Received: Added at production | Revised: Added at production | Accepted: Added at production

DOI: xxx/xxxx

2020 SDSS SPECIAL ISSUE

RankFromSets: Scalable Set Recommendation with Optimal Recall

Jaan Altosaar*[1] | Rajesh Ranganath[2] | Wesley Tansey[3]

[1]Columbia University

[2]New York University

[3]Memorial Sloan Kettering Cancer Center

Correspondence

*Corresponding author. Email: j@jaan.io

Present Address

Columbia University

Abstract

We study a variant of user-item recommendation 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 focus on the latter example, with the goal of building a meal recommender for a diet tracking app. Meal recommendation is challenging: (i) each item (meal) is rarely logged by more than a handful of users, (ii) the database of attributes (foods) is large, and (iii) each item is tagged with only a handful of attributes. We propose RankFromSets (RFS), a flexible and scalable class of models for recommending items with attributes. RFS treats item attributes as set-valued 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 RFS objective to optimal recall and show that the learnable class of models for RFS is a superset of several previously-proposed models. We then develop a stochastic optimization method for RFS that uses negative sampling to scale to massive problems like meal recommendation. In experiments on a real dataset of 55k users logging 16M meals, the new method outperforms competing approaches while learning embeddings that reveal interpretable structure in user behavior. Code is available at https://github.com/altosaar/rankfromsets.

KEYWORDS:

set-valued recommendation; food recommender systems; permutation-invariant models

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 a user consumed an item. Modern applications often gather rich side information about items in the form of a set of attributes or tags. Item attributes provide valuable side information for recommender systems. With a large number of items or a sparse user-item matrix, attribute information is necessary for good performance.

We are motivated by a specific dataset with these properties: a dataset of 55k users logging 16M meals using the LoseIt! diet tracking app. Table 1 shows the kind of data logged by users, where each row is an item (meal), each left-hand column is an attribute (food), and each right-hand column is a user. The food attributes can clearly inform recommendations: User 1 does not log meat, User 4 is omnivorous and undiscriminating, and User 3 mostly eats salads. In the LoseIt! data, there is a massive number of possible items to recommend: there are 12M unique meals, composed of

subsets of 3M foods. Meals containing only a few foods, or those ordered at chain restaurants, may be logged by many users. But these represent a small proportion of the meals people actually eat, so a long tail of meals are logged by single users.

Modeling item attributes in these non-standard recommender systems is not straightforward. Popular ways to use item attributes like multiple matrix factorization (Gopalan et al., 2014; Wang & Blei, 2011) struggle when the attribute vocabulary or number of items is large. Conversely, simple models are computationally tractable but risk losing the ability to capture nonlinear patterns of user consumption. For instance, a user may enjoy meals tagged with foods `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 may not be clear how its training procedure connects to the recommender system evaluation metric. A matrix factorization method might minimize mean squared error when the recommender system is evaluated on recall. While it is plausible that minimizing mean squared error will improve recall, the connection between the two is implicitly assumed in many methods. Ideally, a recommender system should have an objective that matches its evaluation metric.

This paper proposes RFS, a class of principled, scalable models for recommending items with sets of attributes. RFS casts the recommendation problem as binary classification. Given a user and an item, 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, we develop an RFS method that is trained using negative sampling of random 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: we show that a classifier with zero worst-case error achieves maximum recall. Second, RFS is provably flexible enough to learn any class of recommendation model based on set-valued side information (including multiple matrix factorization). This generality makes RFS a natural drop-in replacement for many specialized models in the literature.

We study the performance of the negative sampling RFS model on a semi-synthetic benchmark dataset and the LoseIt! dataset. The semi-synthetic paper recommendation dataset consists of 65k users clicking on 636k papers posted to the arXiv; the attributes of each paper are the unique words in its abstract. We then apply the method to the LoseIt! dataset to make out-of-sample meal recommendations. In both cases, the RFS method outperforms the state of the art in terms of recall. In addition to good performance, the RFS model learns interpretable embeddings that intuitively capture the structure of the underlying data.

2 RankFromSets

RankFromSets (RFS) is a class of recommendation models that recommend items with attributes to users. Let $u \in \{1, \dots, N\}$ be a user, $m \in \{1, \dots, M\}$ 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. The observed data is a collection of user-item interactions $\{(u, m)\}$ and the sets of attributes associated with items $\{x_m\}$.

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 $\mathbf{r}_u = (r_{u1}, \dots, r_{uK})$ for each user. The goal of the recommendation task in this paper is to maximize the expected Recall@K,

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

with the expectation over users in the empirical distribution $\mathcal{D}$.

We combine three techniques to maximize Recall@K with RFS. 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 using a stochastic optimization-based negative sampling training procedure.

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

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

where x_m is the set of attributes of item m and σ is the sigmoid function. Recommendations made by RFS are the maximum likelihood set formed by ranking a set of items for a user according to the model $f(u, x_m)$. We motivate treating recommendation as classification with the following observation.

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

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

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, assigning greater probability to data with positive labels than to data with negative labels. In other words, it ranks positive examples above negative examples. Recall@K is measured by the fraction of items with positive labels 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. For a dataset for which a perfect (zero worst-case error) classifier exists, a consistent method for learning a classifier will be a consistent method for learning a recommendation system that targets expected recall. Put another way, recall is inherently binary: a model does or does not recall an item; an item is or is not in the top K recommendations in the numerator of Equation (1). So the best one can hope to do if recall is used to assess recommendation performance is to train a binary classifier. 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.

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

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

Each element in the inner product regression function in Equation (2) 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 $g(x_m) \in \mathbb{R}^d$ represents 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 scalar item intercept function $h(x_m) \in \mathbb{R}$ makes an item more or less likely due to availability.

To define scalable item embedding and item intercept functions, note that 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, a scalable parameterization of item embeddings g can be defined 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 (2) has several benefits. It requires computing a sum over only the attributes with which each item is associated. This enables RFS to scale to large attribute vocabularies where traditional matrix factorization methods are intractable. Second, the embed-and-average approach to set modeling is provably flexible as we show later. We now describe deep variants of RFS and detail how RFS can approximate other recommendation models.

The RFS inner product regression function in Equation (2) 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 may outperform another, or prove insufficient to capture patterns in user consumption. (Later we describe a more general class of models which can, in theory, achieve the same performance.) First, as an alternative to the log-bilinear model in Equation (2), we can use a deep neural network as a regression function:

$$f\left(u, x_m\right) = \phi \left(\theta_u, \frac{1}{|x_m|} \sum_{j \in x_m} \beta_j, g(x_m) \right) + h(x_m) , \tag{3}$$

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 (2); *ex ante*, it is unclear whether a finite-depth, finite-width neural network can represent the inner product.

Another regression function for RFS is a combination of Equations (2) and (3), using an idea borrowed from deep residual networks for image classification (K. He et al., 2016). In this architecture, a neural network ϕ with the same inputs as in Equation (3) learns the residual of the inner product model:

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

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 S1.4, and motivate exploring other architectures than the three examples here.

Stepping 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. We show that any RFS architecture is sufficiently flexible to approximate recommendation models that operate on set-valued input. We define permutation-invariant models before deriving this result.

