ProtoCF: Prototypical Collaborative Filtering for Few-shot Recommendation

Adrian Urbański

ProtoCF

ProtoCF: Prototypical Collaborative Filtering for Few-shot Recommendation

Aravind Sankar*
asankar3@illinois.edu
University of Illinois at Urbana-Champaign
Illinois, USA

Junting Wang*
junting3@illinois.edu
University of Illinois at Urbana-Champaign
Illinois, USA

Adit Krishnan
aditk2@illinois.edu
University of Illinois at Urbana-Champaign
Illinois, USA

Hari Sundaram
hs1@illinois.edu
University of Illinois at Urbana-Champaign
Illinois, USA

ABSTRACT

In recent times, deep learning methods have supplanted conventional collaborative filtering approaches as the backbone of modern recommender systems. However, their gains are skewed towards popular items with a drastic performance drop for the vast collection of *long-tail* items with sparse interactions. Moreover, we empirically show that prior neural recommenders lack the resolution power to accurately rank relevant items within the long-tail.

In this paper, we formulate long-tail item recommendations as a *few-shot learning* problem of *learning-to-recommend* few-shot items with very few interactions. We propose a novel *meta-learning* framework ProtoCF that learns-to-compose robust prototype representations for few-shot items. ProtoCF utilizes episodic few-shot learning to extract meta-knowledge across a collection of diverse meta-training tasks designed to mimic item ranking within the tail. To further enhance discriminative power, we propose a novel architecture-agnostic technique based on knowledge distillation to *extract, relate, and transfer* knowledge from neural base recom-

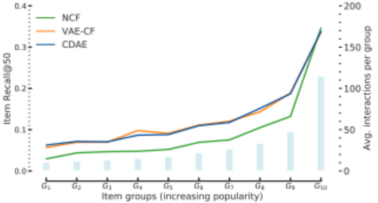


Figure 1: Item Recall@50 of three neural recommenders for item-groups (increasing popularity) in Epinions. Model performance is considerably lower for *long-tail* items.

are critical to diverse e-commerce applications. However, a close examination of neural recommenders' performance reveals a *paradox*:

Source: ProtoCF

Motivation

Strong bias of NCF methods towards popular items

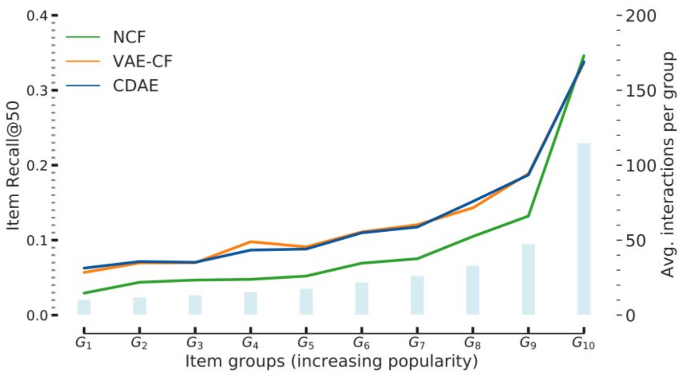


Figure 1: Item Recall@50 of three neural recommenders for item-groups (increasing popularity) in Epinions. Model performance is considerably lower for *long-tail* items.

Motivation

Lack of resolution power to accurately rank long-tail items

Item Subset	Top 50% Items		Bottom 50% Items	
Metric	N@50	R@50	N@50	R@50
NCF [12]	0.0906	0.1874	0.0352	0.0973
VAE-CF [24]	0.1055	0.2106	0.0457	0.1125
CDAE [51]	0.1050	0.2102	0.0471	0.1149

Table 1: Recommendation performance within top-50% *head* and bottom-50% *tail* items by item popularity on Epinions. R@50 and N@50 denote Recall@50 and NDCG@50 metrics. We observe poor ranking *resolution* within the long-tail.

