, as most recommendation models that use item side information require learning parameters for every item.

3 THE RANKFROMSETS MODEL

RANKFROMSETS is a recommendation model that recommends items with attributes to users. Let $u \in \{1, \ldots, N\}$ be a user, $m \in \{1, \ldots, M\}$ be an item, and $y_{um} \in \{0, 1\}$ be a binary indicator where 1 indicates user u consumed item m. For each item m, there is an associated set of attributes $x_m \in \{0, 1\}^{|V|}$ from a vocabulary of V attributes.

We assume that a recommendation model is given a budget of K recommendations to be made for each user. In response, the recommender system produces a list of K distinct recommendations $\mathbf{r}_u = (r_{u1}, \ldots, r_{uK})$ for each user. The goal of the recommendation task in this paper is to maximize the expected recall,

$$\text{Recall@}K = \mathbb{E}_u \left[\frac{\sum_{r \in \mathbf{r}_u} y_{ur}}{\sum_m y_{um}} \right], \tag{1}$$

where the expectation is over all users in the empirical data distribution $\mathcal{D}$. RANKFROMSETS (RFS) combines three techniques to maximize Equation (1). First, we cast recommendation as a classification task. Second, we learn user- and attribute-level embeddings. Statistical strength is shared between items with similar attributes by representing items as the mean of their attribute embeddings. Third, we scale RFS to large datasets by using a stochastic optimization-based negative sampling training procedure to fit the model.

3.1 Recommendation through classification

RANKFROMSETS (RFS) casts the recommendation problem as a classification task. Given a user-item pair (u, m), RFS learns to predict the probability that item m will be consumed by user u,

$$p(y_{um} = 1 | u, m) = \sigma\left(f\left(u, x_m\right)\right), \tag{2}$$

where x_m is the set of attributes associated with item m and σ is the sigmoid function. The recommendations made by RFS are the maximum likelihood set,

$$\mathbf{r}_u(K) = \underset{\mathbf{r} \in \mathbb{N}^K}{\text{argmax}} \sum_{m \in \mathbf{r}} f\left(u, x_m\right). \tag{3}$$

To motivate treating recommendation as classification, we make the following observation.

PROPOSITION 1. *Let $u \in \mathcal{U}$ be a user, $x \in \mathcal{X}$ be an item, and $y(u, x) \in \{0, 1\}$ be an indicator of whether user u logged item x. Let $\mathcal{E}$ be the worst-case error for binary classifier $\hat{y}(u, x)$ on any (u, x) pair drawn from the data $\mathcal{D}$,*

$$\mathcal{E} = \max_{(u,x) \in \mathcal{D}} 1\left[\hat{y}(u, x) \neq y(u, x)\right].$$

A binary classifier with zero worst-case error ($\mathcal{E} = 0$) maximizes recommendation recall.

PROOF. A model with zero worst-case error is a perfect classifier: it assigns greater probability to positively-labeled datapoints than to negatively-labeled datapoints. In other words, it ranks positive examples above negative examples. Recall in Equation (1) is measured by the fraction of positively-labeled items in a ranking returned by the model. In a classifier that achieves zero worst-case error, positively-labeled datapoints must be ranked higher than other datapoints, maximizing recall. □

Proposition 1 is simple, but conceptually important. Under the assumption that a perfect classifier exists, a consistent method for