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; the set of attributes associated with an item enter into Equations (2) to (4) via summation. Other examples of permutation-invariant recommendation models are multiple matrix factorization, models based on word embeddings, and permutation-marginalized recurrent neural networks. These models are shown to be permutation-invariant in Section 3 and evaluated in Section 4. We now show that RFS can approximate other permutation-invariant recommendation models such as matrix factorization.

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

The proof follows directly from Theorem 2 in Zaheer et al. (2017) and we will not restate it here. (The only change to the proof is the mapping from set elements to one-hot vectors, $c \colon V \to \{0, 1\}^{|V|}$ to yield a unique representation of every object in the powerset.) Proposition 2 means that any of the parameterizations in Equations (2) to (4) is flexible enough to approximate other principled recommendation models that leverage item attributes, such as multiple matrix factorization (Gopalan et al., 2014; Wang & Blei, 2011).

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

$$\mathcal{L}(\gamma, \lambda_u) = \mathbb{E}_u \left[\mathbb{E}_{m \sim \mathcal{D}_u \mid y_{um}=1} \left[\log p(y_{um} = 1 \mid x_m; \gamma) \right] + \lambda_u \mathbb{E}_{k \sim \mathcal{D}_u \mid y_{uk}=0} \left[\log p(y_{uk} = 0 \mid x_k; \gamma) \right] \right] \tag{5}$$

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@K 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 (5), and describe two negative sampling schemes that are dependent on the choice of evaluation metric.

Negative samples can be drawn uniformly over the entire corpus of items, which we define to be corpus sampling. If the item set is large, this can be an expensive procedure. This negative sampling scheme leads to objective functions used in other recommender systems (X. He et al., 2017; Song et al., 2018).

On large datasets, it is infeasible to calculate Recall@K for evaluation, as this requires ranking every item for every user (e.g. in Section 4 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.

Sampled recall is defined as follows. Consider 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 $r_{u1}, \ldots, r_{uK}$. SampledRecall@k is the fraction of the K held-out datapoints that the model ranks in the top k:

$$\text{SampledRecall@}k = \frac{1}{K}\mathbb{E}_{um}\left[\sum_{r\in\{\mathbf{r}_{u1},\dots\mathbf{r}_{uk}\}} y_{ur}\right]. \tag{6}$$

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, SampledRecall@k requires ranking only 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. This metric is used in recommender systems when the number of items is large (Ebesu et al., 2018; Yang et al., 2018).

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 negative sampling procedures above have the advantage of implicitly balancing the classifier. As shown in Veitch et al. (2019), using stochastic gradient descent with negative sampling is equivalent to a Monte Carlo approximation of the reweighted (balanced) classification loss.

3 Permutation-invariant Recommender Models

Proposition 2 shows that RFS can approximate permutation-invariant recommendation models. We describe several common recommendation models and show that they are permutation-invariant, before comparing their performance to RFS in Section 4.

Gopalan et al. (2014) develop a probabilistic matrix factorization model of user consumption data. Collaborative topic Poisson factorization (CTPF) models user preferences using a generative process,

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 $w_{dv} \sim \text{Poisson}(\theta_d^\intercal \beta_v)$.
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 $r_{ud} \sim \text{Poisson}(\eta_u^\intercal(\theta_d + \epsilon_d))$.

To show that CTPF 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:

$$p(w_d \mid \theta_d, \beta_v) = \prod_{w_{dv}\in w_d} p(w_{dv} \mid \theta_d, \beta_v)\,. \tag{7}$$

CTPF makes predictions 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, and therefore CTPF is permutation-invariant.

Word embedding models (Mikolov et al., 2013) can be used as recommendation models if the embeddings are learned using a modified context window. For an item with attributes x_m, let the context window for attribute $j \in x_m$ be the set of other attributes of the same item $j' \in x_m : j' \neq j$. To recommend items using this model of attributes 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, and recommendation is performed using the cosine similarity of user and item embeddings. This is a permutation-invariant model, as the output of the model depends on the sum of attribute embeddings (summation is invariant to permutation).

StarSpace is also an embedding model and represents users as a sum over a user's consumed items' attribute embeddings (there is no explicit user embedding). In contrast to the word embedding model, StarSpace is trained on a classification objective with negative samples drawn from

the the set of items (Wu et al., 2018). As model predictions depend on sums of attributes, StarSpace is a permutation-invariant recommendation model.

We next consider LightFM (Kula, 2015), a permutation-invariant recommendation model. We show that if the Bayesian Personalized Ranking (BPR) objective (Rendle et al., 2009) is used, LightFM is an instance of RFS. [1] Although the BPR objective is designed for ranking, models trained with it can be used to construct classifiers. The BPR objective is

$$\log \sigma \left(f(u, x_m; \gamma) - f(u, x_k; \gamma) \right),$$

where m corresponds to a positive label $y_{um} = 1$, k corresponds to a negative label $y_{uk} = 0$, and f is parameterized as in RFS (Kula, 2015). A ranking function f optimizes the BPR objective if $f \to \infty$ for the positive example and f is constant for the negative example; or, if f is constant for the positive example and $f \to -\infty$ for the negative example. In either case, a constant can be added to yield a perfect classifier from the ranking function f (positive examples are ranked higher than negative examples in the optimal ranking, so there exists such a constant). That we can construct a classifier from the BPR objective means that Proposition 1 applies: permutation-invariant models such as LightFM, trained with the BPR objective, are instances of the RFS class of recommendation models.

The regression function f in RFS can also be parameterized using a recurrent network, as in T. Bansal et al. (2016). Such a recommendation model can be made permutation-invariant if averaged over permutations of attributes fed to the network. Attributes are treated as a sequence and the marginalization is over these permutations,

$$p(y_{um} = 1 \mid x_m) = \frac{1}{|\pi(x_m)|} \sum_{\pi \in \pi(x_m)} \sigma \left(\phi(\theta_u, \{\beta_{\pi(1)}, \ldots, \beta_{\pi(J)}\}) \right) . \tag{8}$$

Here β_j are attribute embeddings, $\pi(x_m)$ denotes the set of all permutations of the attributes x_m, and ϕ is the output of a recurrent neural network architecture (T. Bansal et al., 2016) projected to a scalar.

4 Empirical Study

We study RFS on two datasets and tasks. The first data consists of researcher reading behavior from the arXiv; the semi-synthetic 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, models in the RFS class outperform several baseline methods. The permutation-invariant models we compare to are described in Section 3, and the hyperparameters used are described in Appendix S1.1. To show the relative ease of implementation of RFS we give example code in Appendix S1.5.[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. This is a semi-synthetic task: it uses real-world data, but the item side information (article abstracts) is not set-valued. Nevertheless, document recommendation is a standard benchmark to study whether RFS performs well in settings outside its target purview of meal recommendation. The arXiv data represents one year of usage (2012) and consists of 65k users, 636k preprints, and 7.6M clicks. For evaluation, we match (Gopalan et al., 2014), using the same test and validation splits and the same set of held-out 10k users. As in (Gopalan et al., 2014) we compute precision in addition to recall. The held-out validation and test splits each consist of 20% of the clicks 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.

