Efficient and Flexible Long-Tail Recommendation Using Cosine Patterns

Yaqiong Wang[*], Junjie Wu[†], Zhiang Wu[‡], Gediminas Adomavicius[§]

Abstract

With the increasing use of recommender systems in various application domains, many algorithms have been proposed for improving the accuracy of recommendations. Among various dimensions of recommender systems performance, *long-tail* (niche) recommendation performance remains an important challenge, due in large part to the *popularity bias* of many existing recommendation techniques. In this study, we propose CORE, a cosine-pattern-based technique, for effective long-tail recommendation. Comprehensive experimental results compare the proposed approach to a wide variety of classical, widely-used recommendation algorithms and demonstrate its practical benefits in accuracy, flexibility, and scalability, in addition to the superior long-tail recommendation performance.[1]

Keywords: Recommender Systems, Pattern-Based Recommendation, Cosine Patterns, Long-Tail Recommendation

[*]Leavey School of Business, Santa Clara University.
[†]School of Economics and Management, Beihang University.
[‡]School of Information Engineering, Nanjing Audit University.
[§]Carlson School of Management, University of Minnesota

[1]This article substantially augments and improves its preliminary version, which appeared in the IEEE ICDM Workshop on Domain Driven Data Mining (Wang et al. 2014).

1 Introduction and Motivation

Recommender systems play an important role in online business, as high-quality personalized recommendations have been shown to have huge impact on users' purchase and consumption decisions (Pathak et al. 2010). For example, 60% of Netflix rentals and 35% of Amazon's sales are attributed to their recommendation systems (Hosanagar et al. 2013); also, more than 40% of users on Spotify continuously listen to personalized playlists generated by the platform (Buskirk 2016). Over the years, a wide variety of methods, typically based on collaborative filtering (CF) techniques, have been proposed to improve the relevance of recommended items (Adomavicius and Tuzhilin 2005, Ricci et al. 2022). Although these methods have been widely applied, such recommender systems also have been shown to have *popularity bias*, which refers to the tendency of the systems to recommend disproportionately more popular items, i.e., items with more ratings or purchases, to users (Fleder and Hosanagar 2009). However, recommending comparatively popular items is not always an advantageous strategy; e.g., recommending items that are already well-known and bestselling (and that the users are much more likely to be aware of) arguably might be less valuable than finding something truly relevant and personalized from the "long tail" of the item popularity distribution. If a platform wants to highlight its "back catalog", CF algorithms typically need specific enhancements to identify relevant but less popular products (Lee and Hosanagar 2019). Yet, due to the fact that less data is available on these products, user preferences for them are harder to predict and, thus, accurately recommending the long-tail (niche)[2] items remains an important challenge. This is the focus of our study.

The value of long-tail recommendations has been increasingly recognized. On the demand side, the niche titles can increase consumer surplus and drive consumption (Brynjolfsson et al. 2006). In particular, consumers are known to have a propensity to seek variety over time (Pessemier 1978). Recommending niche items can encourage users to try products that are outside their awareness (Brynjolfsson et al. 2011) and, thus, better satisfy consumers' het-

[2]Throughout the paper, we use terms *long-tail items* and *niche items* interchangeably (i.e., as fully synonymous) to describe the items that fall in the tail of consumption distribution.

erogeneous needs. The discovery of niche or unexpected items can also help to increase user satisfaction (Hijikata et al. 2009, Kotkov et al. 2018). In the long run, more user engagement can be stimulated, driving up consumption and increasing overall demand (Anderson et al. 2020, Lin et al. 2017). On the supply side, niche products can be more profitable for companies, e.g., niche movies cost a fraction of blockbusters to make and market (Anderson 2006). For platforms like Netflix, recommending users more niche movies and fewer blockbusters can be advantageous, as blockbusters tend to have much higher licensing costs (Goldstein and Goldstein 2006). Long-tail items may even boost popular item sales by offering convenience of "one-stop shopping" to consumers for both their mainstream and niche interests, as discussed in Goel et al. (2010). Also, niche recommendations in online marketplaces like Amazon can incentivize the sellers of niche products to stay on the platform instead of being crowded out by popular products (Abdollahpouri et al. 2019).

Long-tail recommendations have been applied in industry as well. For example, Netflix views democratizing access to long-tail products and services as well as generalizing predictions usefully at the tail as critical missions of their recommender systems (Gomez-Uribe and Hunt 2015). At Amazon, the goals of their content recommender system design include maximizing the value of deep content catalogs, increasing revenue for long-tail content by surfacing it in recommendations (Rabowsky and Morrison 2020). Similarly, the music discovery approach used on Spotify focuses suggestions on the long end of the popularity tail (Jacobson et al. 2016), e.g., it creates more exposure for long-tail artists and content by ensuring they receive certain amount of recommendations (Semerci et al. 2019, Tian et al. 2019). In summary, niche item recommendation can be of importance in various application domains; therefore, having access to accurate long-tail recommendation models is of interest.

However, many platforms have not taken advantage of niche products' potential, as discovering relevant long-tail items is not easy (Tan et al. 2017), and popularity bias (which is perpetuated in many existing recommender systems) is known to exacerbate this problem. To alleviate the popularity bias caused by accuracy-oriented recommendation algorithms,

other recommendation performance aspects, e.g., diversity and novelty, have been studied in prior work (Castells et al. 2015). Novelty and diversity could be enhanced by re-ranking the initial recommendation list (Ziegler et al. 2005, Zhang and Hurley 2008, Adomavicius and Kwon 2012) or by optimizing the ranking process using a combined objective of accuracy and diversity (Yin et al. 2012, Shi 2013, Ye et al. 2021). While improvements in recommendation diversity and novelty can sometimes be associated with better long-tail performance as well, it is not guaranteed to be so. For example, high diversity can be easily achievable in many contexts by recommending diverse popular items, which would do nothing for improving long-tail recommendations; similarly, recommender systems can improve individual diversity while still reducing aggregate diversity (Fleder and Hosanagar 2009, Jannach et al. 2013, Chen et al. 2018). Long-tail recommendation could also be achieved by improving rating estimation specifically for niche items (Park 2013, Niemann and Wolpers 2013, Ferraro 2019, Liu and Zheng 2020); however, such methods tend to achieve this at the substantial expense of overall recommendation accuracy or by requiring a much richer set of features and extra preprocessing. Meanwhile, as the user population and item catalog in the system grow over time, scalability of long-tail recommendation is also an important concern. The ability to easily adjust (parameterize) the popularity of recommended items is another highly practical and important characteristic that is absent in many long-tail recommendation approaches.

Pattern-based, especially association-rule-based, recommendation algorithms have attracted some attention since the early days of recommender systems research (Mobasher et al. 2001, Lin et al. 2002). One key reason is the recommendation interpretability (e.g., "people who bought X also bought Y"). Typically, pattern-based algorithms first build a knowledge base containing item co-occurrence patterns (e.g., association rules or itemsets) and then recommend items to users based on this knowledge. Some platforms have used this approach in their commercial recommender systems, e.g., YouTube used association rules to recommend relevant videos to their users (Davidson et al. 2010). However, the traditional framework for association rule discovery has certain limitations that can lead to less accurate

recommendations (especially with skewed data distributions), as will be discussed later.

To address some of these limitations, in this study we propose the CORE (COsine pattern-based REcommendation) approach to accurate long-tail recommendation. CORE is a pattern-based method, which finds associations, represented by *cosine patterns*, among different items (especially niche items) and then utilizes the discovered associations for the purpose of item recommendation. Compared to numerous widely used recommendation baselines, CORE demonstrates excellent long-tail performance in a variety of contexts. Due to its ability to limit the discovery of spurious patterns, accuracy of CORE remains highly competitive on sparser, heavier-tailed data. The proposed method also supports convenient parameterization of the popularity of recommended items, i.e., provides flexibility in generating recommendations of different popularity (or long-tail) levels in order to achieve various recommendation goals. The scalability of CORE is facilitated by a specialized data structure that is advantageous for real-time recommendation capabilities in large-scale applications. The paper provides comprehensive experiments on multiple real-world data sets to illustrate the advantages of CORE with respect to a number of classical, widely used pattern-based and collaborative-filtering-based approaches.

2 Background and Related Work

With the popularity of the consumer-oriented content delivery and retail platforms, recommender systems have been progressively developed for various application domains, including movies, music, books, etc., to alleviate information overload and facilitate personalized information retrieval (Ricci et al. 2022). Over the years, *accuracy* of recommender systems has been the major lens through which their performance is evaluated and compared. For example, Netflix held an open competition (with $1M prize for the winner) for the most accurate recommendation algorithm to predict user ratings for movies (Koren et al. 2009).

As mentioned earlier, research studies increasingly point out that focusing on accuracy alone in recommender systems can result in popularity bias (Fleder and Hosanagar 2009), especially in classical collaborative-filtering-based methods. Taking advantage of the long-

tail market is one of the keys towards increasing profits on e-commerce platforms (Anderson 2006). Thus, in this study, we focus on the *long-tail* perspective of recommender systems with the goal of developing a recommendation method that can achieve better *long-tail* performance while still being highly competitive in terms of recommendation accuracy.

Previous studies have attempted to address the long-tail challenge in different ways (Qin 2021). Some work focused on taking a broader perspective on recommender systems evaluation, rather than focusing just on accuracy, which gave rise to a number of additional recommendation performance dimensions, e.g., diversity and novelty (Castells et al. 2015). Different metrics related to recommendation diversity and novelty have been proposed, e.g., individual (intra-list) diversity (Zhang and Hurley 2008, Ziegler et al. 2005), aggregate diversity (Fleder and Hosanagar 2009), serendipity (Murakami et al. 2007, Zhou et al. 2010), unexpectedness (Adamopoulos and Tuzhilin 2014), and recommendation algorithms for improving these metrics (Zhang and Hurley 2008, Adomavicius and Kwon 2012, Adamopoulos and Tuzhilin 2014). However, diversity and novelty metrics represent an indirect way to affect long-tail recommendation performance; i.e., although these metrics have some correlation with long-tail performance (as they typically affect the distribution of recommended items), it is not guaranteed to be the case. For example, diversity metrics like ILAD (intra-list average distance) and ILMD (intra-list minimal distance) used in some previous studies (Chen et al. 2018, Ye et al. 2021) mainly focus on diversifying items in each recommendation list instead of considering niche items directly. Although these metrics are effective in increasing individual diversity, aggregate diversity of recommendations can still decrease (Fleder and Hosanagar 2009, Jannach et al. 2013), leading to worse long-tail recommendation.