Problems with Long-Tail Items

- sparsity and heterogeneity - tail items have few interactions, but belong to diverse item categories
- distribution mismatch - overall interaction distribution is biased towards head items

Proposed solution

- Few-shot learning to eliminate distribution mismatch
- Composition of discriminative prototypes for tail items
- Architecture-agnostic knowledge transfer from neural base recommender to enhance item prototypes

Neural Base Recommender

A neural base recommender $\mathbf{R}_B$ is trained to learn high-quality user representations and infer item-item relationships.

- X - interactions
- ϕ - model parameters

- $h_u = F_U(u, X \mid \phi)$ - user preference vector
- $h_i = F_I(i, X \mid \phi)$ - item preference vector
- $\hat{y}_b(u, i) = F_{INT}(h_u, h_i)$ - user-item relevance score
- $L_B = l(\hat{y}_b(u, i), y_{ui})$ - training objective obtained from a pointwise or pairwise loss function

Neural Base Recommender

- X - interactions
- ϕ - model parameters

A neural base recommender $\mathbf{R}_B$ is trained to learn high-quality user representations and infer item-item relationships.

- $h_u = F_U(u, X \mid \phi)$ - user preference vector
- $h_i = F_I(i, X \mid \phi)$ - item preference vector
- $\hat{y}_b(u, i) = F_{INT}(h_u, h_i)$ - user-item relevance score
- $L_B = l(\hat{y}_b(u, i), y_{ui})$ - training objective obtained from a pointwise or pairwise loss function

F_{INT} is usually modelled using inner product, however for the purposes of few-shot training cosine similarity is used.

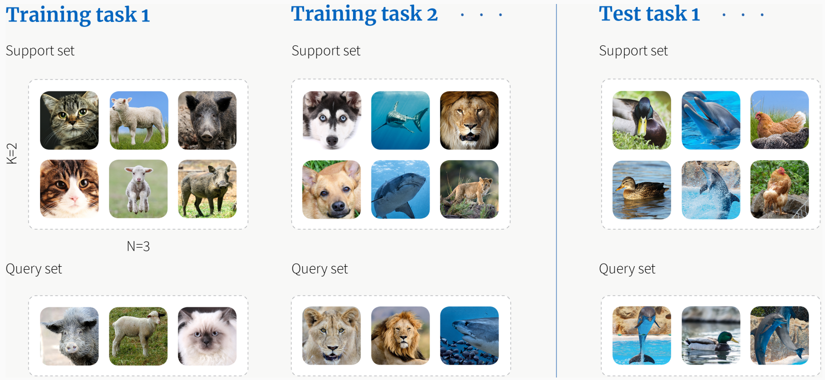
Neural Base Recommender

The authors considered three neural CF methods as base recommenders $\mathbf{R}_B$:

- Matrix Factorization (BPR)
- Variational AutoEncoder (VAE-CF)
- Denoising AutoEncoders (CDAE)

Few-Shot Learning

An example of an *N-way, K-shot* classification problem



Source: Borealis.ai

Collection of meta-training tasks $\{\mathcal{T}_1, \mathcal{T}_2, \dots\}$.

A *K-shot*, *N-item* training task $\mathcal{T} = \{\mathcal{I}_{\mathcal{T},N}, \mathcal{S}, \mathcal{Q}\}$ consists of:

- $\mathcal{I}_{\mathcal{T},N} \subset \mathcal{I}$ - a subset of items chosen for $\mathcal{T}$
- $\mathcal{S} = \{\mathcal{S}_i : i \in \mathcal{I}_{\mathcal{T},N}\}$ - a set of support user sets
- $\mathcal{Q} = \{\mathcal{Q}_i : i \in \mathcal{I}_{\mathcal{T},N}\}$ - a set of query user sets
- $\mathcal{S}_i = \{u_{i,1}, \dots, u_{i,K}\}$ - K users who interacted with item i
- $\mathcal{Q}_i = \{u'_{i,1}, \dots, u'_{i,K'}\}$ - K' users who interacted with item i