Figure 2 shows that models in the RFS class outperform others. RFS with the inner product parameterization or LightFM with the BPR objective have identical performance (as we showed, the BPR objective yields a classifier equivalent to RFS). These RFS models outperform CTPF in terms of in-matrix recall by over 90%. RFS models also improve over 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 comparably to CTPF in terms of recall, and performs worse in terms of precision. Recurrent neural network recommendation models were implemented following T. Bansal et al. (2016) and given access to full sequence information, unlike RFS. (The permutation-marginalized version of these models in Equation (8) is evaluated on meal recommendation where the order of foods in a meal does not carry information.) The training details for the recurrent neural networks are in Appendix S1.1, but their performance was an order of magnitude worse than the other methods and these results are omitted. The RFS regression function used is in Equation (2); the other parameterizations did not fit in GPU memory.

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

[1] The LightFM paper (Kula, 2015) uses a logistic objective to which Proposition 1 applies. LightFM with the BPR objective is unpublished but implemented in code released by the author. For completeness, we studied LightFM with both objectives to ensure its performance is equivalent to RFS when the BPR objective is used.

[2] Full source code is available at https://github.com/altosaar/rankfromsets for reproducibility.

techniques. Machine learning researchers (`stat.ML` readers) neighbor statisticians (`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.

This experiment in recommending research papers also highlights a trade-off in computational budget and desired performance in recommender systems. As described in Appendix S1.2, the recurrent neural network recommendation models in T. Bansal et al. (2016) did not perform well with the computational budget allocated for all methods (one day of compute on Tesla P100 GPUs). In further experiments, the performance improved marginally with a larger computational budget of several days. Further research in this domain might compare to transformer models (Devlin et al., 2019; Vaswani et al., 2017). Transformers preserve sequence information, unlike RFS, although they require large computational budgets to make accurate predictions. This means transformer-based methods may present a different trade-off in recommendation performance than the recurrent neural networks we evaluated in this task.

Recommending Meals. We evaluate RFS on data collected from the LoseIt! diet 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. To preprocess, 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. For evaluation, 1% of the items (meals) are held out for evaluating validation and test performance respectively. We evaluate models using SampledRecall@K with $K = 10$.

Figure 3 shows the sampled recall: models in the RFS class outperform others, such as permutation-marginalized recurrent neural networks and word embedding models. The residual RFS model outperforms the RFS inner product parameterization (and the equivalent LightFM model trained on the BPR objective). The code released with Gopalan et al. (2014) or Wang and Blei (2011) did not scale to this size of data, despite sufficient computing resources. This experiment further verifies Proposition 1: RFS models can maximize recall.

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 Related Work

We survey food recommender systems and recommendation models, focusing on models that leverage content information and scale to large numbers of users, items, and attributes.

Existing food recommendation systems focus on healthy recommendation (Freyne et al., 2011; Khan et al., 2019; Trattner & Elsweiler, 2019a; Yang et al., 2017), while RFS focuses on the scalability challenge of meal recommendation. After training a recommendation model, it is possible to filter the recommendations by nutritional information to nudge users towards healthier eating habits (Elsweiler et al., 2017); such approaches can be used to include nutritional information into RFS recommendations. When data is used in food recommender systems, it is usually recipe data (Trattner & Elsweiler, 2018); RFS is designed to recommend meals using crowdsourced food consumption data which may accurately reflect user behavior (Trattner & Elsweiler, 2019b).

We highlight several themes in research on recommendation models. We describe recommendation models that incorporate side information, models that recommend through classification, and models that optimize proxies of ranking metrics. This related work is summarized in Table 3. We focus on deep learning-based and matrix factorization methods to include side information in recommendation models. Item side information can be modeled with deep representations or can be included in content-based matrix factorization models as an additional matrix. Some deep learning approaches scale to large datasets, but may not have objective functions tied to evaluation metrics, or may require data beyond user-item interactions (Okura et al., 2017). Content-based matrix factorization methods require learning parameters for every item, and do not scale to data with large numbers of items (Gopalan et al., 2014; Wang & Blei, 2011), whereas RFS scales and is tied to evaluation.

Deep Representations of Side Information. Deep learning-based recommendation models incorporate side information in multiple ways (S. Zhang et al., 2019). For example, items that have words as attributes can be represented using neural networks (T. Bansal et al., 2016; Y. Chen & de Rijke, 2018) or embeddings (Wu et al., 2018). RFS uses both embeddings and deep learning techniques such as residual networks (K. He et al., 2016) to include side information. Lian et al. (2018) use an attention mechanism to weight recommendations according to available item and user side information, and Dong et al. (2017) use denoising autoencoders to model side information in a deep recommendation model, but these methods require fitting parameters for every item and hence cannot scale. An example of a more efficient approach is the method in T. Chen et al. (2017), where embeddings are jointly learned for users, items, and item text for recommendation, but this method focuses on unsupervised pre-training of text representations. RFS is complementary to such approaches, as the user, attribute, and item embeddings can be initialized using pre-training.

Deep structured semantic models are designed for document retrieval given query words (Huang et al., 2013; Palangi et al., 2016); it is unclear how to use this setup for recommending items with set-valued side information to users. There are several examples of 'tag-aware' or 'tag-based' deep recommendation models (N. Liang et al., 2018; Zuo et al., 2016), such as Xu et al. (2017), which focuses on data where users and items have different attributes and uses autoencoders to learn user, item, and attribute representations. Xu et al. (2017) uses a cosine similarity-based objective function which is not tied to a metric used to evaluate recommendation performance, whereas RFS is tied to recall as shown in Proposition 1.

Recommendation via Classification. The framing of recommendation as classification has been around for a long time (Basu et al., 1998), and several works build deep learning-based classifiers for recommendation (Cheng et al., 2016; Covington et al., 2016; Guo et al., 2017; X. He et al., 2017). Covington et al. (2016) focus on scalable inclusion of user and item attributes for video recommendation, Cheng et al. (2016) jointly train generalized linear models and deep neural networks for recommendation, while Guo et al. (2017) use factorization machines to learn high- and low-order interactions of features. Our work is complementary to these approaches: RFS focuses on scalable inclusion of set-valued side information, and provides theoretical undergirding to these recommendation models. We connect such models that rely on classification to optimal recall in Proposition 1. And if a specific architecture developed in these works is a permutation-invariant recommendation model, we proved that RFS is a universal function approximator (Proposition 2). So if performance is measured by recall, an RFS model can converge to an optimal recommender.

Matrix Factorization with Side Information. While matrix factorization methods perform well in recommending items that have consumption data in the training set (Hu et al., 2008; D. Liang et al., 2016), they cannot recommend items that have not been consumed in the training data. Including side information in matrix factorization enables recommendation of these items with no consumption data. Shi et al. (2014) survey several matrix factorization methods that leverage side information. Gopalan et al. (2014) develop a Bayesian matrix factorization model for recommending items based on side information in the form of words in documents, and we compare RFS to this method in Section 4. Wang and Blei (2011) develop a regression model that uses a topic model to incorporate side information into recommendations. There are also several 'tag-based' or 'tag-aware' content-based matrix factorization models (Bogers, 2018; Loepp et al., 2019; Zhen et al., 2009). Such content-based matrix factorization methods maximize the conditional log-likelihood of the data (or a bound on the log-likelihood); optimizing these objective functions may not optimize an evaluation metric. These methods are not scalable to large numbers of items as they require learning unique parameters for every item. Specifically, such content-based matrix factorization methods require learning a matrix that has a row for every item. For items with attributes, it is often infeasible to store this matrix in memory or exploit efficient coordinate ascent optimization schemes that require processing this entire matrix. RFS, however, is designed to scale to tens of millions of items, as we demonstrate empirically in Section 4.