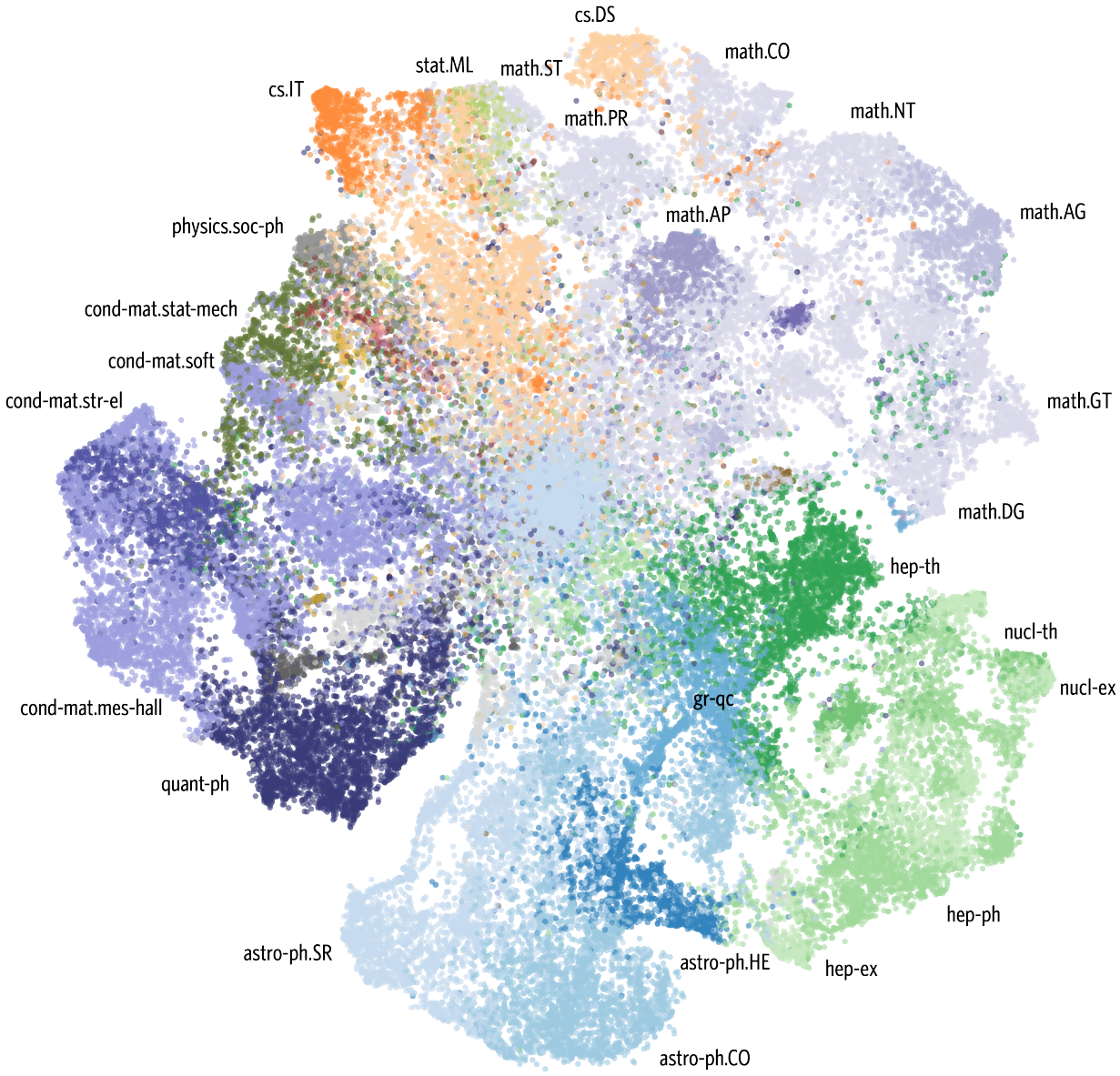


Figure 1: RANKFROMSETS trained on arXiv reading behavior clusters researchers by their most frequently-read arXiv category (best viewed in color). On this data, RANKFROMSETS is trained to recommend items using their attributes as described in Section 4.2; the set of attributes for a paper is the set of words in the abstract. A visualization of the user embeddings in the inner product regression function in Equation (4) yields an interpretable map of science. We use the t-SNE algorithm [21] for visualizing the high-dimensional embeddings in two dimensions. In this map of science, fields of study are related according to patterns in how people read papers in neighboring fields. Each marker represents a user embedding; the color assigned to a user is determined the user's most-read arXiv category. The color assigned to a category is determined by the most-read categories across the arXiv, with similar colors assigned to similar fields according to the arXiv ontology. For an interactive version of this map, please visit https://github.com/kdd-anonymous/rankfromsets which enables zoom and display of all 143 arXiv category labels to explore the relationships between different fields of science.

learning a classifier will be a consistent method for learning a recommendation system that targets expected recall. In practice, as with any regression method, a perfect classifier is unachievable. Proposition 1 is a guiding principle rather than a finite-sample guarantee of maximal performance. As we show in Section 4, the classification approach of RFS performs well in practice.

3.2 Embedding users and items

For recommending items with attributes, Proposition 1 says that building a classifier such as RANKFROMSETS is optimal if we measure recommendation performance with recall. To parameterize the RFS classifier, we need to choose a regression function $f(u, x_m)$. A straightforward parameterization is an inner product,

$$f(u, x_m) = \theta_u^\top \left(\frac{1}{|x_m|} \sum_{j \in x_m} \beta_j + g(x_m) \right) + h(x_m) . \quad (4)$$

Each element in the inner product regression function in Equation (4) has an intuitive interpretation:

- The user embedding $\theta_u \in \mathbb{R}^d$ captures the latent preferences for user u. This captures the individual-level tastes of a user and is analogous to the user preference vector in classical collaborative filtering or the row embedding in matrix factorization.
- The attribute embedding $\beta_j \in \mathbb{R}^d$ is the latent quality conveyed through item m having attribute j. The set x_m contains only attributes with $x_{mj} = 1$. Attributes that are not associated with item m are ignored.
- The item embedding function $g(x_m) \in \mathbb{R}^d$ captures qualities not conveyed through the set of item attributes. This term in the regression function enables collaborative filtering by capturing unobserved patterns in item consumption such as popularity. We describe how to construct this function below.
- The item intercept function $h(x_m) \in \mathbb{R}$ makes an item more or less likely simply due to availability.

Scalable item embedding and item intercept functions. The parameterization of the item embedding function $g(x_m)$ depends on the size of the data. If the number of items is small, g can function as a lookup for unique intercepts for every item. However, if the number of items is so large that unique item intercepts lead to overfitting, we need a scalable parameterizations of item embeddings g using additional information about every item. For example, if the data consists of foods in meals, we can define a meal intercept as the mean of food intercepts, yielding a scalable item intercept function. The item intercept function $h(x_m)$ that maps item attributes to scalars is constructed in the same way. We study both of these choices in Section 4.

The inner product regression function in Equation (4) has several benefits. It requires computing a sum over only the attributes each item is associated with. This enables RFS to scale to large attribute vocabularies where traditional matrix factorization methods are intractable. Second, the embed-and-sum approach to set modeling is provably flexible. We describe deep variants of RFS and detail how RFS can approximate other recommendation models in the next section.

3.3 Deep extensions

The RFS inner product regression function in Equation (4) is a log-bilinear model. But there are several other choices of regression function, and we draw on the deep learning toolkit for classification to build two other example architectures. With finite data and finite compute, one architecture will outperform another, or prove insufficiently flexible to capture patterns in user consumption. The optimal choice of architecture depends on computational tradeoffs. We conclude this section by showing that in the regime of infinite data and compute, all the RFS architectures we propose, including the inner product, can approximate other recommendation models that operate on set-valued input such as matrix factorization.

Parameterizing RFS using a neural network. As an alternative to the log-bilinear model in Equation (4), we can use a deep neural network as a regression function:

$$f(u, x_m) = \phi \left(\theta_u, \frac{1}{|x_m|} \sum_{j \in x_m} \beta_j, g(x_m) \right) + h(x_m) , \quad (5)$$

where the deep network ϕ has weights and biases and takes as inputs the user embedding, sum of attribute embeddings, and item intercept. Such a neural network can represent functions that may or may not include the inner product in Equation (4) (*ex ante*, it is unclear whether a finite-depth, finite-width neural network can represent the inner product).