Typically $K \approx 5$ to 20

Learn-to-Recommend

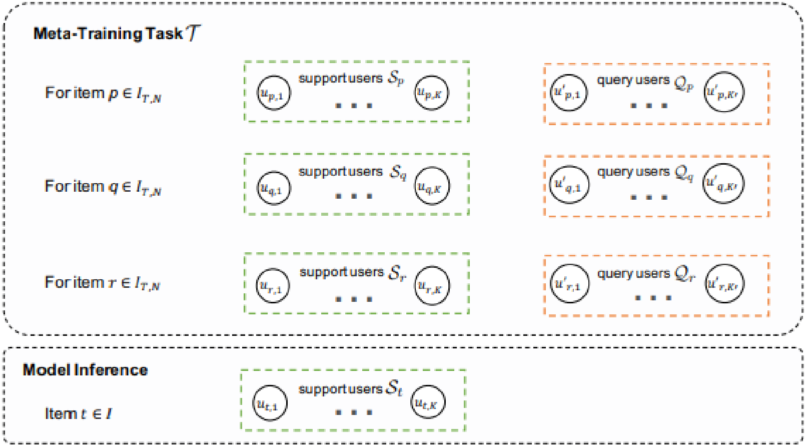


Figure 2: Episodic few-shot learning with meta-training task $\mathcal{T}$ and item embedding inference at meta-testing.

Learn-to-recommend

- $\mathcal{T} = \{\mathcal{I}_{\mathcal{T},N}, \mathcal{S}, \mathcal{Q}\}$ - few-shot task
- $\mathcal{I}_{\mathcal{T},N} \subset \mathcal{I}$ - items in $\mathcal{T}$
- $\mathcal{S}$ - set of support users in $\mathcal{T}$
- $\mathcal{Q}$ - set of query users in $\mathcal{T}$

The *few-shot recommender* $\mathbf{R}_F$ takes as input the support users $\mathcal{S}$ to *learn-to-compose* representations for items $i \in \mathcal{I}_{\mathcal{T},\mathcal{N}}$

$\mathbf{R}_F$ is trained by matching the item recommendations it generates for query users $\mathcal{Q}$ with their corresponding ground-truth interactions over $\mathcal{I}_{\mathcal{T},N}$

Initial Item Prototype

- $\mathcal{T}$ - few-shot task
- $\mathcal{I}_{\mathcal{T},N}$ - items in $\mathcal{T}$
- X_H - interactions in $\mathcal{T}$

We want $\mathbf{R}_F$ to learn a shared metric space of users and items

- $F_U(\cdot \,|\, \phi)$ - pre-trained user encoder of base recommender $\mathbf{R}_B$, parametrized with ϕ
- $G_U(\cdot \,|\, \theta)$ - few-shot user encoder, with parameters initialized from $F_U(\cdot \,|\, \phi)$, but parametrized with learnable parameters θ
- $\mathbf{p}_i = \frac{1}{\mathcal{S}_i} \sum_{u_{i,k} \in \mathcal{S}_i} G_U(u_{i,k}, X_H \,|\, \theta)$ - prototype for item $i \in \mathcal{I}_{\mathcal{T},N}$

Initial Item Prototype

Challenges in handling long-tail items:

- due to sparse support sets, the prototypes are noisy and sensitive to outliers
- due to diversity of tail items, averaging may lack the resolution to discriminate across them

Initial Item Prototype

Challenges in handling long-tail items:

- due to sparse support sets, the prototypes are noisy and sensitive to outliers
- due to diversity of tail items, averaging may lack the resolution to discriminate across them

For these reasons, the few-shot recommender $\mathbf{R}_F$ needs a strong *inductive bias* during prototype learning.
Thus, *item-item* relationship knowledge acquired by base recommender $\mathbf{R}_B$ is used to enhance item prototypes.

Head-Tail Meta Knowledge Transfer

- R_B - base recommender
- h_i - item embedding in $\mathbf{R}_B$

Item-item proximity $sim_b(\cdot)$ in the latent space of $\mathbf{R}_B$ is denoted by:

$$p_B(i,j) \propto sim_b(h_i, h_j) = cos(h_i, h_j) \qquad\qquad i,j \in \mathcal{I}$$

The goal is to extract knowledge from items most related to i. However, dynamically identifying related items during prototype construction is not scalable. Thus, a compact representation of *item-item proximity* knowledge is required.