Learning to Rank. The learning to rank literature includes several recommendation models trained on objectives that approximate ranking-based evaluation metrics (J. Liang et al., 2018; Rendle et al., 2009; Song et al., 2018; Yu et al., 2018), and some of these models include side information (Cao et al., 2017; Okura et al., 2017; Shi, Karatzoglou, Baltrunas, Larson, Hanjalic, et al., 2012; Shi, Karatzoglou, Baltrunas, Larson, Oliver, et al., 2012; Ying et al., 2016; Yuan et al., 2016). Such approaches can require data in addition to the user-item matrix, such as per-item parameters, or might use models whose output depends on the ordering of item attributes (making them infeasible for set-valued side information). In Section 4, we show that the ranking-based BPR objective function (Kula, 2015; Rendle et al., 2009) is in the RFS class, so Proposition 1 can help frame this related work. Li et al. (2016) use an objective that is in the same class as BPR, and other work bounds the BPR objective (Y. Zhang et al., 2018); these are also examples of RFS models if a permutation-invariant architecture is specified and we study one such choice (Kula, 2015) in Section 4.

6 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 (for example, content-based matrix factorization) are not directly tied to evaluation metrics. We developed RFS, a class of scalable recommendation models for items with attributes. Theoretically, we showed that optimizing the RFS objective optimizes recall, and that RFS can approximate permutation-invariant recommendation models including content-based matrix factorization. Empirically, models in the RFS class outperform competing models and scale to large datasets, such as our motivating problem of meal recommendation for 55k users who consume 16M meals.

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. For generalization theory, we conjecture that a different loss function should allow a similar proof to Proposition 1. With sufficient data, RFS can learn arbitrary distributions of users consuming items with attributes. But performance on finite data can vary, and developing generalization theory for RFS remains an open question.

Finally, R. Bansal et al. (2020) compare RFS to a state-of-the-art model, BERT (Devlin et al., 2019), and find that it is faster to train and performs better on a document recommendation task. Characterizing why RFS performs better than BERT in terms of recall and computational speed may yield further improvements in future work.

How to cite this article: Altosaar J., R. Ranganath, and W. Tansey, (2020), RankFromSets: Scalable Set Recommendation with Optimal Recall , *Under review*, *2020;00:1–6*.

References

Bansal, R., Olmstead, J., Bram, U., Cottrell, R., Reder, G., & Altosaar, J. (2020). Recommending interesting writing using a controllable, explanation-aware visual interface. Workshop on Interfaces and Human Decision Making for Recommender Systems, ACM Recommender Systems.

Bansal, T., Belanger, D., & McCallum, A. (2016). Ask the GRU: Multi-task Learning for Deep Text Recommendations. ACM Recommender Systems.

Basu, C., Hirsh, H., & Cohen, W. (1998). Recommendation as classification: Using social and content-based information in recommendation. AAAI Conference on Artificial Intelligence.

Bogers, T. (2018). Social information access. Social information access: Systems and technologies.

Cao, D., Nie, L., He, X., Wei, X., Zhu, S., & Chua, T.-S. (2017). Embedding factorization models for jointly recommending items and user generated lists. ACM SIGIR Conference on Research and Development in Information Retrieval.

Chen, T., Hong, L., Shi, Y., & Sun, Y. (2017). Joint text embedding for personalized content-based recommendation. arXiv:1706.01084.

Chen, Y., & de Rijke, M. (2018). A collective variational autoencoder for top-n recommendation with side information. ACM Workshop on Deep Learning for Recommender Systems.

Cheng, H.-T., Koc, L., Harmsen, J., Shaked, T., Chandra, T., Aradhye, H., Anderson, G., Corrado, G., Chai, W., Ispir, M., Anil, R., Haque, Z., Hong, L., Jain, V., Liu, X., & Shah, H. (2016). Wide & deep learning for recommender systems. ACM Workshop on Deep Learning for Recommender Systems.

Covington, P., Adams, J., & Sargin, E. (2016). Deep neural networks for youtube recommendations. ACM Conference on Recommender Systems.

Devlin, J., Chang, M.-W., Lee, K., & Toutanova, K. (2019). BERT: Pre-training of deep bidirectional transformers for language understanding. Association for Computational Linguistics.

Dong, X., Yu, L., Wu, Z., Sun, Y., Yuan, L., & Zhang, F. (2017). A hybrid collaborative filtering model with deep structure for recommender systems. AAAI Conference on Artificial Intelligence.

Ebesu, T., Shen, B., & Fang, Y. (2018). Collaborative memory network for recommendation systems. ACM Special Interest Group on Information Retrieval.

Elsweiler, D., Trattner, C., & Harvey, M. (2017). Exploiting food choice biases for healthier recipe recommendation. ACM SIGIR Conference on Research and Development in Information Retrieval.

Freyne, J., Berkovsky, S., & Smith, G. (2011). Recipe recommendation: Accuracy and reasoning. User modeling, adaption and personalization.

Gopalan, P., Charlin, L., & Blei, D. M. (2014). Content-based Recommendations with Poisson Factorization. Neural Information Processing Systems.

Guo, H., Tang, R., Ye, Y., Li, Z., & He, X. (2017). Deepfm: A factorization-machine based neural network for ctr prediction. International Joint Conference on Artificial Intelligence.

He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. IEEE Conference on Computer Vision and Pattern Recognition.

He, X., Liao, L., Zhang, H., Nie, L., Hu, X., & Chua, T.-S. (2017). Neural collaborative filtering. International World Wide Web Conference.

Hu, Y., Koren, Y., & Volinsky, C. (2008). Collaborative filtering for implicit feedback datasets. IEEE International Conference on Data Mining.

Huang, P.-S., He, X., Gao, J., Deng, L., Acero, A., & Heck, L. (2013). Learning deep structured semantic models for web search using clickthrough data. ACM International Conference on Information & Knowledge Management.

Khan, M. A., Rushe, E., Smyth, B., & Coyle, D. (2019). Personalized, health-aware recipe recommendation: An ensemble topic modeling based approach. ACM Conference on Recommender Systems, Workshop on Health Recommender Systems.

Kula, M. (2015). Metadata embeddings for user and item cold-start recommendations. arXiv:1507.08439.

Li, H., Hong, R., Lian, D., Wu, Z., Wang, M., & Ge, Y. (2016). A relaxed ranking-based factor model for recommender system from implicit feedback. International Joint Conference on Artificial Intelligence.