Parameterizing RFS with a residual network. Another possibility of regression function for RFS is a combination of Equations (4) and (5), using an idea borrowed from deep residual networks for image classification [10]. In this architecture a neural network ϕ with the same inputs as in Equation (5) is used to learn the residual in the inner product model:

$$f(u, x_m) = \theta_u^\top \left(\frac{1}{|x_m|} \sum_{j \in x_m} \beta_j + g(x_m) \right) + \phi + h(x_m) . \quad (6)$$

The choice of regression function in RFS depends on the data. On finite data, with finite compute, one parameterization of RFS will outperform another. To demonstrate this, we simulated synthetic data from the same generative process RFS employs with a ground-truth regression function (a square kernel), and found that the residual and deep parameterizations outperformed the inner product architecture. These results are included in Appendix B, and motivate exploring other architectures than the three examples we give.

3.4 Generalization property

If we take a step back from the setting of finite data and compute, a bigger picture emerges, which reveals the choice of regression function in RFS does not matter. Any RFS architecture is sufficiently

flexible to approximate other recommendation models that operate on set-valued input. Before deriving this result, we first describe the class of recommendation models that operate on set-valued input.

RFS and multiple matrix factorization are examples of permutation-invariant models. The regression function f in RFS operates on set-valued input: the unordered collection of item attributes x_m. A set is, by definition, permutation-invariant: it remains the same if we permute its elements. Functions that operate on set-valued inputs must also be permutation-invariant. RFS is permutation-invariant as the set of attributes associated with an item enter into Equations (4) to (6) via summation. Other examples of permutation-invariant models are matrix factorization, recommendation models based on word embeddings, and permutation-marginalized recurrent neural networks. We show these models are permutation-invariant and evaluate them in Section 4.

RFS can approximate other permutation-invariant models such as matrix factorization. We apply a theorem from Zaheer et al. [38] to show that RFS can approximate other recommendation models.

PROPOSITION 2. *Assume the vocabulary of attributes (set elements) is countable, $|V| < |\mathbb{N}_0|$. RFS can approximate any permutation-invariant recommendation model.*

The proof follows directly from Theorem 2 in Zaheer et al. [38] and we will not restate it here. (The only change to the proof is the mapping from set elements to one-hot vectors, $c : V \to \{0,1\}^{|V|}$ to yield a unique representation of every object in the powerset.)

Proposition 2 means that any of the parameterizations in Equations (4) to (6) are flexible enough to approximate other principled recommendation models that leverage item attributes, such as content-based matrix factorization [8]. This proposition also supports exploring other parameterizations of RFS that may have better computational or statistical properties.

3.5 Stochastic optimization

The parameters for RANKFROMSETS are learned by stochastic optimization. Denote the full set of RFS model parameters by $\boldsymbol{\gamma}$, and let $\mathcal{D}_u$ be the empirical data distribution for a user. Let λ_u be a reweighting parameter. The maximum likelihood objective for RFS is

$$\begin{aligned}\mathcal{L}(\boldsymbol{\gamma}, \lambda_u) = \mathbb{E}_u\big[&\mathbb{E}_{x_m \sim \mathcal{D}_u \mid y_{um}=1}\left[\log p(y_{um}=1 \mid x_m; \boldsymbol{\gamma})\right] \\ &-\lambda_u \mathbb{E}_{x_k \sim \mathcal{D}_u \mid y_{uk}=0}[\log p(y_{uk}=0 \mid x_k; \boldsymbol{\gamma})]\big]\end{aligned} \tag{7}$$

In traditional regression, altering the ratio of positive to negative examples by reweighting leads to inconsistent parameter estimation. The inconsistency stems from the randomness in the labels, given the features. However, recall in Equation (1) assumes that each user, item attribute set pair (u, x_m) uniquely determines whether the item was consumed or not (the label y_{um}). Here, all reweightings produce the same result. This means that for any negative example weight λ_u, the learned model will be the same. In practice we set λ_u to balance the positive and negative examples for each user. We use stochastic optimization to maximize Equation (7). We describe two negative sampling schemes that are dependent on the choice of evaluation metric.

Corpus sampling. Negative samples can be drawn uniformly over the entire corpus of items. If the item set is large, this can be an expensive procedure. This negative sampling scheme and resulting approximation of Equation (7) is similar to the objective functions in other recommender systems [11, 28].

On large datasets, it is infeasible to calculate recall for evaluation, as Equation (1) requires ranking every item for every user (e.g. in Section 4.3 we study a dataset with over 10M items). We define a scalable evaluation metric based on recall, and describe how it leads to a natural choice of negative sampling distribution.

Batch sampling and sampled recall. We define sampled recall as follows. Start with held-out datapoints with positive labels, $(x_m, y_{um} = 1)$. For every held-out datapoint, $K-1$ datapoints with negative labels $(x_k, y_{uk} = 0)$ are sampled from the rest of the held-out data, which together yield a set of K datapoints. A recommendation model is used to rank the K datapoints $r_{u1}, \ldots, r_{uK}$. Sampled recall is the fraction of the held-out datapoints that the model ranked in the top k:

$$\text{SampledRecall@}k = \mathbb{E}_{um}\left[\frac{\sum_{r \in \{r_{u1}, \ldots r_{uk}\}} y_{ur}}{K}\right], \tag{8}$$

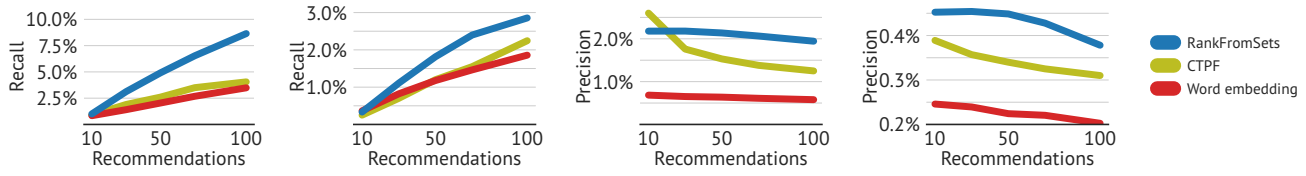
where the expectation is over users and items in the held-out set of datapoints. This evaluation metric is scalable: instead of using a model to rank every item, it requires ranking K items. Sampled recall is 1 if $k = K$, as the held-out datapoint with $y_{um} = 1$ is in each list of K datapoints to be ranked.