Group-Enhanced Item Prototype Learning

A set of M ($M \ll |\mathcal{I}|$) group embeddings $\mathcal{Z}_M$ is learned to serve as a *basis vectors* modeling *item-item proximity* in the latent space of $\mathbf{R}_B$.

$$\mathcal{Z}_M = \{z_m \in \mathbb{R}^D : m \in \{1, \ldots, M\}\}$$

To enhance prototype of item $i \in \mathcal{I}_{\mathcal{T},N}$, a *group-enhanced prototype* $\mathbf{g}_i \in \mathbb{R}^D$ is synthesized as a mixture over the M group embeddings.

Group-Enhanced Item Prototype Learning

The mixture coefficients of a *group-enhanced prototype* $\mathbf{g}_i$ are estimated by a learnable attention mechanism.

$$\mathbf{g}_i = \sum_{m=1}^{M} \alpha_{im} z_m \qquad \alpha_{im} = \frac{\exp(\mathbf{W}_q \mathbf{p}_i \cdot \mathbf{k}_m)}{\sum_{m'=1}^{M} \exp(\mathbf{W}_q \mathbf{p}_i \cdot \mathbf{k}_{m'})}$$

Where $\mathcal{K}_M = \{\mathbf{k}_m \in \mathbb{R}^D : m \in \{1, \ldots, M\}\}$ is an auxiliary set of trainable keys to index the group embeddings, and $\mathbf{W}_q \in \mathbb{R}^{DxD}$ projects the prototype $\mathbf{p}_i$ into a query to index the centroids.

Task-level Stochastic Knowledge Distillation

In order to learn *group embeddings* $\mathcal{Z}_M$ that capture *item-item relationships* in $\mathbf{R}_B$, a *knowledge distillation* strategy is used.

A compact *student* model (group embeddings $\mathcal{Z}_M$) is encouraged to emulate predictions of the teacher (item proximity distribution in $\mathbf{R}_B$).

Since operating directly on all items in $\mathcal{I}$ is not scalable, student model is trained at the granularity of each meta-training task $\mathcal{T}$.

Task-level Stochastic Knowledge Distillation

- $\mathcal{T} = \{\mathcal{I}_{\mathcal{T},N}, \mathcal{S}, \mathcal{Q}\}$ - few-shot task
- $\mathcal{I}_{\mathcal{T},N} \subset \mathcal{I}$ - items in $\mathcal{T}$
- $p_B(i,j)$ - proximity of $i,j \in \mathcal{I}$ in $\mathbf{R}_B$
- $\mathbf{g}_i$ - group enhanced item prototype

For each item $i \in \mathcal{I}_{\mathcal{T},N}$,
a soft probability distribution
$p_B(j \mid i, \mathbf{R}_B)$ over other items $j \in \mathcal{I}_{\mathcal{T},N}$ is calculated.
$T > 0$ is a temperature scaling hyper-parameter.

$$p_B(j \mid i, \mathbf{R}_B) = \frac{\exp(p_B(i,j)/T)}{\sum_{k \in \mathcal{I}_{\mathcal{T},N}} \exp(p_B(i,k)/T)} \qquad i,j \in \mathcal{I}_{\mathcal{T},N}$$

Analogously, item similarity distribution $p_F(j \mid i, \mathbf{Z}_M)$ for the student model $\mathcal{Z}_B$ is defined.

$$p_F(j \mid i, \mathcal{Z}_M) = \frac{\exp(\mathrm{sim}_m(\mathbf{g}_i, \mathbf{g}_j))}{\sum_{k \in \mathcal{I}_{\mathcal{T},N}} \exp(\mathrm{sim}_m(\mathbf{g}_i, \mathbf{g}_k)}) \qquad i,j \in \mathcal{I}_{\mathcal{T},N}$$

Task-level Stochastic Knowledge Distillation

- $p_B(j \mid i, \mathbf{R}_B)$ - item similarity distribution for base recommender
- $p_F(j \mid i, \mathcal{Z}_M)$ - item similarity distribution for group-enhanced prototypes