Lian, J., Zhang, F., Xie, X., & Sun, G. (2018). Towards better representation learning for personalized news recommendation: A multi-channel deep fusion approach. International Joint Conference on Artificial Intelligence.

Liang, D., Altosaar, J., Charlin, L., & Blei, D. M. Factorization meets the item embedding: Regularizing matrix factorization with item co-occurrence. In: Acm conference on recommender systems. 2016.

Liang, J., Hu, J., Dong, S., & Honavar, V. G. (2018). Top-n-rank: A scalable list-wise ranking method for recommender systems. arXiv:1812.04109.

Liang, N., Zheng, H.-T., Chen, J.-Y., Sangaiah, A., & Zhao, C.-Z. (2018). Trsdl: Tag-aware recommender system based on deep learning–intelligent computing systems. Applied Sciences.

Liu, Y., Xie, M., & Lakshmanan, L. V. (2014). Recommending user generated item lists. ACM Recommender Systems.

Loepp, B., Donkers, T., Kleemann, T., & Ziegler, J. (2019). Interactive recommending with tag-enhanced matrix factorization (tagmf). International Journal of Human-Computer Studies.

Maaten, L. v. d., & Hinton, G. (2008). Visualizing Data using t-SNE. Journal of Machine Learning Research.

Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013). Distributed Representations of Words and Phrases and their Compositionality. Neural Information Processing Systems.

Okura, S., Tagami, Y., Ono, S., & Tajima, A. (2017). Embedding-based news recommendation for millions of users. ACM Knowledge Discovery and Data Mining.

Palangi, H., Deng, L., Shen, Y., Gao, J., He, X., Chen, J., Song, X., & Ward, R. (2016). Deep sentence embedding using long short-term memory networks: Analysis and application to information retrieval. IEEE/ACM Transactions on Audio, Speech and Language Processing, (4).

Rendle, S., Freudenthaler, C., Gantner, Z., & Schmidt-Thieme, L. (2009). Bpr: Bayesian personalized ranking from implicit feedback. Uncertainty in Artificial Intelligence.

Shi, Y., Karatzoglou, A., Baltrunas, L., Larson, M., Hanjalic, A., & Oliver, N. (2012). Tfmap: Optimizing map for top-n context-aware recommendation. ACM SIGIR Conference on Research and Development in Information Retrieval.

Shi, Y., Karatzoglou, A., Baltrunas, L., Larson, M., Oliver, N., & Hanjalic, A. (2012). Climf: Learning to maximize reciprocal rank with collaborative less-is-more filtering. ACM Recommender Systems.

Shi, Y., Larson, M., & Hanjalic, A. (2014). Collaborative filtering beyond the user-item matrix: A survey of the state of the art and future challenges. ACM Computing Surveys, (1).

Song, B., Yang, X., Cao, Y., & Xu, C. (2018). Neural collaborative ranking. ACM Conference on Information and Knowledge Management.

Trattner, C., & Elsweiler, D. (2018). Food recommender systems: Important contributions, challenges and future research directions. Collaborative recommendations: Algorithms, practical challenges and applications.

Trattner, C., & Elsweiler, D. (2019a). An evaluation of recommendation algorithms for online recipe portals. ACM Conference on Recommender Systems, Workshop on Health Recommender Systems.

Trattner, C., & Elsweiler, D. (2019b). What online data say about eating habits. Nature Sustainability, (7).

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser,., & Polosukhin, I. (2017). Attention is all you need. Neural Information Processing Systems.

Veitch, V., Austern, M., Zhou, W., Blei, D. M., & Orbanz, P. (2019). Empirical risk minimization and stochastic gradient descent for relational data. International Conference on Artificial Intelligence and Statistics.

Wang, C., & Blei, D. M. (2011). Collaborative topic modeling for recommending scientific articles. ACM Knowledge Discovery and Data Mining.

Wu, L., Fisch, A., Chopra, S., Adams, K., Bordes, A., & Weston, J. (2018). Starspace: Embed all the things! AAAI Conference on Artificial Intelligence.

Xu, Z., Lukasiewicz, T., Chen, C., Miao, Y., & Meng, X. (2017). Tag-aware personalized recommendation using a hybrid deep model. International Joint Conference on Artificial Intelligence.

Yang, L., Bagdasaryan, E., Gruenstein, J., Hsieh, C.-K., & Estrin, D. (2018). Openrec: A modular framework for extensible and adaptable recommendation algorithms. ACM International Conference on Web Search and Data Mining.

Yang, L., Hsieh, C.-K., Yang, H., Pollak, J. P., Dell, N., Belongie, S., Cole, C., & Estrin, D. (2017). Yum-me: A personalized nutrient-based meal recommender system. ACM Transactions on Information Systems, (1).

Ying, H., Chen, L., Xiong, Y., & Wu, J. (2016). Collaborative deep ranking: A hybrid pair-wise recommendation algorithm with implicit feedback. Pacific-Asia Conference on Knowledge Discovery and Data Mining.

Yu, L., Zhang, C., Pei, S., Sun, G., & Zhang, X. (2018). Walkranker: A unified pairwise ranking model with multiple relations for item recommendation. AAAI Conference on Artificial Intelligence.

Yuan, F., Guo, G., Jose, J. M., Chen, L., Yu, H., & Zhang, W. (2016). Optimizing factorization machines for top-n context-aware recommendations. Web Information Systems Engineering.

Zaheer, M., Kottur, S., Ravanbakhsh, S., Póczos, B., Salakhutdinov, R., & Smola, A. J. (2017). Deep sets. Neural Information Processing Systems.

Zhang, S., Yao, L., Sun, A., & Tay, Y. (2019). Deep learning based recommender system: A survey and new perspectives. ACM Computing Surveys, (1).

Zhang, Y., Wang, H., Lian, D., Tsang, I. W., Yin, H., & Yang, G. (2018). Discrete ranking-based matrix factorization with self-paced learning. ACM Knowledge Discovery and Data Mining.

Zhen, Y., Li, W.-J., & Yeung, D.-Y. (2009). Tagicofi: Tag informed collaborative filtering. ACM Recommender Systems.

Zuo, Y., Zeng, J., Gong, M., & Jiao, L. (2016). Tag-aware recommender systems based on deep neural networks. Neurocomputing.