When sampled recall is used as an evaluation metric, batch sampling is a natural way to draw negative samples. Sampled recall is calculated on items drawn from other user's data. We define batch sampling as generating negative samples by permuting mini-batch items. Besides corresponding to the sampled recall metric, this technique is memory-efficient, as it requires that only the current mini-batch be in memory.

In addition to scalability, both of the negative sampling procedures above have the advantage of implicitly balancing the classifier. As Veitch et al. [31] note, using stochastic gradient descent with negative sampling is equivalent to a Monte Carlo approximation of the reweighted, or balanced, classification loss.

4 EXPERIMENTS

We study the performance of RANKFROMSETS on two datasets. The first data consists of researcher reading behavior from the arXiv; the task is to recommend documents to scientists. The second is crowdsourced food consumption data from a diet tracking app, and the task is meal recommendation. On both benchmarks, RFS outperforms several baseline methods.



(a) In-matrix recall **(b) Out-matrix recall** **(c) In-matrix precision** **(d) Out-matrix precision**

Figure 2: RANKFROMSETS with the inner product regression function in Equation (4) outperforms collaborative topic Poisson factorization (CTPF, described in Gopalan et al. [8]) and a word embedding model on recommending arXiv papers to scientists. In this dataset the items are documents and the attributes are the unique words in the abstracts. Recommendation performance is evaluated using precision and recall to match the evaluation in Gopalan et al. [8], and these metrics are reported on training (in-matrix) documents and cold-start (out-matrix) documents with no clicks in the training set.

Reproducibility. An example implementation is in Appendix A and anonymized code for running the simulation study in Appendix B is at https://github.com/kdd-anonymous/rankfromsets (data cannot be shared, as releasing diet or arXiv data is associated with several privacy concerns).

4.1 Baselines

We compare RFS to a suite of recommendation models that rank items using their sets of attributes.

Word embedding model. We fit a word embedding model [22] on the training data using the implementation of fastText from Bojanowski et al. [3]. The dimensionality of the embeddings is set to match models being compared to. We use a context window in the model that is slightly different than in the original implementation. For an item with attributes x_m, the context for attribute $j \in x_m$ is the set of other attributes of the same item $j' \in x_m : j' \neq j$. After learning the attribute embeddings β_j for $j \in V$, item embeddings are computed as the average of their attribute embeddings. Users are represented as the average of the embeddings of the items they consume. Recommendation is performed by cosine similarity of an item embedding and a user embedding.

Collaborative topic Poisson factorization. This is a probabilistic matrix factorization model of user consumption data. The recommendation model follows a latent variable scheme,

1. **Document model:**
   (a) Draw topics $\beta_{vk} \sim \text{Gamma}(a, b)$
   (b) Draw document topic intensities $\theta_{dk} \sim \text{Gamma}(c, d)$
   (c) Draw word count $w_{dv} \sim \text{Poisson}(\theta_d^T \beta_v)$.
2. **Recommendation model:**
   (a) Draw user preferences $\eta_{uk} \sim \text{Gamma}(e, f)$
   (b) Draw document topic offsets $\epsilon_{dk} \sim \text{Gamma}(g, h)$
   (c) Draw $r_{ud} \sim \text{Poisson}(\eta_u^T(\theta_d + \epsilon_d))$.

Gopalan et al. [8] develop a variational inference algorithm for posterior inference in this model; we use their implementation.

Collaborative topic Poisson factorization is a permutation-invariant recommendation model. To show that collaborative topic Poisson factorization [8] is permutation-invariant, consider the Poisson likelihood function over words w_{dv}. Conditional on the latent item representation θ_d and latent word representation β_v, every word in the document w_{dv} is independent; the joint probability of words in a document factorizes:

$$p(w_d \mid \theta_d, \beta_v) = \prod_{w_{dv} \in w_d} p(w_{dv} \mid \theta_d, \beta_v). \tag{9}$$

In this model, predictions are made using expectations under the posterior. The posterior is proportional to the the log joint of the model, and the attributes of items (words in documents) enter into the model only via the above product. The product of the probability of words in a document is invariant to a reordering of the words in the document. Therefore, collaborative topic Poisson factorization is permutation-invariant.

Permutation-marginalized recurrent neural network. The Bernoulli distribution used in Equation (4) can be parameterized using a recurrent neural network. We treat attributes as a sequence and marginalize over all permutations of orderings,

$$p(y_{um} = 1 \mid x_m) = \frac{1}{|\pi(x_m)|} \sum_{\pi \in \pi(x_m)} \sigma\left(\phi(\theta_u, \{\beta_{\pi(1)}, \ldots, \beta_{\pi(J)}\})\right). \tag{10}$$

Here β_j are attribute embeddings, $\pi(x_m)$ denotes the set of all permutations of the set x_m, and ϕ is output of an LSTM recurrent neural network [12] projected to one dimension using a linear layer. This model is permutation-invariant as there is an explicit sum over all permutations.

4.2 Recommending research papers

We benchmark RFS on data of scientists reading research papers on the arXiv, where the goal is to recommend papers to scientists. The arXiv usage data represents one year of usage (2012) and consists of 65k users, 637k preprints, and 7.6M clicks.

Evaluation. We follow Gopalan et al. [8] and use the same test and validation splits, and the same set of held-out 10k users. To match

Gopalan et al. [8], we compute precision in addition to recall. The held-out validation and test splits each consist of 20% of the ratings and 1% of the documents. In-matrix documents refer to documents that have clicks in the training data, while out-matrix or cold-start documents have no previous clicks.

Hyperparameters for RFS. We test the stochastic gradient descent algorithm with and without momentum [29]. We use a linear learning rate decay that decays to zero in the maximum number of iterations, 200k. We perform a grid search over learning rates of $1, 5, 10, 15, 25$ and momenta of $0.5, 0.9, 0.95, 0.99$. The minibatch size is set to $\{2^{16}\}$. We use a single negative sample per datapoint, sampled uniformly over the entire dataset (corpus sampling is defined in Section 3.5). To match the hyperparameters in Gopalan et al. [8], we set the dimensionality of embeddings to 100. Evaluation is performed every 20k iterations.