Therefore, some other studies have investigated ways to tackle the long-tail recommendation problem more directly. Several such studies incorporated the long-tail-aware computations as part of the pre-processing step of the recommendation process. As one example of a pre-processing approach, Park (2013) proposed to first split items into head- and tail-groups based on their rating frequency, cluster tail items into different clusters, and then

predict user ratings within each cluster; the results show that accuracy of the long-tail item recommendations indeed increases through clustering. A similar idea was explored by Zhang and Hurley (2009), Sreepada and Patra (2020), where items in each active user's profile are clustered first, and recommendations are generated based on each cluster instead of complete user profiles. In contrast to pre-processing approaches, several other studies propose to embed information about item popularity more directly into the recommendation generation process, e.g., discounting popular items when learning to rank. For instance, in order to promote long-tail items in recommendations, probability for a user to consume a certain item (i.e., recommendation score) based on the whole user-item interaction graph could be discounted by the rating frequency of that item (Yin et al. 2012, Krishnan et al. 2018, Menon et al. 2020, Chen et al. 2021). Similarly, Shi (2013) proposes a Markovian graph-based recommendation approach, where weights on edges could be tuned to enhance the probability of recommending long-tail items. Another recent study (Liu and Zheng 2020) on session-based recommendation uses rectification factors in a neural network to adjust the probability of recommending head and tail items. Other related studies propose to optimize the recommendation list based on different objectives, e.g., increasing accuracy, reducing popularity (Hamedani and Kaedi 2019), or prioritizing niche items based on their usage (e.g., co-occurrence with other popular items) (Niemann and Wolpers 2013, Qin et al. 2020).

Another set of studies propose *hybrid* approaches to improve long-tail recommendation performance. For example, in Alshammari et al. (2017), content-based and collaborative-filtering recommendation methods are used in combination to recommend long-tail and popular items, respectively. Similarly, in Zhang et al. (2012) and Ribeiro et al. (2015), an ensemble of outputs of multiple recommendation algorithms is used to balance accuracy and novelty. Other related studies use side information (i.e., information other than user-item interactions) to direct long-tail recommendation. Examples include adding semantic knowledge extracted from content information to better represent long-tail items (Bai et al. 2017, Zhang et al. 2021) or explicitly collecting users' preferences for different types of items

(Wen et al. 2020). Recent advances in deep learning also yielded new approaches for hybrid long-tail recommendation. These approaches use deep neural networks to combine different sources of information, such as text, images, audio, other complex content types and user profiles (Ferraro 2019, Li et al. 2019, Zhang and Hong 2021), which allows for more flexibility in representing and recommending a variety of different items.

In this paper, we focus on a new pattern-based recommendation method that tries to avoid some of the limitations of existing long-tail recommendation approaches (such as requiring a much richer set of features, significant pre-processing, or resulting in substantial reductions in accuracy), while exhibiting scalability, flexibility, and explainability benefits.

3 Cosine-Pattern-Based Recommendation

3.1 Basics of Cosine Patterns

Association rule or frequent itemset mining is a very popular and well-known approach to discovering item co-occurrence patterns in data (Agrawal et al. 1993, Ceglar and Roddick 2006). Many metrics of pattern strength (or interestingness) have been proposed in research literature (Tan et al. 2002), *support* and *confidence* being the most canonical and widely used among them. However, the traditional rule discovery framework based on support and confidence also has well-known limitations that make it less appealing for recommendation and, especially, long-tail recommendation. In particular, the confidence metric often might not reflect a meaningful association among items, in part due to item-popularity-related issues, and the traditional support-based pruning strategy for pattern mining might be inadequate on data with skewed popularity distributions (Xiong et al. 2006), leading to either popularity bias or lower-quality recommendations. We provide the key background details on association rules and some of their limitations in Appendix A1 of Online Supplement.

To address these limitations, in this paper, we adopt *cosine* (Tan et al. 2002) as an interestingness measure to be used simultaneously with *support* for pattern evaluation and pattern-based recommendation. The *cosine* value of itemset P with K items is defined as:[3]

[3]The metric reduces to the traditional *cosine* measure when $K = 2$ (Wu et al. 2012).

$$cos(P) = \frac{supp(P)}{\left(\prod_{k=1}^{K} supp(\{i_k\})\right)^{1/K}},\ K \geq 2, \tag{1}$$

Itemset P is a cosine pattern w.r.t. user-defined support and cosine thresholds τ_s and τ_c; i.e., only patterns where $supp(P) \geq \tau_s$ and $cos(P) \geq \tau_c$ would be considered.

As shown in (1), the cosine value is calculated as the support (i.e., overall prevalence) of the pattern normalized by the geometric mean of the supports of individual items within the pattern. Intuitively, the cosine value can be viewed as "cohesiveness" of a pattern. Patterns with higher cosine values contain items with similar levels of popularity. This is also independent of the overall prevalence of the pattern, i.e., patterns with lower support can have high cosine values as long as the co-occurring items have similar support. In other words, a key appeal of the cosine measure lies in its alignment with the *anti-cross-support* property. P is a *cross-support pattern* (CSP) w.r.t. τ ($0 \leq \tau \leq 1$) if its CSP value $V_{csp}(P) \leq \tau$ (Xiong et al. 2006). Here $V_{csp}(P) = s(i_l)/s(i_h)$, with i_l and i_h representing items with lowest and highest support values in P, respectively. Thus, a CSP is a pattern containing items with significantly different support levels. As shown by Wu et al. (2012), $cos(P) \leq \sqrt[K]{V_{csp}(P)}$. This implies that a pattern tends to have a lower cosine value as $V_{csp}(P)$ gets smaller; i.e., CSPs are less likely to be cosine patterns.

Importantly, cosine patterns (i.e., more "cohesive", anti-cross-support patterns) are extremely useful for long-tail recommendation, as they are more likely to point to stronger, more meaningful associations among items, as we discuss next. Also, cosine patterns can be mined efficiently due to their *Conditional Anti-Monotone Property* (CAMP). More details about cosine patterns, including CAMP, are provided in Appendix A2 of Online Supplement.

3.2 Cosine Patterns for Recommendation

As mentioned earlier, pattern-based recommender systems have attracted substantial attention, partly for their high interpretability of recommendations. A survey (Paraschakis et al. 2015) on more than 30 popular e-commerce platforms also reveals that industries of-

ten favor less complex recommendation techniques like association rules mining or nearest neighbor-based collaborative filtering for efficiency and engineering cost concerns. This indicates that improving the performance of pattern-based (e.g., itemset- or rule-based) methods is of theoretical and practical importance.

The prevalence of the cross-support patterns is largely responsible for the lower accuracy and higher popularity bias of traditional rule-based recommender systems. To illustrate this, Table 1 shows some representative examples of 2-item movie association rules (10 out of top 150 highest confidence rules) and 2-item cosine patterns (10 out of top 150 highest cosine patterns)[4] discovered from the MovieLens data[5] with $\tau_s = 2\%$, $\tau_{conf} = 76\%$, and $\tau_c = 0.5$.[6]

Table 1 shows a large imbalance of support levels between the antecedent and consequent in each example association rule, as indicated by low V_{csp} values; in contrast, the V_{csp} values for cosine patterns are substantially higher. In the last two columns of Table 1, we present two additional indicators of how related are the two movies that appear in each pattern (or how "cohesive" the pattern is). The first one is the correlation coefficient (CorrCoef) of the ratings for two movies, i.e., how similar the preferences are for these two movies among the users who saw both of them. The second indicator is the Jaccard similarity (JSim) of the movie consumptions, i.e., how similar are the sets of users who saw each movie. Both of these indicators consistently show that association rules contain items that are less related to each other (i.e., have substantially lower CorrCoef and JSim) than cosine patterns. This insight is further emphasized by Table 2, which provides the aggregate statistics across top-150 association rules and top-150 cosine patterns – the confidence-based patterns (i.e., association rules) contain items that are highly imbalanced in terms of their support and substantially less related to each other than the patterns mined based on the cosine measure.

In summary, recommendations generated from association rules have several limitations. As shown above, association rules tend to contain items that are less related to each other,

[4]For comparison, the illustrative sets of 10 cosine patterns and 10 association rules were picked to have similar *support* (and to be representative of the broad range of support values).

[5]Detailed information about this data set could be found in Table 3.

[6]Thresholds were set to have similar number (between 150 and 200) of patterns discovered in both cases.

Table 1: Examples of association rules and cosine patterns mined from the MovieLens data.