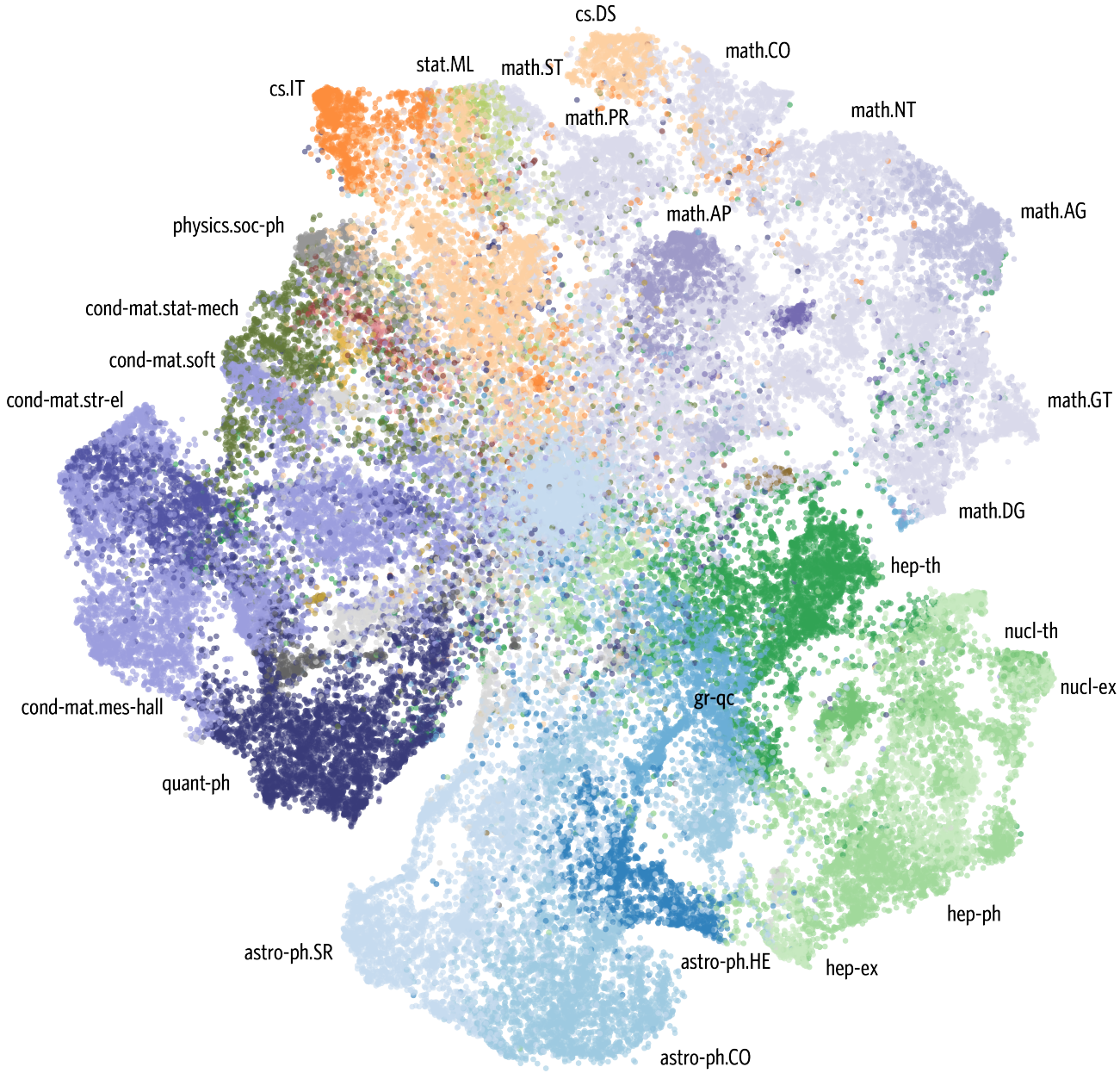
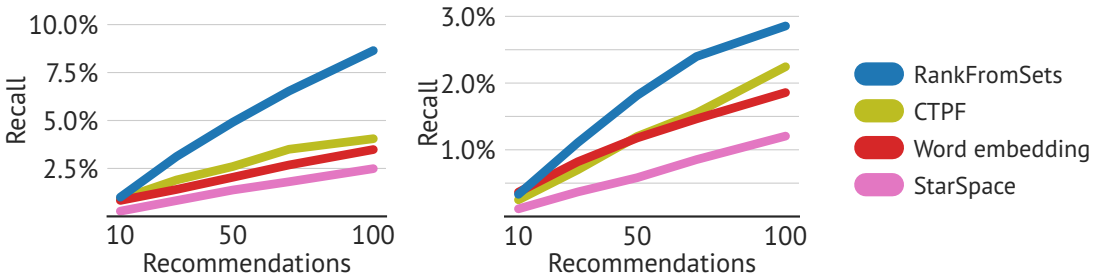
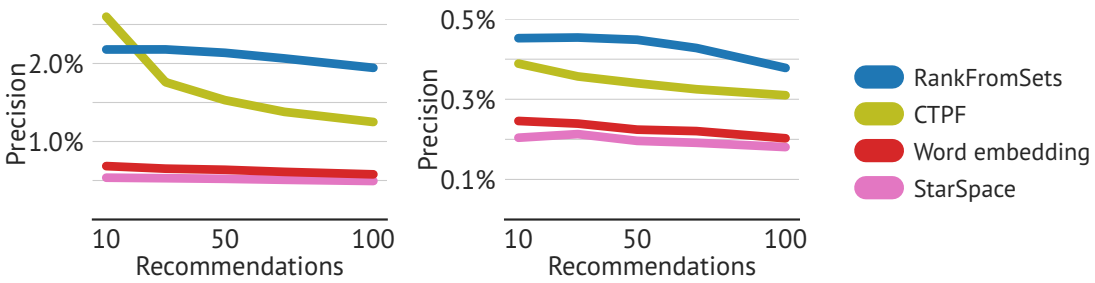


Figure 1 RankFromSets trained on arXiv reading behavior clusters researchers by their most frequently-read arXiv category (best viewed on a screen). RFS is trained to recommend items using their attributes (words in the abstract). t-SNE (Maaten & Hinton, 2008) is used to visualize the user embeddings θ_u in the inner product regression function in Equation (2). Each marker represents a user embedding; its color represents a user's most-read arXiv category. Unique colors are determined using the most-read categories across the arXiv, and colors are assigned according to the arXiv ontology. RFS captures usage patterns, as fields of study are related by patterns of reading behavior across neighboring fields (e.g. `stat.ML` and `cs.IT`). An interactive map with all labels and zoom can be accessed at https://jaan.io/files/rankfromsets-arxiv.html.



(a) Recall for in-matrix (left) and out-matrix (right) documents.



(b) Precision for in-matrix (left) and out-matrix (right) documents.

Figure 2 RankFromSets outperforms collaborative topic Poisson factorization (CTPF) (Gopalan et al., 2014) and other models on recommending arXiv papers to scientists. The items are documents and the attributes are the unique words in the abstracts. Recommendation performance is evaluated using both precision and recall to match the evaluation in (Gopalan et al., 2014). The metrics are reported on training (in-matrix) documents and cold-start (out-matrix) documents with no clicks in the training set. All GRU and LSTM-based models in T. Bansal et al. (2016) performed an order of magnitude worse, and these results are omitted (training details are in Appendix S1.1).

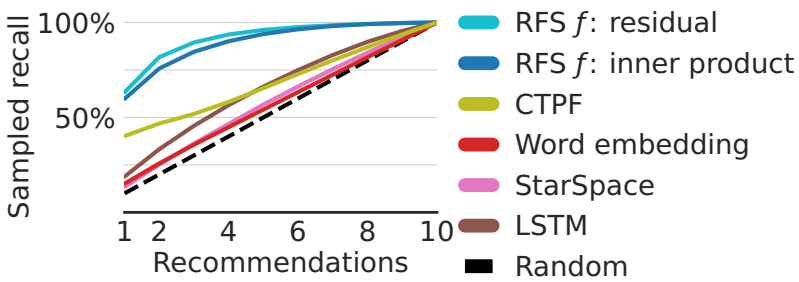


Figure 3 RankFromSets models outperform competitors in meal recommendation in terms of sampled recall computed using Equation (6). Comparison models are described in Section 3 (see Appendix S1.1 for hyperparameters). The RFS regression functions f are defined in Equations (2) to (4) for the inner product, neural network, and residual models, respectively.