The two distributions are aligned by minimizing cross-entropy between their task-level similarities. Since each item is typically only related to very few items within task T, the distillation loss L_G minimizes distribution divergence over the top-n related items ($n \approx 10$).

$$L_G = -\frac{1}{nN} \sum_{i \in \mathcal{I}_{\mathcal{T},N}} \sum_{j \in \pi_{B,n}(i)} p_B(j \mid i, \mathbf{R}_B) \log p_F(j \mid i, \mathcal{Z}_M)$$

$\pi_{B,n}(i)$ denotes the top-n most related items to i within $\mathcal{I}_{\mathcal{T},N}$ based on teacher $\mathbf{R}_B$. The loss is trained jointly with the rest of the framework.

Item Prototype Fusion via Neural Gating

The initial prototype $\mathbf{p}_i$ for item $i \in \mathcal{I}_{\mathcal{T},N}$ directly encodes its support users $\mathcal{S}_i$, while the group-enhanced prototype $\mathbf{g}_i$ captures the knowledge transferred from related items.

Final *gated item prototype* $\mathbf{e}_i \in \mathbb{R}^D$ is created by merging $\mathbf{p}_i$ and $\mathbf{g}_i$ using a neural gating layer.

$$\begin{aligned} \mathbf{gate} &= \sigma(\mathbf{W}_{g1}\mathbf{p}_i + \mathbf{W}_{g2}\mathbf{g}_i + \mathbf{b}_g) \qquad i \in \mathcal{I}_{\mathcal{T},N} \\ \mathbf{e}_i &= \mathbf{gate} \odot \mathbf{p}_i + (1 - \mathbf{gate}) \odot \mathbf{g}_i \end{aligned}$$

$\mathbf{W}_{g_1} \in \mathbb{R}^{D \times D}, \mathbf{W}_{g_2} \in \mathbb{R}^{D \times D}$, and $\mathbf{b}_g \in \mathbb{R}^D$ are learnable parameters, $\odot$ denotes element-wise product operation, and σ is the sigmoid non-linearity.

Few-shot Recommender Training

Each task $\mathcal{T}$ minimizes a negative log-likelihood L_P between the few-shot recommendations for query users $\mathcal{Q}$ and their ground-truth interactions in $\mathcal{T}$.

$$L_P = -\frac{1}{KN} \sum_{i \in \mathcal{I}_{\mathcal{T},N}} \sum_{u'_{i,k'} \in \mathcal{Q}_i} \log p_F(i \mid u'_{i,k'}, \theta)$$

$p_F(i \mid u'_{i,k'}, \theta)$ is computed based on cosine similarity and the choice of likelihood function for few-shot training.

Few-shot Likelihood Choices

$$L_P = -\frac{1}{KN} \sum_{i \in \mathcal{I}_{\mathcal{T},N}} \sum_{u'_{i,k'} \in \mathcal{Q}_i} \log p_F(i \mid u'_{i,k'}, \theta)$$

The authors considered the following likelihood functions for few-shot training:

- **Multinomial log-likelihood:**

$$p_F(i \mid u', \theta) = \frac{\exp(\text{sim}_m(\mathbf{e}_{u'}, \mathbf{e}_i))}{\sum_{j \in \mathcal{I}_{\mathcal{T},N}} \exp(\text{sim}_m(\mathbf{e}_{u'}, \mathbf{e}_j))}) \qquad u' \in \mathcal{Q}_i$$

- **Logistic log-likelihood:**

$$\log p_F(i \mid u', \theta) = \beta \log \sigma(\hat{y}_{u'i}) + \sum_{j \in \mathcal{I}_{\mathcal{T},N}, u' \notin N_j} \log(1 - \sigma(\hat{y}_{u'j}))$$

Few-shot Recommender Training

The overall loss is composed of the few-shot recommendation loss L_P and the knowledge distillation loss L_G:

$$L = L_P + \lambda L_G$$

where λ is a tunable hyper-parameter.