Association Rules							Association Rules
Antecedent	**Consequent**	**Supp (%)**	**Conf (%)**	**Cosine**	**V_{csp}**	**Corr Coef**	**JSim**
Dead Man (34)	Star Wars (583)	2.3	77.3	0.15	0.06	-0.22	0.05
Only You (39)	The Princess Bride (324)	2.9	77.8	0.24	0.12	0.06	0.10
Giant (51)	Casablanca (243)	3.9	81.1	0.33	0.21	0.21	0.20
Trees Lounge (50)	Fargo (508)	4.1	79.5	0.24	0.10	0.23	0.10
Weekend at Bernie's (60)	Back to the Future (350)	4.6	79.1	0.30	0.31	0.31	0.16
Dumb and Dumber (50)	Back to the Future (350)	4.8	82.2	0.29	0.20	-0.06	0.18
Flirting With Disaster (42)	The Empire Strikes Back (367)	5.5	77.4	0.25	0.10	0.12	0.24
Nell (81)	Raiders of the Lost Ark (420)	5.8	76.4	0.30	0.20	0.32	0.17
Victor/Victoria (77)	Return of the Jedi (507)	5.8	80.0	0.28	0.15	0.09	0.13
Casino (91)	Raiders of the Lost Ark (420)	7.0	78.8	0.34	0.22	0.14	0.20
Cosine Patterns							
Movie 1	**Movie 2**	**Supp (%)**	**Conf (%)**	**Cosine**	**V_{csp}**	**Corr Coef**	**JSim**
Manon of the Spring (58)	Jean de Florette (64)	2.3	37.9	0.51	0.38	0.76	0.60
Three Colors: White (59)	Three Colors: Blue (64)	2.9	45.8	0.87	0.42	0.75	0.58
A Grand Day Out (66)	The Wrong Trousers (118)	3.9	56.1	0.58	0.31	0.66	0.50
Private Benjamin (66)	Home Alone (137)	4.1	58.2	0.52	0.28	0.32	0.31
Bram Stoker's Dracula (120)	Interview with the Vampire (137)	4.8	38.3	0.52	0.34	0.45	0.55
Batman Forever (114)	Batman Returns (142)	4.8	35.4	0.51	0.40	0.35	0.55
Star Trek: Final Frontier (63)	Star Trek: Motion Picture (117)	5.6	45.3	0.55	0.33	0.44	0.52
Die Hard With a Vengeance (151)	Die Hard 2 (166)	5.7	35.8	0.51	0.33	0.75	0.68
Under Siege (124)	Clear and Present Danger (179)	5.8	44.4	0.51	0.31	0.50	0.47
Ghost (170)	Mrs. Doubtfire (192)	7.0	39.4	0.51	0.35	0.45	0.48

In parentheses after each movie title: the number of users who rated the movie;
V_{csp}: support ratio of items within rules/patterns;
CorrCoef: correlation coefficient of movie ratings;
JSim: Jaccard similarity of movie consumptions.

Table 2: Descriptive statistics of top-150 rules/patterns.

	Supp (%)	**Conf (%)**	**Cosine**	**V_{csp}**	**CorrCoef**	**JSim**
Association Rules	3.0 (1.08)	80.37 (3.49)	0.25 (0.05)	0.13 (0.05)	0.17 (0.22)	0.11 (0.05)
Cosine Patterns	11.6 (3.70)	42.50 (4.00)	0.53 (0.02)	0.83 (0.14)	0.32 (0.15)	0.55 (0.06)

Standard Deviation in Parentheses.

which can lead to lower accuracy when deployed for recommendations. As importantly, because many discovered association rules have low cross-support values, movies that end up being recommended using rules (i.e., movies in the consequent of the rule) are largely high-support (i.e., popular) items, such as *Star Wars*, *Fargo*, *Back to the Future*, and *Raiders of the Lost Ark*. This perpetuates the so-called popularity bias existing in many collaborative-filtering-based recommender systems, leading to insufficient recommendation of niche movies and, more generally, to long-tail recommendation challenges. As shown in Tables 1 and 2, cosine pattern mining can effectively overcome these limitations due to its anti-cross-support objective, which allows for discovery of more cohesive patterns, including patterns with less common (niche) items.[7] The ability to discover connections among items with comparatively

[7]E.g., consider pattern {*Manon of the Spring, Jean de Florette*}. Only a few users watched these two movies, but cohesiveness of the pattern is quite high: the movies are liked similarly by users who saw both

smaller audience is a key to addressing the long-tail recommendation challenge and, thus, is one of the motivating factors for the proposed cosine pattern-based method for long-tail recommendation. At the same time, the minimum support and cosine threshold parameters provide the flexibility to fine-tune the proposed method to the desired specifications (e.g., in terms of recommending popular vs. niche items), as will be shown later in the paper.

3.3 Basic Recommendation Scheme

We assume that the set of all applicable cosine patterns $\mathcal{CP}$ (i.e., patterns with $supp(P) \geq \tau_s$ and $\cos(P) \geq \tau_c$ for some user-specified thresholds τ_s and τ_c) has been mined in advance (e.g., using the standard library *CoPaMi*s (Wu et al. 2014)) and focus on the problem of generating top-K recommendations for each user. Given the set of all discovered cosine patterns $\mathcal{CP}$ and user u (represented by her consumption history C_u), the recommendation process consists of three main stages: (i) identifying u's target items T_u; (ii) identifying the set of eligible patterns $\mathcal{EP}_{ui}$ (where $\mathcal{EP}_{ui} \subseteq \mathcal{CP}$) for each target item $i \in T_u$; and (iii) calculating recommendation scores for each $i \in T_u$ (by aggregating information from $\mathcal{EP}_{ui}$) and ranking all target items according to the scores. We describe each stage below.

Stage 1: For each user u, target item set T_u consists of items not yet consumed by user u, i.e., $T_u = I \setminus C_u$, where I represents the set of all possible items.

Stage 2: For each target item i in T_u, we need to find eligible pattern set $\mathcal{EP}_{ui}$. An *eligible pattern* is defined as follows.

Definition 3.1 (ui-Related Eligible Pattern)**.** Given cosine pattern P and user u's consumption history C_u, P is an **eligible pattern** for u w.r.t. i, if $i \in P$, $i \notin C_u$, and $P \setminus \{i\} \subseteq C_u$.

Thus, cosine pattern P is a ui-related eligible pattern if: (1) P contains target item i, and (2) all other items in P (i.e., other than i) have been consumed by user u. Thus, any ui-related eligible pattern represents a cohesive itemset consisting of some items already consumed by u and one new item i, making item i a natural candidate for recommendation. The set of all ui-related eligible patterns is denoted as $\mathcal{EP}_{ui}$.

of them (CorrCoef = 0.76), and the sets of users who saw each movie are similar as well (JSim = 0.60). But, such a pattern is unlikely to be discovered as an association rule due to its relatively low confidence.

As a simple illustration, let's assume that we have a recommendation application with seven items, i.e., $I = \{A, B, C, D, E, F, G\}$, where the three cosine patterns (itemsets) have been mined from users' consumption histories: $\mathcal{CP} = \{\{A, B, C\}, \{A, D, E, G\}, \{A, B, D\}\}$. Also, let's assume that user u's consumption history is $C_u = \{B, C, D, E\}$. Then, user u's target item set is $T_u = I \setminus C_u = \{A, F, G\}$. Considering item A as a target item, we can see that $\mathcal{EP}_{uA} = \{\{A, B, C\}, \{A, B, D\}\}$. I.e., $\{A, B, C\}$ is a uA-related eligible pattern, since all items in it except A have been consumed by u, and so is pattern $\{A, B, D\}$. Note that $\{A, D, E, G\}$ is not an uA-related eligible pattern even though it also contains target item A, because there are multiple (i.e., more than one) target items in it (i.e., A and G).

Stage 3: For user u, recommendation score for target item i is derived from $\mathcal{EP}_{ui}$ by summing the cosine values of all patterns in $\mathcal{EP}_{ui}$, i.e., $score(u, i) = \sum_{P \in \mathcal{EP}_{ui}} cos(P)$. For any given user u, all target items in T_u will be ranked by their recommendation scores, and the recommendation list L_u for user u would be generated by selecting the top-K items.

In other words, CORE adopts a simple scoring method for target items $i \in T_u$, which adds up the cosine values of all ui-related eligible patterns. There have been several studies investigating ways in which pattern-based recommendation algorithms could combine eligible patterns to provide better recommendations. E.g., Wickramaratna et al. (2009) proposed a Dempster-Shaffer-based approach to combine rules with conflicting predictions. Ghoshal and Sarkar (2014) proposed to group rules with disjoint antecedents and same consequent and developed a probability model to select the group that maximizes the likelihood of purchasing target item for recommendation. Lin et al. (2002) adopted heuristics like adding up the supports and confidences of all eligible rules with the same consequent as its recommendation score. Although there exist different strategies for determining recommendation scores for items based on discovered patterns, theoretical or empirical studies on deriving optimal strategies are rarely seen. In this study, we used the cosine value of a pattern, as it provides a meaningful quantification of the itemset cohesiveness, as discussed earlier. Moreover, if target item i appears in *multiple* ui-related eligible patterns (i.e., in multiple cohesive combinations

of consumed itemsets of user u), arguably this provides an even stronger signal of item i's relevance to u; hence, we chose to use a simple aggregation of the cosine values across *all* eligible patterns to empirically show the benefits of using cosine patterns for long tail recommendation. This scoring approach is not only easy to implement and computationally scalable, but also demonstrates excellent recommendation performance (as shown by our experimental evaluation). In-depth analysis of optimal recommendation score aggregation across multiple patterns represents an interesting direction for future work.

We call the above approach *COsine pattern-based REcommendation*, or CORE for short. As will be demonstrated in the evaluation section, by leveraging the anti-cross-support property of cosine patterns, CORE not only exhibits good recommendation accuracy, but is also able to successfully recommend long-tail items. Moreover, the two thresholds for cosine pattern mining (i.e., τ_s and τ_c) provide CORE with flexibility in recommending items across the popularity distribution; e.g., to recommend more popular items we can set a high τ_s and a moderate τ_c, while to recommend more niche items we can set a small τ_s but a high τ_c.

3.4 Cosine-Pattern Tree Traversal Approach

The key computational challenge of the proposed cosine-pattern-based recommendation process is finding ui-related eligible patterns for given user u's all target items i. To deal with this, we propose to use a data structure called Cosine-Pattern Tree to boost the eligible pattern discovery process, which leads to CORE+ (an enhanced version of CORE).

For given user u and target item $i \in T_u$, a simple way to compute $\mathcal{EP}_{ui}$ is to first find all cosine patterns that contain i, i.e., $\mathcal{CP}_i = \{P \in \mathcal{CP} | i \in P\}$, and then keep only those where the remaining items are covered by C_u, i.e., $\mathcal{EP}_{ui} = \{P \in \mathcal{CP}_i | P \setminus \{i\} \subseteq C_u\}$. The most time-consuming aspect is determining whether pattern $P \in \mathcal{CP}_i$ satisfies the latter condition. A straightforward way to do this is to examine each item in $P \setminus \{i\}$ to see whether it is contained in C_u. By storing C_u in a hash table, where time complexity to look up any item is $\Theta(1)$, the overall time complexity for checking all items in one pattern is $O(|P|)$, and it would take $O(\sum_{P \in \mathcal{CP}_i} |P|)$ to go through all candidates and identify all ui-related eligible patterns.