Hyperparameters for the permutation-marginalized recurrent neural network. The embedding and hidden state sizes are fixed to 100. Evaluation is performed every 20k iterations. We grid search over learning rates of $10^{-1}, 10^{-2}, 10^{-3}$ with the Adam optimizer [13]. If validation performance does not improve, we reload the best parameters and optimizer states, and decay the learning rate by 0.9. We subsample single permutations during training and testing.

Results. Figure 2 shows that RFS with the inner product parameterization outperforms collaborative topic Poisson factorization in terms of in-matrix recall by over 90%. It also improves over collaborative topic Poisson factorization (CTPF) in terms of out-matrix recall, out-matrix precision, and in-matrix precision (for the latter, only when the number of recommendations is greater than 30). The word embedding model performs about as well as CTPF in terms of recall, but performs worse in terms of precision. The permutation-marginalized recurrent neural network model performance was an order of magnitude worse than the other methods; we report its performance in Table 3.

Qualitatively, RFS reveals patterns in usage of the arXiv. Figure 1 is a dimensionality-reduced plot of the user embeddings that reveals connections between fields of study. Scientists who focus on high energy physics, `hep`, neighbor specialists in differential geometry, `math.DG`; these areas share techniques. Machine learning researchers, `stat.ML` readers, neighbor statisticians or `math.ST` readers, highlighting the close connection between these fields. Plots for document embeddings show similar patterns. This illustrates how RFS captures rich patterns of interaction between users and items, while benefitting from information in the item attributes.

4.3 Recommending meals

We evaluate RFS on data collected from the LoseIt! food tracking app. This app enables users to track their food intake to eat healthy. We use a year's worth of data from 55k active users. This corresponds to 16M meals, where each meal is comprised of a subset of 3M foods.

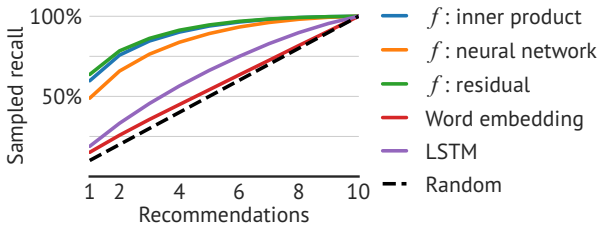


Figure 3: RANKFROMSETS outperforms models based on word embeddings permutation-marginalized recurrent neural networks (denoted by LSTM) on meal recommendation. The recommendation models are trained on data from a food tracking app as described in Section 4.3 and are evaluated using the sampled recall metric, Equation (8). The inner product, neural network, and residual regression functions for RANKFROMSETS are in Equations (4) to (6).

Preprocessing. Each meal is a subset of 3M foods. We filter the vocabulary by keeping words that occur at least 20 times in the food names, resulting in 9963 words. A meal is represented as the union of the sets of words occurring in the food names.

Evaluation. We hold out 1% of the items (meals) for validation and hold out 1% of the items for evaluating test performance. We use the sampled recall metric in Equation (8) with $K = 10$.

Hyperparameters for RFS. The embedding size is set to 128. For the neural network and residual models in Equations (5) and (6) the number of hidden layers is two, and the number of hidden units is set to 256. The item embeddings $g(x_m)$, and item intercepts $h(x_m)$, are computed as the mean of learned food embeddings or intercepts, respectively. We use the RMSProp optimizer in Graves [9] and grid search over learning rates in $\{10^{-2}, 10^{-3}, 10^{-4}, 10^{-5}\}$. We use a batch size of 64 and a single negative sample for every datapoint in a minibatch (batch sampling is defined in Section 3.5). Evaluation is performed every 50k iterations.

Hyperparameters for the permutation-marginalized recurrent neural network. We grid search over batch sizes of $\{32, 64, 128\}$. For every item in a minibatch, we sample a single permutation of attributes to approximate the sum in Equation (10). We use a single negative sample per datapoint, and set the embedding and hidden state sizes to 128. We use the Adam optimizer [13] and grid search over learning rates of $\{10^{-2}, 10^{-3}, 10^{-4}\}$. Evaluation is performed every 1k iterations. If the validation performance does not improve, we reload the best parameters and optimizer states, and decay the learning rate by 0.9.

Results. Figure 3 plots the sampled recall. This shows that RFS outperforms the permutation-marginalized recurrent neural network and the word embedding model. The code released by the authors of the collaborative topic Poisson factorization model in [8] did not scale to this size of data.

Query meal	Nearest meal by cosine similarity
Two scoops of Raisin Bran cereal, organic Moroccan green tea, almond milk, light honey, tap water, large banana, large strawberries	Vita Bee bread, salted butter, fresh medium tomatoes, large fried whole egg, small banana
Iceberg lettuce, cantaloupe cubes, diced honeydew melon, cherry tomatoes, olives, dry-cooked unsalted hulled sunflower seed kernels, chopped hard-boiled egg, cucumbers, dried cranberries, fat-free ranch dressing	Green leaf lettuce, chopped sweet red bell peppers, crumbled feta cheese, large hard-boiled egg, chopped cucumber, oil-roasted salted sunflower seeds, sliced radishes, sliced strawberries, pitted Calamata olives, fat-free balsamic vinegar
Boston roast pork, mackerel, artichoke hearts, spinach, pimiento-stuffed Manzanilla olives, carrots, mushrooms, peppercorn ranch dressing	Broiled top round steak, tomatoes, cucumber, baby yellow squash, zucchini, black olives, extra virgin olive oil
Meatloaf with tomato sauce, chopped sweet red bell peppers, extra virgin olive oil, cooked asparagus spears, sweet potatoes, orange, cantaloupe cubes	Chicken breast, breadcrumbs, fresh tomatoes, shredded green leaf lettuce, extra virgin olive oil, spinach, chopped yellow onion, sweet large yellow bell peppers, whole mushrooms, chili peppers, vinaigrette
Ciabatta bun, cooked skinless chicken breast, fresh baby spinach, shredded iceberg lettuce, shredded mozzarella cheese, ketchup, frozen yogurt bar	Small whole wheat submarine roll, broiled round roast beef, roasted light turkey meat without skin, fresh medium tomatoes, honey smoked ham, shredded iceberg lettuce, sliced mozzarella cheese