Model inference

The gated prototype $\mathbf{e}_i$ is inferred for each item $i \in \mathcal{I}$ by sub-sampling K interactions as the support set.

Item recommendations for each user $u \in \mathcal{U}$ are given by:

$$\hat{y}_f(u, i) = \mathrm{sim}_m(\mathbf{e}_u, \mathbf{e}_i) \qquad i \in \mathcal{I} \qquad \mathbf{e}_u = G_U(u, X \mid \theta)$$

The final recommendation is given by ensembling predictions from $\mathbf{R}_F$ and $\mathbf{R}_B$.

$$\hat{y}(u, i) = (1 - \eta) \cdot \hat{y}_b(u, i) + \eta \cdot \hat{y}_f(u, i) \qquad \eta \in (0, 1)$$

Architecture overview

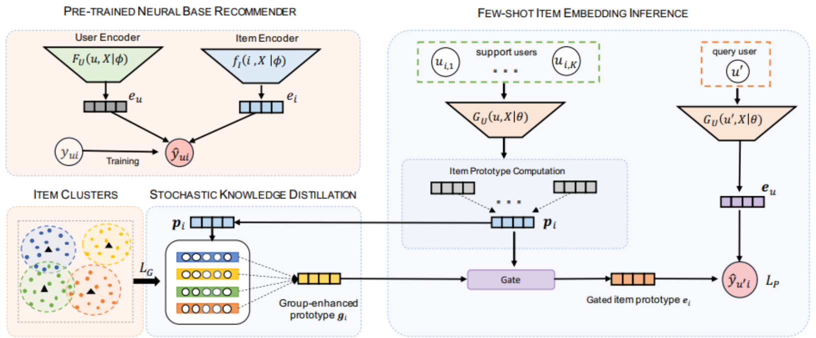


Figure 3: Architecture diagram of PROTOCF depicting the different model components: pre-trained neural base recommender R_B (top left), group embedding learning via stochastic knowledge distillation L_G (bottom left), initial item prototype construction via support set averaging followed by group-enrichment and adaptive gating to construct gated item prototype e_i (right).

Experiments

(RQ1) Does ProtoCF beat state-of-the-art NCF and sparsity-aware methods on *overall* recommendation performance?

Experiments

(RQ1) Does ProtoCF beat state-of-the-art NCF and sparsity-aware methods on *overall* recommendation performance?

(RQ2) What is the impact of item *interaction sparsity* on the *few-shot* recommendation performance of ProtoCF?

Experiments

(RQ1) Does ProtoCF beat state-of-the-art NCF and sparsity-aware methods on *overall* recommendation performance?

(RQ2) What is the impact of item *interaction sparsity* on the *few-shot* recommendation performance of ProtoCF?

(RQ3) How do the different *architectural* choices impact the few-shot and overall performance of ProtoCF?

Experiments

(RQ1) Does ProtoCF beat state-of-the-art NCF and sparsity-aware methods on *overall* recommendation performance?

(RQ2) What is the impact of item *interaction sparsity* on the *few-shot* recommendation performance of ProtoCF?

(RQ3) How do the different *architectural* choices impact the few-shot and overall performance of ProtoCF?

(RQ4) How do the *hyper-parameters* (distillation loss balance factor λ and meta-training task size N) affect ProtoCF?

Datasets

- Epinions - product ratings for an e-commerce platform
- Yelp - user ratings on local businesses located in the state of Arizona
- Weeplaces - check-ins for businesses of different categories, like Nightlife, Outdoors, or Entertainment
- Gowalla - restaurant check-ins by users across different cities in United States

Baselines

- Neural Base Recommenders (BPR, VAE-CF, CDAE)
- Neural Collaborative Filtering
- Neural Graph Collaborative Filtering
- Cofactor
- EFM
- DropoutNet
- MetaRec-LWA
- MetaRec-NLBA

Overall Recommendation Results (RQ$_1$)