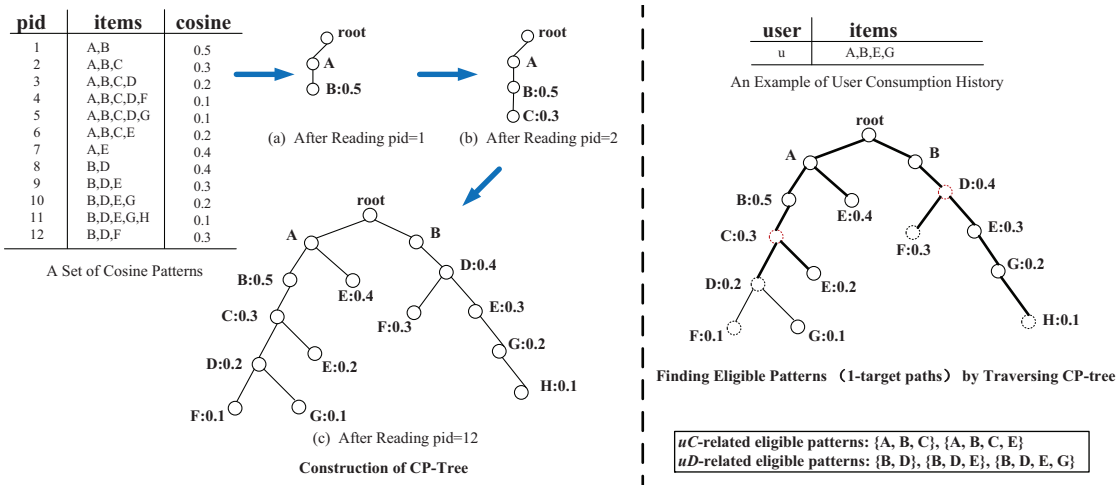


Figure 1: An example for CP-tree construction and traversal.

Under this strategy, two factors can slow down the recommendation process. First, the target item set (i.e., non-consumed items) for each user is typically very large; thus, time required to find candidate and eligible patterns for all items can add up quickly. Second, due to CAMP, a cosine pattern typically contains many similar cosine patterns as subsets, which would entail a lot repeated matching of highly similar patterns with C_u.

Both of these factors can be addressed by using advantageous data structures and algorithms for storing and retrieving cosine patterns. In particular, to facilitate efficient cosine pattern traversal and reduce redundant matching in eligible pattern detection, we build upon FP-tree (Han et al. 2000) and the ideas from the *CoPaMi* cosine pattern mining approach (Wu et al. 2014) to use *Cosine-Pattern Tree* (CP-tree). While FP-tree is proposed for compact storage of user consumption and efficient discovery of frequent patterns, CP-tree in our study is used for compact representation of discovered cosine patterns. Based on CP-tree, an intelligent, depth-first search strategy is designed to find all eligible patterns efficiently.

Generally, CP-tree is constructed in a similar fashion as FP-tree, i.e., it is a fully-connected, hierarchical tree structure, where each node represents an item. Each path from the root node to any other node in the CP-tree represents a cosine pattern (itemset) containing all the items on that path. Thus, cosine patterns that share a common prefix will share (a part of) the same path. More formally, CP-tree consists of two types of nodes: (i)

one special node denoting the start of all paths (i.e., all patterns stored in the tree) and labeled as *root*, and (ii) possibly multiple regular nodes denoting items, each labeled with a corresponding item name. Each node has three fields, storing its relevant information: `item` (i.e., the label of this node), `cosine` (i.e., the cosine value of the pattern that ends with this node), and `childlist` (i.e., the list of children nodes that are connected to this node).

Before tree construction, all items in each cosine pattern P are sorted in a support-ascending order, which guarantees that every prefix of P is also a cosine pattern, due to CAMP. E.g., if $supp(A) \leq supp(B) \leq supp(C)$ and $\{A, B, C\}$ is a cosine pattern, then $\{A, B\}$ is guaranteed to be a cosine pattern, due to CAMP. An illustration of CP-tree construction process is shown on the left side of Fig. 1, where all items are assumed to be sorted in the support-ascending order. Initially, the CP-tree contains only the default root node. After reading the first cosine pattern $\{A, B\}$, the nodes labeled as A and B are created, and path $root \rightarrow A \rightarrow B$ is then added, and the value of $cos(\{A, B\}) = 0.5$ is saved in the `cosine` field of node B, as shown in Fig. 1a. The second cosine pattern $\{A, B, C\}$ shares a common prefix $\{A, B\}$ with the first pattern, and therefore only one new node marked as C is added to the end of path $root \rightarrow A \rightarrow B$ with the corresponding $cos(\{A, B, C\}) = 0.3$ value in Fig. 1b. This process continues until every cosine pattern has been mapped onto one of the paths in CP-tree, which leads to the final CP-tree in Fig. 1c. Due to this specific prefix-based tree construction as well as the CAMP property of the cosine measure, all distinct cosine patterns are represented by all the paths from the root node to *every* node that is below the first two levels of CP-tree (since a cosine pattern must have at least two items).

Thus, any arbitrary path $[P] = root \rightarrow i_1 \rightarrow \cdots \rightarrow i_p$ in CP-tree represents cosine pattern $P = \{i_1, \cdots, i_p\}$, when $|P| \geq 2$. With respect to specific user u, any given path $[P]$ (and its corresponding cosine pattern P) can be classified into three different categories – 0-target path, 1-target path, and multi-target path – depending on how many target items the path contains. That is, $[P]$ is a *0-target path* if $|P \setminus C_u| = 0$ or, equivalently, if $P \subseteq C_u$, i.e., the path does not contain any target items for user u; $[P]$ is a *1-target path* if $|P \setminus C_u| = 1$;

or, $[P]$ is a *multi-target path* if $|P \setminus C_u| \geq 2$, i.e., if it contains multiple target items for user u. Importantly, note that only 1-target paths represent ui-related eligible cosine patterns that can be used for recommendation, as they contain exactly one target item.

The above categorization suggests an intelligent and highly efficient computational strategy that allows, for any user u, to find all relevant target items and calculate their recommendation scores with a *single traversal through CP-tree*. Intuitively, the main idea is to traverse CP-tree, visiting nodes in a depth-first manner. Each visited node represents path $[P]$ (i.e., path from the root to this node), which can be one of the following: (i) 0-target path, in which case no recommendation decisions need to be made, and the depth-first search continues to the children of this node; (ii) 1-target path, in which case the recommendation score for target item i (that is contained in $[P]$) is increased by $cos(P)$, and the depth-first search continues to the children of this node; or (iii) multi-target path, in which case no recommendation decisions need to be made, and the depth-first search is stopped along this path, as all subsequent extensions of path $[P]$ will continue to be multi-target paths. In summary, the proposed approach does not have to check all possible target items, but rather finds them (and calculates their recommendation scores) organically in one shot by efficiently browsing cosine patterns and matching them with each user's consumption history.

Algorithm 1 provides more detailed overview of the implementation of the proposed search-and-scoring routine. The main function `SearchCP` (Lines 1-23) employs a depth-first search on CP-tree. Besides variable *cur* to indicate the current node, variable *target* is introduced to indicate the target item contained in the current eligible pattern. Traversal along any given path in CP-tree terminates as soon as the path becomes a multi-target path (Lines 13-14), which avoids unnecessary search of longer patterns. Last-in-first-out stack S is used to execute the depth-first traversal, and variable *target* corresponding to target item contained in the current path is pushed into (or popped out from) the stack simultaneously with each node (Lines 6, 8, 12, 17, 22). Lines 11 and 21 update the recommendation scores of target items whenever the traversed path is a 1-target path.

Algorithm 1 Eligible-pattern searching and target-item scoring from CP-tree.

Input: $root$, C_u ▷ $root$ is the root node of CP-tree, C_u is the consumption history of user u
Output: $score$ ▷ list of recommendation scores of all items for user u

```
1: procedure SearchCP(root, C_u)
2:     S := ∅;                                              ▷ S: a last-in-first-out stack
3:     for i ∈ I do                                         ▷ I: the set of all items
4:         score(i) := 0;                                   ▷ initialization of recommendation score for each item
5:     target := null                                       ▷ detected target item in the beginning is null
6:     PushChildNodes(S, root, target);
7:     while S ≠ ∅ do                                       ▷ continue traversing until the stack is empty
8:         (cur, target) := S.POP;                          ▷ pop out the top tuple (current node, target item) in S
9:         if target is not null then                       ▷ check whether a target item is already detected
10:            if cur.item ∈ C_u then                       ▷ check whether current item is in user's consumption history
11:                score(target) := score(target) + cur.cosine;   ▷ update recommendation score of the target item
12:                PushChildNodes(S, cur, target);          ▷ push current node's children and current target item into the stack
13:            else
14:                continue                                 ▷ traversal of a path stops when a second target item is detected
15:        else                                             ▷ in case no target item has been detected so far
16:            if cur.item ∈ C_u then
17:                PushChildNodes(S, cur, target);          ▷ keep traversing when no target item is detected
18:            else
19:                target := cur.item;                      ▷ update the value of taget when a target item is detected
20:                if cur is not child of root then         ▷ update score only when the target item locates beyond the first level
21:                    score(target) := score(target) + cur.cosine;
22:                PushChildNodes(S, cur, target);          ▷ push current node's children and updated target item into the stack
23: end procedure
24: procedure PushChildNodes(S, cur, target)
25:     for node ∈ cur.childlist do
26:         S.Push(node, target);                            ▷ push current node's children and detected target item into the stack
27: end procedure
```

The right side of Fig. 1 illustrates the use of Alg. 1 for user u with specific consumption history C_u – the cosine values of patterns are shown next to the end nodes of their corresponding paths, the dashed nodes are target items (i.e., items that are not in C_u), and thick lines represent actual traversals performed by Alg. 1. Consider traversal along $root \rightarrow A \rightarrow B \rightarrow C$. Because C is the first item not contained in C_u, $root \rightarrow A \rightarrow B \rightarrow C$ becomes a 1-target path with C as its target item. This path can be extended further either with D or with E. In the case of former, traversal would stop at node D, as it is the second item not contained in C_u, making the path a multi-target path at that point; going deeper would not detect any new eligible patterns. This avoids redundant checks of successive nodes D, F, and G along the same path. In contrast, $root \rightarrow A \rightarrow B \rightarrow C \rightarrow E$ would be identified as another 1-target path with C as its target item. As another example, after traversal of path $root \rightarrow B \rightarrow D$, D is identified as another target item, and three 1-target paths with D as target item would be discovered: $root \rightarrow B \rightarrow D$, $root \rightarrow B \rightarrow D \rightarrow E$, and

$root \rightarrow B \rightarrow D \rightarrow E \rightarrow G$. Eventually, in this example, five eligible patterns that contribute to recommending C and D are identified with a single traversal of CP-tree.