	Attributes									Users				
Items	Pizza	Eggs	Taco	Salad	Avocado	Chicken	Sardines	Beer	Coffee	1	2	3	4	5
Morning Pizza	●	●							●		●		●	
Dinner Pizza	●						●	●		●			●	
Small Salad				●	●		●					●		●
Big Salad		●		●	●	●			●			●	●	
Taco			●		●	●			●					●
Fish Taco			●				●				●		●	

Table 1 An example of the data we focus on, where tagged items are recommended to users based on both item attributes and items users have consumed in the past. This example dataset of meals contains meals with different foods (left) and users log which meals they ate (right). The goal is to leverage the attributes to recommend items to users.

Query Item	Nearest Item 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 meal recommendations. RFS with Equation (4) is fit to data from a diet tracking app; items are meals and attributes are the ingredients in the meal. Meals are represented the average of their attribute embeddings, and cosine similarity between meal representations is used to find the nearest neighbors of meals (user-level information cannot be shown as this is personal diet data). RFS reveals eating patterns: for example, the second-last query meal is a mix of meat, vegetables, and fruit, and the nearest neighbor meal is a different meat with a side of salad; the last query meal is a sandwich, and its nearest neighbor is also a sandwich with different ingredients.

Model	Attributes	Implicit	Scalable	Invariant	Evaluation
RankFromSets	✔	✔	✔	✔	✔
CTPF, Gopalan et al. (2014)	✔	✔		✔	
StarSpace, Wu et al. (2018)	✔	✔	✔	✔	
LightFM, Kula (2015)	✔	✔	✔	✔	
BPR, Rendle et al. (2009)		✔			✔
Wang and Blei (2011)	✔	✔		✔	
Lian et al. (2018)			✔		
Dong et al. (2017)	✔			✔	
T. Chen et al. (2017)	✔		✔		
T. Bansal et al. (2016)	✔	✔			
Xu et al. (2017)	✔		✔	✔	
Shi, Karatzoglou, Baltrunas, Larson, Oliver, et al. (2012)		✔			✔
Y. Chen and de Rijke (2018)	✔	✔		✔	
Liu et al. (2014)	✔			✔	✔
Cao et al. (2017)	✔				✔
Okura et al. (2017)	✔		✔		✔

Table 3 RankFromSets recommends items using attributes, and is trained to maximize the evaluation metric of recall. Most methods we highlight leverage item attributes (Attributes); some require data in addition to the implicit feedback data of user-item interactions (Implicit). Few methods are scalable, as most models that use item side information require learning parameters for every item. Some models are invariant to permutation of the attributes (Invariant), and some enjoy a loss function that is connected to a recommender performance metric (Evaluation).

Appendix

S1 Appendix

S1.1 Empirical Study Hyperparameters

Experiments for RFS, LightFM, and recurrent neural network models are run on a cluster with Tesla P100 GPUs using PyTorch; all other experiments are performed on a 20-core computer.

Hyperparameters for Word Embedding Model and StarSpace. For the word embedding model and StarSpace, we use the software packages released alongside the respective papers (Bojanowski et al., 2017; Wu et al., 2018) with recommended hyperparameters, and grid search over embedding sizes of $\{128, 256, 512, 1024\}$ for both datasets.

Hyperparameters for LightFM. On both datasets, LightFM with the logistic objective reported in Kula (2015) performs poorly and we omit these results. Kula (2015) does not use the BPR objective in the paper; nevertheless, we study LightFM with the BPR objective (this variant is unpublished, yet implemented in the code released in (Kula, 2015)) and use the same hyperparameters as RFS for comparison.

S1.2 Recommending Research Papers

Hyperparameters for RFS. We test the stochastic gradient descent algorithm with and without momentum (Sutskever et al., 2013). 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; such corpus sampling is defined in Section 2. As the number of items is small relative to the larger diet tracking data, the item intercept function is simply a scalar for every item, and the item embedding function learns item embeddings. To match the hyperparameters in Gopalan et al. (2014), we set the dimensionality of user and item embeddings to 100. Evaluation is performed every 20k iterations.

Hyperparameters for Recurrent Neural Network Models. We implement the model in Bansal et al. (2016) using PyTorch and match the hyperparameters where possible. We test gated recurrent unit (GRU) cells and long short-term memory (LSTM) cells with the objective function in Equation (5). The model has access to full sequence information, unlike RFS, as abstracts of research papers have meaningful sequence information (marginalizing using Equation (8) would destroy information and decrease performance). The attribute embedding size is fixed to 100 to match the other models and attribute embeddings are initialized to word embeddings pretrained on all 636k document abstracts as in Bansal et al. (2016), using the word embedding implementation in Bojanowski et al. (2017). The first layer of the recurrent neural network is bidirectional and of hidden size 400, the second layer is unidirectional and of hidden size 200, and dropout is used with the same settings as in Bansal et al. (2016). Evaluation is performed every 20k iterations. We grid search over learning rates of $\{10^{-1}, 10^{-2}, 10^{-3}, 10^{-4}\}$ with the Adam optimizer (Kingma & Ba, 2015) and batch sizes of $\{64, 128, 256, 512, 1024, 4096, 8192\}$. As evaluation is much more expensive for sequence models, we randomly select a subset of 100 users from the held-out set of 10k users. If validation performance does not improve, we reload the best parameters and optimizer states, and divide the learning rate by half. In both experiments, LSTM cells outperformed GRU cells.

S1.3 Recommending Meals

Hyperparameters for RFS. The embedding size is set to 128. For the neural network and residual models in Equations (3) and (4) the number of hidden layers is two, and the number of hidden units is set to 256 with rectifier nonlinearities. The item embeddings $g(x_m)$, and item intercepts $h(x_m)$, are computed as the mean of learned food embeddings and intercepts, respectively. We use the RMSProp optimizer in Graves (2013) and grid search over the learning rates $\{10^{-2}, 10^{-3}, 10^{-4}, 10^{-5}\}$. We use a batch size of 64 and a single negative sample for every datapoint in a minibatch (batch sampling is defined in Section 2). Evaluation is performed every 50k iterations.

Hyperparameters for Permutation-marginalized Recurrent Neural Networks. We use the same settings as described in Appendix S1.2, but the data in this case has no sequence information so we use Equation (8) to average predictions of the model in Bansal et al. (2016) over permutations. Evaluation for sequence models is already prohibitive, so for every item in a minibatch we sample a single permutation of attributes to approximate the sum over permutations in Equation (8). We use a single negative sample per datapoint (minibatch sampling), and set the embedding and hidden state sizes to 128. We use the Adam optimizer (Kingma & Ba, 2015) and grid search over the same learning rates, learning rate decay, and batch sizes as in Appendix S1.2, with evaluation every 1k iterations.