Dataset	Epinions		Yelp		Weeplaces		Gowalla	
Metric	**N@50**	**R@50**	**N@50**	**R@50**	**N@50**	**R@50**	**N@50**	**R@50**
			STANDARD NEURAL COLLABORATIVE FILTERING METHODS					
BPR [34]	0.0860	0.1666	0.0749	0.1416	0.2537	0.3778	0.1661	0.2703
NCF [12]	0.0878	0.1694	0.0752	0.1429	0.2462	0.3694	0.1702	0.2745
NGCF [48]	0.0913	0.1725	0.0826	0.1579	0.2533	0.3764	0.1696	0.2758
VAE-CF [24]	0.0938	0.1778	0.0854	0.1602	0.2482	0.3730	0.1710	0.2769
CDAE [51]	0.0927	0.1774	0.0870	0.1611	0.2570	0.3760	0.1634	0.2644
			SPARSITY-AWARE LONG-TAIL ITEM RECOMMENDATION METHODS					
DropoutNet [45]	0.0881	0.1697	0.0761	0.1435	0.2516	0.3751	0.1697	0.2768
Cofactor [23]	0.0845	0.1639	0.0734	0.1402	0.2342	0.3539	0.1596	0.2642
EFM [4]	0.0742	0.1534	0.0741	0.1403	0.2306	0.3429	0.1532	0.2584
MetaRec-NLBA [43]	0.0453	0.0937	0.0381	0.0875	0.1698	0.2889	0.0753	0.1384
MetaRec-LWA [43]	0.0467	0.0943	0.0392	0.1425	0.1702	0.2997	0.0722	0.1391
			PROTOTYPICAL COLLABORATIVE FILTERING RECOMMENDERS (PROTOCF)					
PROTOCF + BPR	0.0964	0.1812	0.0815	0.1533	0.2576	0.3879	0.1737	0.2800
PROTOCF + VAE	**0.0977**	**0.1830**	0.0857	0.1605	**0.2725**	**0.4035**	**0.1899**	**0.3004**
PROTOCF + CDAE	0.0972	0.1824	**0.0883**	**0.1623**	0.2697	0.4011	0.1786	0.2875
Percentage Gains	4.16%	2.92%	1.50%	0.75%	6.03%	6.80%	11.05%	8.49%

Table 4: Overall item recommendation results on four datasets, R@K and N@K denote Recall@K and NDCG@K metrics at $K = 50$. **Sparsity-aware models are generally outperformed by standard NCF methods on overall item recommendation; PROTOCF achieves *overall* NDCG@50 gains of 6% and Recall@50 gains of 4% (over the best baseline) across all datasets.**

Overall Recommendation Results (RQ$_1$)

Key observations:

- Models based on autoencoders (VAE-CF, CDAE) and graph neural networks (NGCF) outperform other latent-factor models (NCF, BPR)
- Model regularization strategies using item co-occurence information (CoFactor, EFM) for improving long-tail recommendations are worse than BPR in overall performance
- Sparsity-aware meta-learning models (MetaRec) perform poorly in overall item rankings
- ProtoCF outperforms state-of-the-art baselines on overall item rankings

Few-Shot Recommendation Results (RQ$_2$)

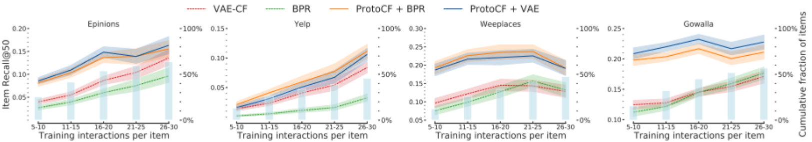


Figure 4: Few-shot item recommendation results: Performance comparison for long-tail items with varying number of training interactions K (5 to 30); lines denote model performance (Recall@50) and background histograms indicate the cumulative fraction of the item inventory covered by tail items with $\leq K$ impressions. Overall performance generally increases with K for all models; ProtoCF achieves notably stronger gains (over baselines) for items with few training interactions (small K).