CORE+ approach is highly computationally efficient. Space complexity of CP-tree is $O(|\mathcal{CP}|)$, since the total number of nodes in the tree equals the total number of cosine patterns plus a small fixed number (i.e., the root node and its immediate children). Meanwhile, given any user consumption history C_u, time complexity of finding all eligible patterns using Alg. 1 is $O(|\mathcal{CP}|)$. To elaborate, there are $O(|\mathcal{CP}|)$ nodes in the CP-tree and, in the extreme case, all of them may have to be examined, with constant amount of time needed per node; e.g., to check whether an item is contained in C_u is $\Theta(1)$ using a hash table. Therefore, with CP-tree and Alg. 1, the time complexity of recommendation given C_u reduces substantially from $O(|T_u| \sum_{P \in \mathcal{CP}} |P|)$ to $O(|\mathcal{CP}|)$, where $|P|$ is the number of items in pattern P. This implies that the upper bound of the running time of CORE+ is simply proportional to the number of cosine patterns, regardless of their size or the number of potential target items.

3.5 Parallelizing the Proposed Approach

We propose a parallelization framework, CORE++, based on *CP-tree partitioning* to further speed up large-scale recommendation tasks. By taking a complete CP-tree as a *forest* with a null root node, each subtree rooted at the second level can then be allocated to one computational node, and thus the entire CP-tree can be stored separately on Z available computational nodes. The parallelized CP-tree based recommendation could then be done in three stages: (i) broadcast user u's consumption history C_u to Z computational nodes; (ii) run Alg. 1 for u on Z nodes in parallel; (iii) aggregate local scores for each target item to obtain the final recommendation score. The time to generate recommendations for each user hinges on the ability to balance the workload (i.e., assign subtrees) among the available computational nodes. The details of this *load-balanced partitioning* problem and proposed solutions are provided in Appendix A3 of Online Supplement, due to space limitations.

Table 3: Summary statistics of datasets.

	#Users	#Items	#Ratings	Sparsity(%)	Avg.#Ratings per Item	% of Ratings from Top 10% Items	% of Ratings from Top 20% Items	% of Ratings from Top 50% Items	Gini Coefficient
`Book-Crossing`	1834	2172	41,337	98.96	13	30%	40%	65%	0.37
`Last.fm`	635	4100	14,175	99.46	5	40%	53%	70%	0.57
`Yelp`	5782	4762	154,190	99.44	26	42%	60%	85%	0.54
`MovieLens`	943	1682	100,000	93.70	59	43%	65%	93%	0.63

Sparsity=100(1−#Ratings/(#Users×#Items)), i.e., percentage of unknown ratings.

4 Evaluation

4.1 Experimental Setup

Data. CORE is tested using four publicly available datasets[8] that are widely used in recommender systems research: `Book-Crossing`, `Last.fm`, `Yelp`, and `MovieLens`. These datasets represent a variety of application domains (books, music, restaurants, movies), different levels of data sparsity, and different item consumption distributions. For `Book-Crossing`, `Last.fm`, and `Yelp`, we sample the data to include users with at least ten ratings to avoid extreme sparsity; even then these three datasets represent a much sparser consumption environment (about 99.0-99.5% data sparsity) than the one of `MovieLens` (93.7%). The rating distribution graphs (provided in Appendix A4 of the Online Supplement) show that all four datasets largely follow a long-tail distribution pattern, i.e., where a significant portion of item consumptions come from a smaller population of high-consumption (popular) items, and the remaining consumptions are from a much larger population of low-consumption items.[9] Table 3 provides the summary statistics of the datasets, including information about the distributional inequality in item consumption. E.g., if we consider the most popular 20% of items in each dataset, in `MovieLens` such items receive over 65% of all ratings, while the numbers are lower for the other datasets, i.e., 40%, 53%, and 60%, indicating a heavier-tailed nature of `Book-Crossing`, `Last.fm`, and `Yelp` datasets. Gini Coefficient[10] values in Table 3 further highlight the dataset differences in terms of item distributions.

[8]Available at `http://grouplens.org/datasets/hetrec-2011/` and `https://www.kaggle.com/datasets/yelp-dataset/yelp-dataset/versions/1?resource=download`.

[9]While the Pareto principle, i.e., 80/20 rule, has been widely used to describe such distributions, the term is often used loosely; that is, the specific cutoff between the supposed *head* and *long tail* of the distribution is often arbitrary and/or application-specific (Anderson 2006).

[10]Gini Coefficient is a distributional inequality (or concentration) metric, where a value of 0 reflects perfect equality, i.e., all items have equal amount of consumptions, while a value of 1 reflects maximal inequality among values (i.e., maximal consumption concentration, where all consumptions are from a single item).

Performance metrics. Although the main goal of the proposed approach is addressing the long-tail recommendation challenges, recommendation accuracy is always an important performance dimension. Precision (or precision-in-top-k) is commonly used to evaluate the accuracy of top-K recommendation lists and is calculated as follows. For each user u,

$$Precision_u = \#hits_u/K, \tag{2}$$

where $\#hits_u$ is the number of items from user u's recommendation list that are also in u's test set, and K is the size of the recommendation list (by default, $K = 10$ in our experiments).

As robustness checks for CORE's accuracy performance, we also used adjusted, rating-aware *Precision* metric as well as two other recommendation accuracy metrics: Normalized Discounted Cumulative Gain (NDCG) and Average Reciprocal Hit Rank (ARHR). Details are in Appendix A6 of the Online Supplement.

To evaluate the long-tail performance of recommendation algorithms, we use two metrics: (i) the average popularity (*AvgPop*) of items in the top-K recommendation list, where each item's popularity is reflected by the number of ratings it has (Yin et al. 2012, Niemann and Wolpers 2013), and (ii) the ratio of niche items (*NicheRatio*) in the top-K recommendation list (Krishnan et al. 2018). In a given dataset, an item is defined to be a *niche* (or *long-tail*) item if it has fewer ratings than some predetermined threshold. In the main paper, we use *average item frequency* in each dataset as the threshold for niche items. As robustness checks for CORE's long-tail recommendation performance, in Appendix A10 of the Online Supplement, we provide results using three alternative *percentile-based* thresholds to delineate popular vs. long-tail items for the *NicheRatio* metric. More formally, for each user u,

$$AvgPop_u = (\sum_{n=1}^{K} \#ItemRatings_n)_u/K, \qquad NicheRatio_u = \#NicheItems_u/K. \tag{3}$$

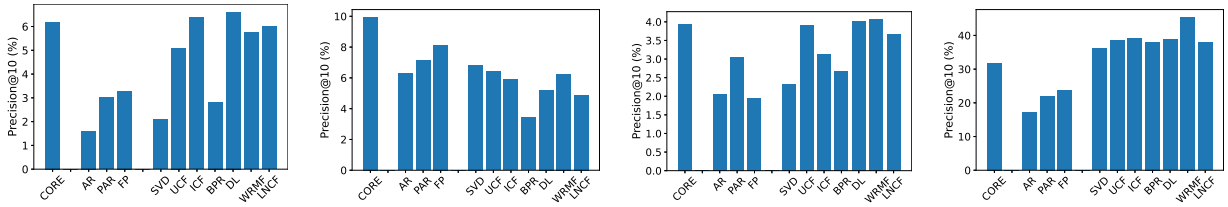
The overall performance for each metric is obtained by averaging its values over all users.

Baselines. We compare CORE with two types of baselines, i.e., pattern-based and collaborative filtering (CF) methods. The former includes the association rule-based method (AR) (Lin et al. 2002), probabilistic association rule-based method (PAR) (Ghoshal et al.

2015), and the frequent pattern-based method (FP) (Nakagawa and Mobasher 2003). The CF category includes six widely adopted methods: UCF (User-based CF) (Resnick et al. 1994), ICF (Item-based CF) (Sarwar et al. 2001), SVD (Funk 2006), WRMF (Weighted Regularized Matrix Factorization) (Hu et al. 2008), BPR (Bayesian Personalized Ranking) (Rendle et al. 2009), and DL (Deep Learning, specifically based on variational autoencoders) (Liang et al. 2018). We also include LNCF (Long-tail oriented Neural CF) (Krishnan et al. 2018), which is adapted from DL (Hu et al. 2008) specifically for long-tail recommendation tasks. Note that CORE as well as most other baselines (AR, FP, BPR, WRMF, DL, LNCF) are designed specifically for implicit feedback data, i.e., 0/1 data that reflects only whether a user consumed or purchased an item, and not the user's explicit preference rating for that item. Thus, for proper comparison across different baselines, we first convert explicit ratings in the four datasets to consumption (i.e., 0/1) data, based on the absence/presence of user rating. We then adopt the standard split-validation method commonly used in recommender systems research to evaluate different methods; that is, we randomly select 70% of the consumed items of each user for model training purposes, and use the remaining 30% of items for performance evaluation. Hyperparameters of each algorithm – the neighborhood size in UCF and ICF, the number of latent factors in SVD, WRMF, and BPR, learning rate in DL and LNCF, the support and cosine thresholds for CORE, etc. – were carefully tuned using standard predictive modeling practices for best accuracy performance.