Table 2: RANKFROMSETS, trained on food consumption data, provides diverse recommendations. We fit the model to data from a food tracking app as described in Section 4.3; items are meals and attributes are the ingredients in the meal. We represent meals as the mean of their attribute embeddings, and use cosine similarity to compute the nearest neighbors of meals. This reveals that RANKFROMSETS uncovers latent patterns of consumption in the attribute embeddings that can be leveraged to improve recommendations. For example, the second-last query meal is a mix of meat, vegetables, and fruit, and the nearest neighbor meal is a different meat and a side of salad. The last query meal is a sandwich, and its nearest neighbor is also a sandwich, but with different ingredients.

Qualitatively, RFS learns an interpretable representation of items, as shown by nearest neighbors of meals in Table 2. In this table, we display breakfast, lunch, and dinner meals, alongside their nearest neighbors. We find that the nearest neighbors are also breakfast, lunch, and dinner meals respectively, showing that the attribute embeddings learned by the model can be used to explore qualitative patterns in the learned latent space.

5 DISCUSSION

The task of recommending items with attributes is difficult for several reasons. It is unclear how to incorporate set-valued side information into models that scale to large numbers of items and attributes. In addition, existing recommendation models that leverage item attributes are not directly tied to evaluation metrics. We developed RANKFROMSETS, a scalable recommendation model for items with attributes. Theoretically, we showed that optimizing its loss function optimizes recall, and that it can approximate a class of recommendation models including content-based matrix factorization. Empirically, RFS outperformed competitors and scaled to large datasets.

Simple theory and implementation. It is surprising that RFS outperforms the collaborative topic Poisson factorization model in Gopalan et al. [8], as RFS is a much simpler model. RFS does not require approximate posterior inference as in CTPF, but mere binary classification. RFS is also simpler in terms of implementation: a scalable example is a few dozen lines of python in Appendix A, compared to several thousand lines of C++ released by the authors of CTPF.

Architectures. There is a wealth of deep learning models that can be used to parameterize RFS. For example, attention-based transformer networks can operate on set-valued input [15] and are an interesting choice of architecture for RFS.

Side information. The RFS model can easily accommodate various metadata about users, items, or attributes. Attribute-level or user-level side information is particularly interesting, and when available, should lead to improved recommendation performance.

Regularization. Further performance gains from regularizing RFS are also interesting avenues for future work. L2 regularization for sparsity and dropout for decorrelation may yield improvements.

Pretrained embeddings. We note that the word embedding baseline performed slightly worse as collaborative topic Poisson factorization, which is surprising given that the word embedding model is much simpler. Further performance gains in RFS should be possible by initializing user and item attribute embeddings to pretrained word embeddings as in Chen et al. [5].

Theory (approximation). Proposition 2 means that RFS can approximate the output of any order-invariant model such as matrix factorization. However, there is no guarantee that fit with maximum likelihood estimation using the objective in Equation (7), RFS *will* approximate a specific model. We leave the development of such approximation techniques to future work.

Theory (ranking metrics). How well does binary classification perform for other ranking-based recommendation metrics, such as non-discounted cumulative gain? Analyzing this question is more difficult, and we leave this to future work. We conjecture that a different loss function should allow a similar proof to Proposition 1.

Theory of generalization. With sufficient data, RANKFROMSETS can learn arbitrary distributions of users consuming items with attributes. But performance on finite data can vary. Developing generalization theory for RANKFROMSETS remains an open question.

REFERENCES

[1] Trapit Bansal, David Belanger, and Andrew McCallum. 2016. Ask the GRU: Multi-task Learning for Deep Text Recommendations. *ACM Recommender Systems.*

[2] Toine Bogers. 2018. *Social Information Access.* Springer International Publishing, Chapter Tag-Based Recommendation, 441–479.

[3] Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. 2017. Enriching Word Vectors with Subword Information. *Association for Computational Linguistics.*

[4] Da Cao, Liqiang Nie, Xiangnan He, Xiaochi Wei, Shunzhi Zhu, and Tat-Seng Chua. 2017. Embedding Factorization Models for Jointly Recommending Items and User Generated Lists. *ACM Special Interest Group on Information Retrieval.*

[5] Ting Chen, Liangjie Hong, Yue Shi, and Yizhou Sun. 2017. Joint Text Embedding for Personalized Content-based Recommendation. *arXiv:1706.01084.*

[6] Yifan Chen and Maarten de Rijke. 2018. A Collective Variational Autoencoder for Top-N Recommendation with Side Information. *ACM Workshop on Deep Learning for Recommender Systems.*

[7] Xin Dong, Lei Yu, Zhonghuo Wu, Yuxia Sun, Lingfeng Yuan, and Fangxi Zhang. 2017. A Hybrid Collaborative Filtering Model with Deep Structure for Recommender Systems. *AAAI Conference on Artificial Intelligence.*

[8] Prem Gopalan, Laurent Charlin, and David M. Blei. 2014. Content-based Recommendations with Poisson Factorization. *Neural Information Processing Systems.*

[9] Alex Graves. 2013. Generating Sequences With Recurrent Neural Networks. *arXiv:1308.0850.*

[10] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep Residual Learning for Image Recognition. *IEEE Conference on Computer Vision and Pattern Recognition.*

[11] Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu, and Tat-Seng Chua. 2017. Neural collaborative filtering. *International World Wide Web Conference.*

[12] Sepp Hochreiter and Jurgen Schmidhuber. 1997. Long Short-Term Memory. *Neural Computation* 9, 8, 1–32.