S1.4 Generalization Simulation Study

Proposition 2 is a universal function approximation theorem in the regime of infinite data. With finite data and a finite number of parameters, the optimal parameterization of RFS is dependent on the data-generating distribution. From Figure 3, the inner product RFS parameterization outperforms the neural network parameterization. We demonstrate a simulated dataset where this order is reversed, to motivate the exploration of novel architectures. 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 (3) and (4) outperforming the inner product architecture in Equation (2) in terms of predictive performance. We will release all code required to replicate this experiment.

1. **For every user** u**:** Draw user embedding $\theta_{\mathsf{u}} \sim \text{Normal}(0, \mathbf{I})$.
2. **For every attribute** j**:** Draw attribute embedding $\beta_{\mathsf{j}} \sim \text{Normal}(0, \mathbf{I})$.
3. **For every item** m**:**
   (a) Draw item topics $\theta_{\mathsf{m}} \sim \text{Dirichlet}(\alpha)$
   (b) Draw number of item attributes $\mathsf{M} \sim \text{Poisson}(\lambda)$
   (c) Draw nonzero item attributes $\mathsf{x}_{\mathsf{m}} \sim \text{Multinomial}(\mathsf{M}, \theta_{\mathsf{m}})$.
4. **For every user, item:** $\mathsf{y}_{\mathsf{um}} \sim \text{Bernoulli}\left(\mathsf{y}_{\mathsf{um}}; \sigma(\mathsf{f}(\theta_{\mathsf{u}}, \mathsf{x}_{\mathsf{m}}))\right)$.

The logit function f is the square kernel:

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

The output of f is standardized across users and centered at 7 to achieve sparse user-item observations.

For this simulation study, 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. The embedding sizes are fixed to 100, and for parameterizations with neural networks two hidden layers are used with rectifier nonlinearities. The hidden size of models with neural networks is chosen so the total number of parameters matches the number of parameters in the inner product model. We fix the momentum to 0.9 (Sutskever et al., 2013) 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. We regenerate data 30 times. Results are averaged over these synthetic datasets, with the best hyperparameters for each model from the first instance.

The results in Table S1 demonstrate that the residual model outperforms both the deep and inner product architectures for data generated by the above generative process. This shows that the choice of architecture in RFS is data-dependent and leads to considering when a RFS recommendation model supports generalization. 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 (5) can be replaced by maximization of a sieved likelihood function. The 'sieve' refers to filtering information as the number of parameters (in this case, the number of parameters in 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 $\mathsf{U} \to \infty$ and the number of items grows $\mathsf{I} \to \infty$. An example of a technique to grow the number of parameters in a way that supports generalization is given in Chapter 25 of Vaart (1998).

S1.5 Code

In Figure S1 we give an example implementation of RankFromSets with the inner product regression function in Equation (2) in python with the PyTorch package. This implementation is easy to port to new applications and achieves state-of-the-art results in Section 4.

S1.6 Including Metadata to Improve Performance

RankFromSets enables leveraging additional information about users or items. For example, in meal recommendation there may be additional data about users such as `age`, `gender`, `height`. One way to build this into a model is to reuse the RFS regression function in Equation (2):

$$f(u, x_m, \texttt{age}, \texttt{gender}, \texttt{height}) = f^0(u, x_m) + f^1(u, \texttt{age}[u]) + f^2(u, \texttt{gender}[u]) + f^3(u, \texttt{height}[u])\,. \tag{A1}$$

This model, RFS-Metadata, is built using the components of RFS. The notation `age`[u] represents a lookup for the age of user u, and superscripts indicate the same function but with separate embeddings learned for every type of metadata. Alternatively, side information (categorical or continuous-valued) can be fed into the neural networks in Equations (3) and (4)—which architecture for the regression function performs best depends on the data at hand. Information about items (such as fat, carbohydrates, and protein) can be included in a similar fashion. Empirically, the RFS-Metadata model above improves recommendation performance as shown in Table S2.

References

Bansal, T., Belanger, D., & McCallum, A. (2016). Ask the GRU: Multi-task Learning for Deep Text Recommendations. ACM Recommender Systems.

Bojanowski, P., Grave, E., Joulin, A., & Mikolov, T. (2017). Enriching word vectors with subword information. Association for Computational Linguistics.

Gopalan, P., Charlin, L., & Blei, D. M. (2014). Content-based Recommendations with Poisson Factorization. Neural Information Processing Systems.

Graves, A. (2013). Generating sequences with recurrent neural networks. arXiv:1308.0850.

Kingma, D. P., & Ba, J. (2015). Adam: A method for stochastic optimization. International Conference on Learning Representations.

Kula, M. (2015). Metadata embeddings for user and item cold-start recommendations. arXiv:1507.08439.

Sutskever, I., Martens, J., Dahl, G., & Hinton, G. (2013). On the importance of initialization and momentum in deep learning. International Conference on Machine Learning.

Vaart, A. W. v. d. (1998). Asymptotic statistics.

Wu, L., Fisch, A., Chopra, S., Adams, K., Bordes, A., & Weston, J. (2018). Starspace: Embed all the things! AAAI Conference on Artificial Intelligence.

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

Table S1 A simulation study demonstrating that the choice of parameterization of RankFromSets is data-dependent. We report the in-matrix recall averaged over 100 users, over 30 replications of the simulation. The residual model in Equation (4) outperforms the deep model in Equation (3) and the inner product model in Equation (2).

```
import torch
import data
class InnerProd(torch.nn.Module):
  def __init__(self, n_users, n_items, n_attr, emb_size):
    super().__init__()
    self.user_embeddings = torch.nn.Embedding(n_users, emb_size)
    self.attribute_emb = torch.nn.EmbeddingBag(n_attr, emb_size)
    self.item_embeddings = torch.nn.Embedding(n_items, emb_size)
    self.intercepts = torch.nn.Embedding(n_items, 1)
  def forward(self, users, items, item_attributes, offsets):
    user_emb = self.user_embeddings(users)
    attr_emb = self.attribute_emb(item_attributes, offsets)
    item_emb = self.item_embeddings(items)
    logits = (user_emb * (attr_emb + item_emb)).sum(-1)
    return logits + self.intercepts(items).squeeze()
train = data.load_data(batch_size=2**16)
model = InnerProd(train.n_users, train.n_items, train.n_attr, 100)
optim = torch.optim.SGD(model.parameters(), learning_rate=15.0)
loss = torch.nn.BCEWithLogitsLoss()
# negative samples are in the last half of every batch
labels = (torch.arange(2**16) < (2**16 / 2)).float()
for batch in train:
  model.zero_grad()
  logits = model(*batch)
  L = loss(logits, labels)
  L.backward()
  optim.step()
```

Figure S1 RankFromSets is simple to implement. This shows the implementation for one epoch of training a RFS model in PyTorch.

Model	Sampled Recall (%)
RFS	62.2
RFS-Metadata	65.3

Table S2 RFS-Metadata improves over RFS by leveraging information in metadata in addition to item attributes in a meal recommendation task. The architecture for RFS-Metadata is in Equation (A1), and the RFS architecture is chosen to be Equation (2); the data are described in Section 4.