4.2 Recommendation Accuracy Performance

Fig. 2 compares CORE to different baselines in terms of precision-in-top-10 (30% of data is used for testing). With respect to other pattern-based approaches, the results show that the accuracy of CORE is consistently higher than that of AR, PAR, and FP across all datasets. Compared to CF methods, Fig. 2 shows that CORE is highly competitive with the best-performing techniques on `Book-Crossing`, `Yelp`, and `Last.fm` (and, in fact, shows best performance among all methods on `Last.fm`). On `MovieLens`, CF baselines show some performance advantages over CORE, and we will take a closer look at this later in the paper.



(a) `Book-Crossing` (b) `Last.fm` (c) `Yelp` (d) `MovieLens`

Figure 2: CORE vs. baseline algorithms on accuracy (Precision@10, test-set ratio = 30%).

Table 4: Average popularity of recommended items.

Data Set	CORE	AR	PAR	FP	UCF	ICF	SVD	WRMF	BPR	DL	LNCF
`Book-Crossing`	**43**	76	80	101	72	45	69	65	60	61	57
`Last.fm`	**31**	89	82	107	61	52	34	48	96	51	45
`Yelp`	**276**	364	320	367	321	356	336	367	302	325	296
`MovieLens`	**135**	318	327	383	333	338	397	301	275	292	287

Best performance denoted in boldface.

To demonstrate robustness of the CORE performance, we provide comparisons of average true ratings of top-K recommended items (as an indicator of recommendation quality) across all methods as well as multiple additional accuracy comparisons: for different sizes of the top-K recommendation list, for adjusted (rating-aware) *Precision* metric, for two additional accuracy metrics (NDCG and ARHR), for different training-test split ratios, and for five-fold cross-validation-based evaluation in Appendices A5, A6, and A8 of the Online Supplement; the results show that the accuracy performance patterns across various methods remain consistent under different configurations.

Overall, accuracy comparisons provided here and in the Online Supplement demonstrate highly promising performance of CORE; i.e., it dominates pattern-based methods across all data sets and is highly competitive with the widely used CF-based approaches on sparser, heavier-tailed data sets, i.e., `Book-Crossing`, `Last.fm`, and `Yelp`.

4.3 Long-Tail Recommendation Performance

In this section, we go beyond recommendation accuracy and focus on the evaluation of long-tail recommendation performance, the key objective of the proposed approach. Tables 4 and 5 provide aggregate performance comparisons of the two long-tail metrics (AvgPop and NicheRatio) for top-10 items recommended by the same exact model configurations evaluated in Section 4.2. Again, the same 30% of data was used for testing.

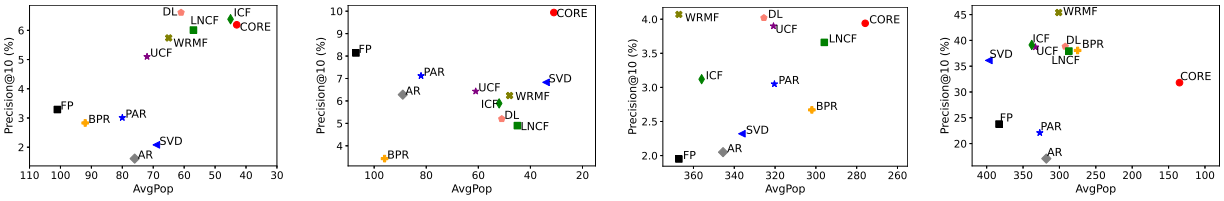
The results highlight the significant advantages of CORE over baseline algorithms for

Table 5: Percentage (%) of niche items recommended.

Data Set	Threshold	CORE	AR	PAR	FP	UCF	ICF	SVD	WRMF	BPR	DL	LNCF
`Book-Crossing`	13	**13.50**	0.05	0.13	0.04	2.10	1.38	1.81	3.22	0.40	3.23	6.13
`Last.fm`	5	**16.90**	1.72	2.23	0.02	2.16	4.44	5.77	9.30	0.20	3.11	8.20
`Yelp`	26	**5.01**	0.12	2.13	0.07	1.32	2.49	1.75	0.09	0.29	2.26	3.38
`MovieLens`	59	**25.35**	0.00	0.18	0.00	0.02	0.20	0.00	0.11	1.41	2.12	5.32

Threshold: threshold for niche item definition, i.e., average number of ratings per item in a data set.
Best performance denoted in boldface.

long-tail recommendation. Baseline algorithms tend to recommend popular items, as indicated by the average popularity metric and by the fact that, in the vast majority of settings, less than 2% of their recommendations are niche items. These results are consistent with previous studies (Fleder and Hosanagar 2009) showing that many existing recommender systems have *popularity bias*, creating rich-get-richer effects for popular items. In contrast, CORE is successfully able to provide recommendations of items with lower popularity and recommendations containing substantially higher percentage of niche items across all datasets.



(a) `Book-Crossing` (b) `Last.fm` (c) `Yelp` (d) `MovieLens`

Figure 3: CORE vs. baseline algorithms on accuracy and long-tail recommendation.

Fig. 3 summarizes the accuracy and long-tail performance results discussed earlier and compares the overall performance of different techniques in a two-dimensional (i.e., accuracy vs. long-tail) performance space. Specifically, Figs. 3a-d show each method's position in the *Precision-AvgPop* performance space for the four datasets.[11] Methods appearing in the top-right corner in these figures demonstrate better performance on both accuracy and long-tail recommendation. As shown in Fig. 3, for `Book-Crossing`, `Last.fm`, and `Yelp`, CORE is not only the advantageous choice in terms of the long-tail performance, but it is also an excellent overall choice on *both* performance dimensions. For `MovieLens` (Fig. 3d), CORE shows dramatic improvements in long-tail performance at the expense of some accuracy reduction with respect to CF baselines (but still significantly outperforming pattern-based ones), and

[11] Analogous results for *Precision-NicheRatio* space are provided in Appendix A7 of Online Supplement.

the final choice of an algorithm for such a setting may depend on the application-specific (or perhaps even user-specific) accuracy/long-tail performance trade-offs. In other words, the choice will depend on whether the increase in long-tail benefits of CORE outweighs the reduction in accuracy, as compared to best baseline CF algorithms. Overall, these results highlight the value of CORE in situations where recommending long-tail items is of importance (to the consumers, providers, or both). And, since traditional (purely accuracy-oriented) approaches are known to have challenges recommending niche items, it is important to have effective recommendation algorithms like CORE designed specifically for this task.

4.4 Additional Experiments

In this section, we discuss several additional important characteristics of the CORE approach.

Preference for Sparser, Heavier-Tailed Datasets. From the main results, we see that traditional pattern-based and CF methods tend to recommend items with higher popularity, while CORE is able to recommend substantially more long-tail items across all datasets. In terms of accuracy, on denser and more skewed datasets (such as `MovieLens`, where smaller percentage of popular items have higher percentage of ratings, as shown in Table 3), traditional CF techniques tend to have some inherent advantages. However, this advantage disappears on sparser, heavier-tailed data sets (such as `Book-Crossing`, `Last.fm` and `Yelp`), where CORE shows highly competitive accuracy performance.

We provide additional support for this finding in Appendix A9 of Online Supplement, where we compare CORE with WRMF (i.e., one of the CF baselines that demonstrates consistently good accuracy performance) on datasets with varying degrees of sparsity and skewness by removing top 10% to 50% most popular movies. The performance of CORE and WRMF on these datasets shows that, as the data gets sparser and less skewed, the accuracy gap between CORE and WRMF gradually narrows to the point where CORE actually starts to outperform WRMF. Meanwhile, the long-tail recommendation performance of CORE remains better than WRMF, albeit by a smaller margin, as there are much fewer popular items for the WRMF's popularity bias to manifest itself strongly. In summary,

this provides additional evidence that, aside from its superior long-tail recommendation performance across a wide variety of datasets, on sparser, heavier-tailed datasets CORE is able to demonstrate highly competitive performance in terms of accuracy as well.

Advantageous Recommendation for Niche Users. Even in the more challenging settings, such as the denser, less heavier-tailed datasets, CORE can be advantageous for certain user *subpopulations*. As an illustration, we compare the performance of WRMF and CORE for different groups of users in the `MovieLens` dataset, i.e., for the denser dataset where CORE (while maintaining very substantial long-tail performance advantages) was not as accurate as WRMF. Specifically, based on users' movie rating histories, we use two statistics – Average Number of Ratings of Rated Movies (*RatedAvgPop*) and Ratio of Niche Movies Rated (*RatedNicheRatio*)[12] – to group users based on their propensity to consume long-tail movies. We first calculated the two statistics for all users in the data and used deciles (i.e., 10th to 90th percentiles) as the thresholds to categorize users into 10 groups. The performance comparison of WRMF and CORE on the 10 groups of users is displayed in Tables 6 and 7. The results show that, for 40% of users who are more interested in long-tail movies, i.e., who consume movies of lower popularity on average or consume higher proportion of niche movies, CORE actually provides more relevant recommendations than WRMF. In other words, even in the settings that seem to be more challenging for CORE (such as `MovieLens` dataset), CORE can be more beneficial (than the baseline approaches) for a significant portion of population in terms of *both* accuracy and long-tailness.

Table 6: Performance comparison with *RatedAvgPop*-based user grouping for `MovieLens` data

User Group (decile)		1	2	3	4	5	6	7	8	9	10
Precision	WRMF	0.31	0.28	0.34	0.38	**0.41**	**0.50**	**0.57**	**0.54**	**0.62**	**0.58**
	CORE	**0.46**	**0.52**	**0.42**	**0.47**	0.33	0.33	0.33	0.28	0.22	0.25
AvgPop	WRMF	231	253	272	289	304	314	317	328	343	355
	CORE	**118**	**112**	**127**	**121**	**135**	**140**	**135**	**145**	**157**	**162**
NicheRatio	WRMF	0.20	0.19	0.18	0.11	0.11	0.09	0.08	0.08	0.05	0.03
	CORE	**28.51**	**27.95**	**26.73**	**26.20**	**26.11**	**25.19**	**25.01**	**23.67**	**22.47**	**21.63**

Group thresholds (i.e., 141, 154, 166, 177, 188, 199, 211, 226, 247) are based on deciles of all users' *RatedAvgPop*.
Users with lower group index are considered to have higher propensity of consuming long-tail movies.
Better performance for each group is denoted in boldface.