[13] Diederik P. Kingma and Jimmy Ba. 2015. Adam: A Method for Stochastic Optimization. *International Conference on Learning Representations.*

[14] Maciej Kula. 2015. Metadata Embeddings for User and Item Cold-start Recommendations. *arxiv:1507.08439.*

[15] Juho Lee, Yoonho Lee, Jungtaek Kim, Adam R. Kosiorek, Seungjin Choi, and Yee Whye Teh. 2018. Set Transformer. *arXiv:1810.00825.*

[16] Jianxun Lian, Fuzheng Zhang, Xing Xie, and Guangzhong Sun. 2018. Towards Better Representation Learning for Personalized News Recommendation: a Multi-Channel Deep Fusion Approach. *International Joint Conference on Artificial Intelligence.*

[17] Junjie Liang, Jinlong Hu, Shoubin Dong, and Vasant G. Honavar. 2018. Top-N-Rank: A Scalable List-wise Ranking Method for Recommender Systems. *arXiv:1812.04109.*

[18] Nan Liang, Hai-Tao Zheng, Jin-Yuan Chen, Arun Sangaiah, and Cong-Zhi Zhao. 2018. TRSDL: Tag-Aware Recommender System Based on Deep Learning–Intelligent Computing Systems. *Applied Sciences.*

[19] Yidan Liu, Min Xie, and Laks V.S. Lakshmanan. 2014. Recommending User Generated Item Lists. *ACM Recommender Systems.*

[20] Benedikt Loepp, Tim Donkers, Timm Kleemann, and Jürgen Ziegler. 2019. Interactive recommending with Tag-Enhanced Matrix Factorization (TagMF). *International Journal of Human-Computer Studies.*

[21] Laurens van der Maaten and Geoffrey Hinton. 2008. Visualizing Data using t-SNE. *Journal of Machine Learning Research* 9, 2579–2605.

[22] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Distributed Representations of Words and Phrases and their Compositionality. *Neural Information Processing Systems.*

[23] Shumpei Okura, Yukihiro Tagami, Shingo Ono, and Akira Tajima. 2017. Embedding-based News Recommendation for Millions of Users. *ACM Knowledge Discovery and Data Mining.*

[24] Steffen Rendle, Christoph Freudenthaler, Zeno Gantner, and Lars Schmidt-Thieme. 2009. BPR: Bayesian Personalized Ranking from Implicit Feedback. *Uncertainty in Artificial Intelligence.*

[25] Yue Shi, Alexandros Karatzoglou, Linas Baltrunas, Martha Larson, Alan Hanjalic, and Nuria Oliver. 2012. TFMAP: Optimizing MAP for Top-N Context-Aware Recommendation. *ACM Special Interest Group on Information Retrieval.*

[26] Yue Shi, Alexandros Karatzoglou, Linas Baltrunas, Martha Larson, Nuria Oliver, and Alan Hanjalic. 2012. CLiMF: Learning to Maximize Reciprocal Rank with Collaborative Less-is-more Filtering. *ACM Recommender Systems.*

[27] Yue Shi, Martha Larson, and Alan Hanjalic. 2014. Collaborative Filtering Beyond the User-Item Matrix: A Survey of the State of the Art and Future Challenges. *Comput. Surveys* 47, 1, 1–45.

[28] Bo Song, Xin Yang, Yi Cao, and Congfu Xu. 2018. Neural Collaborative Ranking. *ACM Conference on Information and Knowledge Management.*

[29] Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hinton. 2013. On the Importance of Initialization and Momentum in Deep Learning. *International Conference on Machine Learning.*

[30] Adrianus Willem van der Vaart. 1998. *Asymptotic Statistics.* Cambridge University Press.

[31] Victor Veitch, Morgane Austern, Wenda Zhou, David M. Blei, and Peter Orbanz. 2019. Empirical Risk Minimization and Stochastic Gradient Descent for Relational Data. *International Conference on Artificial Intelligence and Statistics.*

[32] Chong Wang and David M. Blei. 2011. Collaborative Topic Modeling for Recommending Scientific Articles. *ACM Knowledge Discovery and Data Mining.*

[33] L. Wu, A. Fisch, S. Chopra, K. Adams, A. Bordes, and J. Weston. 2018. StarSpace: Embed All The Things! *AAAI Conference on Artificial Intelligence.*

[34] Zhenghua Xu, Thomas Lukasiewicz, Cheng Chen, Yishu Miao, and Xiangwu Meng. 2017. Tag-Aware Personalized Recommendation Using a Hybrid Deep Model. *International Joint Conference on Artificial Intelligence.*

[35] Haochao Ying, Liang Chen, Yuwen Xiong, and Jian Wu. 2016. Collaborative Deep Ranking: A Hybrid Pair-Wise Recommendation Algorithm with Implicit Feedback. *Pacific-Asia Conference on Knowledge Discovery and Data Mining.*

[36] Lu Yu, Chuxu Zhang, Shichao Pei, Guolei Sun, and Xiangliang Zhang. 2018. WalkRanker: A Unified Pairwise Ranking Model With Multiple Relations for Item Recommendation. *AAAI Conference on Artificial Intelligence.*

[37] Fajie Yuan, Guibing Guo, Joemon M. Jose, Long Chen, Haitao Yu, and Weinan Zhang. 2016. Optimizing Factorization Machines for Top-N Context-Aware Recommendations. *Web Information Systems Engineering.*

[38] Manzil Zaheer, Satwik Kottur, Siamak Ravanbakhsh, Barnabás Póczos, Ruslan Salakhutdinov, and Alexander J. Smola. 2017. Deep Sets. *Neural Information Processing Systems.*

[39] Shuai Zhang, Lina Yao, and Aixin Sun. 2017. Deep Learning Based Recommender System: A Survey and New Perspectives. *arXiv:1707.07435.*

[40] Yi Zhen, Wu-Jun Li, and Dit-Yan Yeung. 2009. TagiCoFi: Tag Informed Collaborative Filtering. *ACM Recommender Systems.*

[41] Yi Zuo, Jiulin Zeng, Maoguo Gong, and Licheng Jiao. 2016. Tag-aware recommender systems based on deep neural networks. *Neurocomputing.*