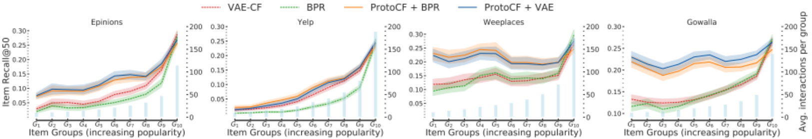


Figure 5: Impact of item interaction sparsity: Performance comparison for item-groups sorted in increasing order by their average training interaction counts; lines denote model performance (Recall@50) and background histograms indicate the average number of interactions in each item-group. ProtoCF has significant performance gains (over baselines) on the tail items (item-groups G_1 to G_8) while maintaining comparable performance on the head items (item-groups G_9 to G_{10}).

Model Ablation Study (RQ_3)

Dataset	Epinions		Gowalla	
Metric	Overall R@50	Few-shot R@50	Overall R@50	Few-shot R@50
ProtoCF	0.1830	0.1070	0.3004	0.2195
w/o Prototype Gating	0.1823	0.0948	0.2992	0.2082
w/o Knowledge Distillation	0.1805	0.0869	0.2923	0.1983
ProtoCF-Avg	0.1801	0.0712	0.2898	0.1696
ProtoCF-logistic	0.1804	0.0896	0.2853	0.1843
VAE-CF [24]	0.1778	0.0549	0.2769	0.1316
MetaRec-LWA [43]	0.0943	0.0898	0.1391	0.1804

Table 5: Model ablation study of ProtoCF; few-shot performance is reported for tail items (less than 20 training interactions). Knowledge transfer and prototype gating contribute 10-19% and 5-11% to few-shot gains respectively.

Parameter Sensitivity (RQ_4)

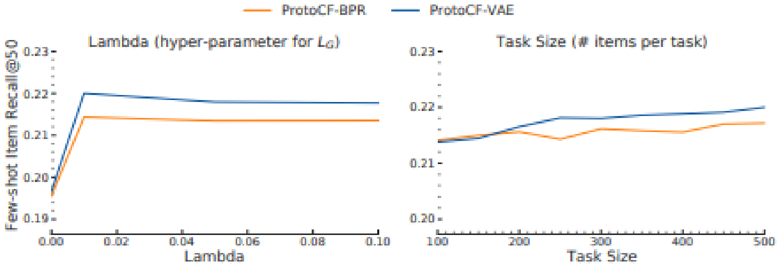


Figure 6: Few-shot performance on Gowalla (for tail items with less than 20 training interactions) is higher for larger meta-training tasks; the empirically optimal value of balance factor $\lambda = 0.01$ also transfers across all datasets.

Reproducibility

set the latent embedding dimension to 128 for consistency. Our implementation of ProtoCF and datasets are publicly available[4].

Reproducibility

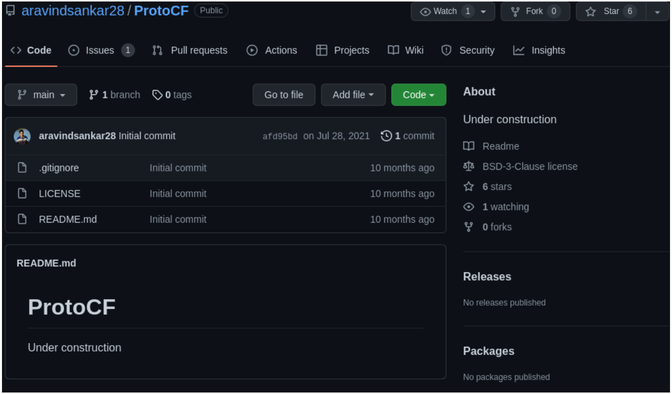
set the latent embedding dimension to 128 for consistency. Our implementation of ProtoCF and datasets are publicly available[4].

[4] https://github.com/aravindsankar28/ProtoCF

Reproducibility

set the latent embedding dimension to 128 for consistency. Our implementation of ProtoCF and datasets are publicly available[4].

[4]https://github.com/aravindsankar28/ProtoCF



Conclusion

- A sophisticated solution for a specific problem
- Research orthogonal to mainstream advances in recommender systems
- Architecture-agnostic method for improving neural recommenders