[12]Niche Movies in this analysis are defined the same way as in long-tail recommendation evaluation, i.e., movies with number of ratings less than the average number of ratings per movie in the dataset.

Table 7: Performance comparison with *RatedNicheRatio*-based grouping for `MovieLens` data

User Group (decile)		1	2	3	4	5	6	7	8	9	10
Precision	WRMF	0.30	**0.36**	0.42	0.44	**0.44**	**0.56**	**0.52**	**0.53**	**0.48**	**0.51**
	CORE	**0.38**	**0.36**	**0.46**	**0.45**	0.43	0.32	0.33	0.34	0.29	0.24
AvgPop	WRMF	251	255	274	286	297	310	312	326	346	340
	CORE	**132**	**127**	**128**	**134**	**129**	**137**	**129**	**141**	**150**	**147**
NicheRatio	WRMF	0.18	0.17	0.17	0.16	0.15	0.11	0.08	0.05	0.04	0.03
	CORE	**29.23**	**28.61**	**27.39**	**27.21**	**26.82**	**25.56**	**23.83**	**22.75**	**21.76**	**20.34**

Group thresholds (i.e., 4, 7, 10, 12, 15, 18, 21, 24, 30) are based on deciles of all users *RatedNicheRatio* (%).
Users with lower group index are considered to have higher propensity of consuming long-tail movies.
Better performance for each group is denoted in boldface.

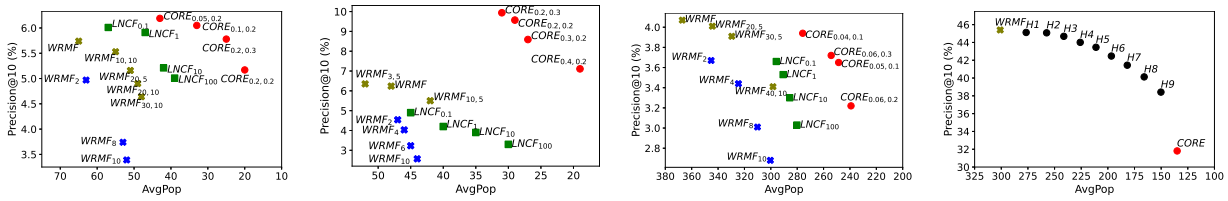
Flexible Recommendation. An important feature of the proposed CORE approach is that it allows fine-tuning of the popularity of recommended items in a flexible manner. I.e., as was mentioned in Section 3.3, the types of items that appear in the discovered cosine patterns can be easily tuned by setting cosine and/or support thresholds accordingly.

Figs. 4a-c show the performance of different variations of $\text{CORE}_{cos,supp}$, i.e., CORE under different cosine and support thresholds. For a given level of support, the popularity of recommended items tends to go down as the cosine threshold goes up. This can be seen from the fact that, for `Book-Crossing` data (Fig. 4a): $\text{AvgPop}(\text{CORE}_{0.05,0.2}) \geq \text{AvgPop}(\text{CORE}_{0.1,0.2}) \geq \text{AvgPop}(\text{CORE}_{0.2,0.2})$; for `Last.fm` (Fig. 4b): $\text{AvgPop}(\text{CORE}_{0.2,0.2}) \geq \text{AvgPop}(\text{CORE}_{0.3,0.2}) \geq \text{AvgPop}(\text{CORE}_{0.4,0.2})$; for `Yelp`, (Fig. 4c): $\text{AvgPop}(\text{CORE}_{0.04,0.1}) \geq \text{AvgPop}(\text{CORE}_{0.05,0.1})$).[13] The intuition is that a higher cosine threshold filters out more cross-support patterns containing high-frequency (popular) items. Also, for a given level of cosine, the popularity of recommended items tends to go up as the support threshold goes up. E.g., for `Book-Crossing` (Fig. 4a): $\text{AvgPop}(\text{CORE}_{0.2,0.2}) \leq \text{AvgPop}(\text{CORE}_{0.2,0.3})$; for `Last.fm` (Fig. 4b): $\text{AvgPop}(\text{CORE}_{0.2,0.2}) \leq \text{AvgPop}(\text{CORE}_{0.2,0.3})$; for `Yelp` (Fig. 4c): $\text{AvgPop}(\text{CORE}_{0.06,0.2}) \leq \text{AvgPop}(\text{CORE}_{0.06,0.3})$. With higher support threshold, only items in comparatively more frequent patterns (more popular items) would be recommended.

Not surprisingly, fine-tuning CORE for even better long-tail recommendation performance may come at the expense of some recommendation accuracy, as Figs. 4a-c show. Thus, in real-world applications, cosine and support thresholds should be set by the domain

[13] Section 4.4 presents long-tail performance results using the AvgPop metric. The results for NicheRatio are highly consistent, and are presented in Appendix A10 of Online Supplement due to the space limitations.

experts keeping in mind the specific application requirements. For example, considering that users exhibit highly heterogeneous, context-specific needs and preferences, it is valuable for the recommendation algorithms to provide flexibility, allowing users to adjust recommendations "on the fly" with well-designed user interfaces (Parra and Brusilovsky 2015, Bostandjiev et al. 2012). In case real-time adjustment of accuracy and long-tailness is not a feasible system design, the tradeoff could also be made empirically for a given application, e.g., through online field experimentation (i.e., A/B testing) (Gomez-Uribe and Hunt 2015, Amatriain and Basilico 2015). Specifically, platforms can provide recommendations with different proportions of niche items to multiple randomized groups of online consumers, quantify and compare the desired outcomes (e.g., user satisfaction) from all groups and determine the most advantageous levels of recommendation long-tailness. From the perspective of a specific platform, the tradeoff between accuracy and long-tailness of the displayed recommendations can also be made by weighing the economic value from each type of recommendation and adjusting (i.e., re-ranking) the recommendation lists accordingly (Jannach and Adomavicius 2017). In summary, a number of methodologies are already available that can be used for determining proper tradeoffs in multi-objective, application-specific recommendation settings. CORE can be used in conjunction with any of these methodologies, as it is a general-purpose long-tail recommendation approach specifically designed to provide the desired flexibility/adjustability that can match the needs of a wide variety of applications.



(a) `Book-Crossing` (b) `Last.fm` (c) `Yelp` (d) `MovieLens`

Figure 4: Two-dimensional (*Precision-AvgPop*) performance comparison of CORE and WRMF: (a)flexible parameterization for Book-Crossing data; (b)flexible parameterization for Last.fm data; (c)flexible parameterization for Yelp data; (d)hybridization for MovieLens data.

Comparisons with the Long-Tail-Oriented Baseline. Our main results demonstrated the benefits of CORE as compared to multiple popular, general-purpose baseline al-

gorithms. The benefits are especially prominent on the sparser, heavier-tailed datasets, where CORE shows advantages not only in long-tail performance, but in accuracy performance as well. Here we conduct an additional experiment to show that CORE's performance advantages remain even when compared to *specialized* long-tail-oriented baselines. Specifically, we compare CORE with three long-tail recommendation strategies: one general-purpose post-processing strategy that explicitly removes popular items from the recommendation list, one general-purpose advanced strategy that can build upon any existing recommendation approach (Park 2013), and one neural collaborative filtering methods designed specifically for long-tail tasks (Krishnan et al. 2018). We chose these approaches due to their adaptability to different recommendation techniques and their flexibility (somewhat similar to CORE's) for parameterizing the long-tail recommendations.

As with main experiments, we hold out 30% of each users' ratings as test data. Again, we chose to use the WRMF baseline in conjunction with the general-purpose long-tail recommendation strategies due to WRMF's consistently good performance across different datasets. To adapt WRMF for the post-processing strategy, we replace top γ% most popular items in the original top-K recommendations with subsequent high-ranked, less popular items. Particularly, we set $\gamma \in \{2, 4, 6, 8, 10\}$ for both datasets. To adapt WRMF for the more advanced strategy, all items in the training data (i.e., 70% of each user's ratings) are first pre-processed by partitioning them into head (H) and tail (T) groups using different rating frequency thresholds α (depending on the overall item frequency in a given dataset). Items with the number of ratings greater than α would be in group H, and the rest in group T. Those in group T are further clustered into β clusters as proposed in Park (2013). The above process guarantees a more balanced item rating distribution within each group and, thus, is intended to alleviate recommendation bias towards highly popular items. In particular, we set $\alpha \in \{10, 20, 30\}$ for `Book-Crossing`, $\alpha \in \{3, 5, 10\}$ for `Last.fm`, $\alpha \in \{20, 30, 40\}$ for `Yelp` and $\beta \in \{5, 10\}$ for all three datasets. Based on this pre-processing, recommendations are first generated using WRMF within H and each of β groups within T, and then aggregated

to form the final top-K recommendation list. In LNCF, a trade-off parameter λ was used to control the level of popularity of recommended items. Following the settings in Krishnan et al. (2018), we set $\lambda \in \{0.1, 1, 10, 100\}$ for all three datasets.

Figs. 4a-c compare different variations of $\text{CORE}_{cos,supp}$ (i.e., CORE under different cosine and support thresholds), WRMF_{γ} (i.e., WRMF under different post-processing settings), $\text{WRMF}_{\alpha,\beta}$ (i.e., WRMF under different pre-processing settings), and LNCF_{λ} (i.e., LNCF under different item popularity settings). Note that, although we run a number of CORE and WRMF variations, for clarity of visualization in Figs. 4a-c we display only the performance "frontiers" for CORE, WRMF_{γ}, $\text{WRMF}_{\alpha,\beta}$, and LNCF_{λ}; i.e., we do not display variations where the performance is dominated by some other variations in *both* recommendation accuracy and long-tail performance. Our results verify the effectiveness of the approaches proposed by Park (2013) and Krishnan et al. (2018) for boosting long-tail recommendation, i.e., they outperform the *naïve* post-processing strategy. Also, not surprisingly, better long-tail recommendation performance tends to come at the expense of recommendation accuracy, which applies to both CORE and long-tail-oriented baselines. Finally and importantly, CORE still provides advantageous performance in terms of both accuracy and long-tail recommendation for sparser, heavier-tailed datasets in comparison to other long-tail-oriented strategies, as illustrated by CORE's performance frontiers.