A REPRODUCIBILITY SUPPLEMENT: CODE

We give an example implementation of RANKFROMSETS with the inner product regression function in Equation (4) in python with the PyTorch package (version 1.0.0). This shows how easy it is to implement RFS; we used a similar implementation to achieve state-of-the-art results in Section 4.2.

```
import torch
from torch import nn
import data

class InnerProduct(nn.Module):
  def __init__(
        self, n_users, n_items, n_attributes, emb_size):
    super().__init__()
    self.user_embeddings = nn.Embedding(n_users, emb_size)
    self.attribute_emb_sum = nn.EmbeddingBag(
        n_attributes, emb_size, 'mean')
    self.item_embeddings = nn.Embedding(n_items, emb_size)
    self.intercepts = nn.Embedding(n_items, 1)

  def forward(
        self, users, items, item_attributes, offsets):
    user_emb = self.user_embeddings(users)
    attr_emb = self.attribute_emb_mean(
        item_attributes, offsets)
    item_emb = self.item_embeddings(items)
    logits = user_emb * (attr_emb + item_emb)
    logits = logits.sum(-1, keepdim=True)
    return logits + self.intercepts(items)

train_data = data.load_data()
learning_rate, batch_size = 0.1, 1000
model = InnerProduct(train_data.n_users,
                     train_data.n_items,
                     train_data.n_attributes,
                     emb_size=100)
optimizer = torch.optim.SGD(
    model.parameters(), learning_rate)
loss = torch.nn.BCEWithLogitsLoss()
# negative samples are in last half of each batch
labels = torch.cat(torch.ones(batch_size // 2),
                   torch.zeros(batch_size // 2))
for batch in train_data:
  model.zero_grad()
  logits = model(*batch)
  L = loss(logits, labels)
  L.backward()
  optimizer.step()
```

# recommendations	10	30	50	70	100
Recall (percent)	0.02	0.06	0.11	0.14	0.21

Table 3: The permutation-marginalized LSTM model performs poorly on the arXiv dataset of researcher reading behavior in terms of in-matrix recall. We report the held-out recall averaged over 100 users due to the computational constraints of the model. It is much lower than that of other methods, but better than random.

B REPRODUCIBILITY SUPPLEMENT: SIMULATION STUDY

With finite data and a finite number of paramaters, the optimal parameterization of RFS is dependent on the data-generating distribution. Recall that observations of user-item interactions are generated by a Bernoulli distribution with logit function f. We describe a choice of logit function f that leads to the residual and deep architectures in Equations (5) and (6) outperforming the inner product architecture in Equation (4) in terms of predictive performance. Note that the number of parameters across architectures must be equal for a fair comparison. The code required to replicate this experiment is included here:

https://github.com/kdd-anonymous/rankfromsets.

We simulate data from the following generative process:

1. **For every user u:**
   (a) Draw user embedding $\theta_u \sim \text{Normal}(0, \text{I})$.
2. **For every item attribute j:**
   (a) Draw attribute embedding $\beta_j \sim \text{Normal}(0, \text{I})$.
3. **For every item m:**
   (a) Draw item topics $\theta_m \sim \text{Dirichlet}(\alpha)$
   (b) Draw number of item attributes $M \sim \text{Poisson}(\lambda)$
   (c) Draw nonzero item attributes $x_m \sim \text{Multinomial}(M, \theta_m)$.
4. **For every user, item observation:**
   (a) $y_{um} \sim \text{Bernoulli}\left(y_{um}; \sigma(f(\theta_u, x_m))\right)$

The logit function f is the square kernel:

$$f(\theta_u, x_m) = \left(\theta_u^\top \frac{1}{|x_m|} \sum_{j \in x_m} \beta_j\right)^2.$$

Before sampling from the Bernoulli, we standardize the logits output by f across users and subtract 7 to achieve sparse user-item observations.

Simulation setup. We set the Dirichlet parameter to be $\alpha = 0.01$ and the Poisson rate to be $\lambda = 20$. We generate data for 1k users, 5k item attributes, 30k items, and hold out 100 users for each of the validation and test sets. We fix the momentum to 0.9 [29] and grid search over stochastic gradient descent learning rates of $10, 1, 0.1, 0.01$ and over two learning rate decay schedules. The first linear learning rate decay goes to zero over 100k iterations, while the second divides the learning rate by 10 if the validation in-matrix recall does not improve (evaluation is performed every 500 iterations). We run the grid search on one instance of data generated from this model, then for the best performing hyperparameters for each model trained on this instance, we regenerate data 30 times and average results over these synthetic datasets.

Regression function	Inner product	Deep	Residual
Recall	0.29 ± 0.15	0.32 ± 0.14	0.33 ± 0.18

Table 4: A simulation study demonstrating that the choice of parameterization of RANKFROMSETS is data-dependent. We report the in-matrix recall averaged over 100 users and 30 replications of the simulation where data is regenerated. The residual model in Equation (6) outperforms the deep model, Equation (5), and the inner product model in Equation (4).

Simulation results. The results in Table 4 demonstrate that the residual model outperforms both the deep and inner product architectures for data generated by the above generative process.

Generalization. The above example shows that the choice of architecture in RFS is data-dependent. To ensure that the model does not overfit as new users or items are included in the training data, we need to compare the number of parameters to the number of datapoints. A model with parameters the size of the training data can overfit by memorizing the training data. For generalization to be possible, overfitting can be avoided if the number of parameters grows slower than the size of the data. The technical backing for this comes from asymptotic statistics and the concept of sieved likelihoods. Specifically, the maximum likelihood estimation procedure with the objective function in Equation (7) can be replaced by maximization of a sieved likelihood function. The 'sieve' refers to filtering information as the number of parameters (in this case, user and item representations) grows with the number of observations. The sieved likelihood function enables the analysis of asymptotic behavior as the number of users grows $U \to \infty$ and the number of items grows $I \to \infty$, An example of a technique to grow the number of parameters in a way that supports generalization is given in Chapter 25 of Vaart [30].