Benefits of Hybridization with CORE. While CORE provides advantages in both long-tail and accuracy performance on sparser, heavier-tailed data, on other kinds of datasets (e.g., `MovieLens`) no method strongly dominates on both performance dimensions. E.g., as shown in Fig. 3d, while CORE exhibits superior long-tail performance, it is WRMF that shows best accuracy (although underperforming significantly in long-tail recommendation). Depending on specific accuracy/long-tail performance trade-offs in a given application, one can obtain advantages by developing a *hybrid* recommender system.

As an illustration, we can hybridize WRMF and CORE in different ways by taking top-i recommended items from CORE and top-$(10 - i)$ recommended items from WRMF and

merging them into the final top-10 list. For any $i \in \{0, 1, \ldots, 10\}$, we denote the resulting hybrid method H_i. The two-dimensional performance comparison of original WRMF (i.e., H_0), CORE (i.e., H_{10}), and all hybrid methods (i.e., H_1, ..., H_9) is shown in Fig. 4d. Consider the performance of H_5, where taking top-5 items from each WRMF and CORE is able to get nearly half of the long-tail performance benefits of CORE (over what original WRMF showed) as well as the vast majority of accuracy benefits of WRMF (over what original CORE showed). This further highlights the practicality and value of CORE in achieving different recommendation goals, e.g., providing accurate (popular) recommendations for mainstream users and long-tail recommendations for variety-seeking, idiosyncratic, or contrarian users.

Scalability Demonstration: Hashtag Recommendation. We further demonstrate advantages of scalable and parallelizable CORE implementations (i.e., CORE+ and CORE++) on a large-scale hashtag recommendation task based on data from a social media platform. Due to space limitations, details are provided in Appendix A11 of Online Supplement.

5 Conclusions

Recommender systems have become an indispensable component of various online platforms, as they help users to find relevant content, products, or services more effectively. However, many traditional recommendation approaches (e.g., CF approaches) exhibit substantial *popularity bias*, i.e., recommendations tend to direct users' attention largely towards popular products. At the same time, it is widely acknowledged that long-tail recommendations can also be valuable, both for consumers and providers, as they better satisfy heterogeneous consumer needs and can lead to increase in demand, engagement, loyalty, and revenues.

However, due to the highly skewed distribution of consumed items and the fact that user preferences for idiosyncratic, less popular items are harder to predict, accurate recommendation of long-tail items remains a significant challenge. To address this challenge, we propose CORE, a novel recommendation approach that uses a special type of item co-occurrence patterns – *cosine* patterns – which are mined from users' consumption histories and are highly advantageous for recommendation purposes, especially for long-tail/niche items. CORE gen-

erates personalized recommendations by matching each user's consumption history against the discovered patterns. To ensure scalability of the proposed approach, we design a data structure for efficient recommendation generation (CORE+) and can further employ a parallel recommendation framework (CORE++).

In our experimental studies, we observe that cosine patterns indeed demonstrate the advantages of discovering more cohesive relationships among items, including niche items. We test the proposed approach (CORE) on four public datasets from different application domains and compare it to two types of baseline algorithms – pattern-based and CF-based – in terms of accuracy and long-tail recommendation performance. The results show that CORE dominates baseline approaches on long-tail recommendation. As importantly, CORE is also highly competitive on recommendation accuracy, either when used directly (e.g., especially on sparser, heavier-tailed datasets) or by offering "hybridization" opportunities for combining it with traditional top-accuracy baselines (e.g., on more skewed datasets). Finally, in addition to its high explainability, which is common to most pattern-based recommendation approaches, CORE demonstrates high flexibility, which provides the ability to fine-tune the system towards the desired popularity of recommended items, as well as high scalability, which can facilitate real-time recommendations in large-scale recommendation applications.

This study provides several directions for future research. As shown in the paper, skewness in item popularity impacts the discovered cosine patterns, which in turn affect the performance of the proposed approach. Developing a deeper theoretical understanding of the role that specific dataset characteristics play in the cosine-pattern-based recommendation performance would allow to further improve the effectiveness of pattern-based recommender systems. Also, the current approach uses the cosine metric to learn item patterns first, and then generates recommendations based on them. Bypassing the intermediate step of pattern generation and designing more direct (end-to-end) methods, e.g., using cosine information as part of a rating-prediction or learning-to-rank approach based on supervised machine learning methodologies, may lead to more performance benefits. Finally, under-

standing how long-tail recommendations may translate to business value gains is of great practical importance. Thus, empirically quantifying such value by measuring improvements in business-oriented (revenue, profit) or user-oriented (satisfaction, acceptance) metrics, e.g., through A/B testing methodologies, constitutes another promising direction for future work.

References

Abdollahpouri H, Mansoury M, Burke R, Mobasher B (2019) The unfairness of popularity bias in recommendation. *arXiv preprint arXiv:1907.13286* .

Adamopoulos P, Tuzhilin A (2014) On unexpectedness in recommender systems: Or how to better expect the unexpected. *ACM Trans. Intell. Syst. Technol.* 5(4).

Adomavicius G, Kwon Y (2012) Improving aggregate recommendation diversity using ranking-based techniques. *IEEE Trans. on Knowledge and Data Eng.* 24(5):896–911.

Adomavicius G, Tuzhilin A (2005) Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. *IEEE Transactions on Knowledge and Data Engineering* 17(6):734–749.

Agrawal R, Imielinski T, Swami AN (1993) Mining association rules between sets of items in large databases. *Proc. of the ACM SIGMOD Int. Conf. on Mgmt. of Data*, 207–216.

Alshammari G, Jorro-Aragoneses JL, Kapetanakis S, Petridis M, Recio-García JA, Díaz-Agudo B (2017) A hybrid CBR approach for the long tail problem in recommender systems. *Proc. of Int. Conf. on Case-Based Reasoning*, 35–45 (Springer).

Amatriain X, Basilico J (2015) Recommender systems in industry: A netflix case study. *Recommender systems handbook*, 385–419 (Springer).

Anderson A, Maystre L, Anderson I, Mehrotra R, Lalmas M (2020) Algorithmic effects on the diversity of consumption on spotify. *Proceedings of the web conference 2020*, 2155–2165.

Anderson C (2006) *The long tail: Why the future of business is selling less of more* (Hachette Books).

Bai B, Fan Y, Tan W, Zhang J (2017) Dltsr: A deep learning framework for recommendations of long-tail web services. *IEEE Transactions on Services Computing* 13(1):73–85.

Bostandjiev S, O'Donovan J, Höllerer T (2012) Tasteweights: a visual interactive hybrid recommender system. *Proceedings of the sixth ACM conference on Recommender systems*, 35–42.

Brynjolfsson E, Hu Y, Simester D (2011) Goodbye pareto principle, hello long tail: The effect of search costs on the concentration of product sales. *Management Science* 57(8):1373–1386.

Brynjolfsson E, Hu YJ, Smith MD (2006) From niches to riches: Anatomy of the long tail. *Sloan Management Review* 47(4):67–71.

Buskirk EV (2016) The most streamed music from spotify discover weekly. URL `https://insights.spotify.com/no/2016/07/07/top-music-discover-weekly/`.

Castells P, Hurley NJ, Vargas S (2015) Novelty and diversity in recommender systems. *Recommender Systems Handbook*, 881–918 (Springer).

Ceglar A, Roddick JF (2006) Association mining. *ACM Computing Surveys* 38(2):5.

Chen J, Wu W, Zheng W, He L (2021) Long-tail session-based recommendation from calibration. *arXiv preprint arXiv:2112.02581* .

Chen L, Zhang G, Zhou E (2018) Fast greedy map inference for determinantal point process to improve recommendation diversity. *Advances in Neural Information Processing Systems* 31.

Davidson J, Liebald B, Liu J, Nandy P, Vleet TV, Gargi U, Gupta S, He Y, Lambert M, Livingston B, Sampath D (2010) The youtube video recommendation system. *Proc. of the 2010 ACM Conf. on Recommender Systems*, 293–296.

Ferraro A (2019) Music cold-start and long-tail recommendation: bias in deep representations. *Proceedings of the 13th ACM Conference on Recommender Systems*, 586–590.

Fleder D, Hosanagar K (2009) Blockbuster culture's next rise or fall: The impact of recommender systems on sales diversity. *Management Science* 55(5):697–712.

Funk S (2006) Netflix update: Try this at home. URL `http://sifter.org/~simon/journal/20061211.html`.

Ghoshal A, Menon S, Sarkar S (2015) Recommendations using information from multiple association rules: A probabilistic approach. *Information Systems Research* 26(3):532–551.

Ghoshal A, Sarkar S (2014) Association rules for recommendations with multiple items. *INFORMS Journal on Computing* 26(3):433–448.

Goel S, Broder A, Gabrilovich E, Pang B (2010) Anatomy of the long tail: ordinary people with extraordinary tastes. *Proceedings of the third ACM international conference on Web search and data mining*, 201–210.

Goldstein DG, Goldstein DC (2006) Profiting from the long tail. *Harvard Bus. Rev.* 84(6):24–28.

Gomez-Uribe CA, Hunt N (2015) The netflix recommender system: Algorithms, business value, and innovation. *ACM Transactions on Management Information Systems (TMIS)* 6(4):1–19.

Hamedani EM, Kaedi M (2019) Recommending the long tail items through personalized diversification. *Knowledge-Based Systems* 164:348–357.

Han J, Pei J, Yin Y (2000) Mining frequent patterns without candidate generation. *Proc. of the 2000 ACM SIGMOD Int. Conf. on Management of Data*, 1–12.

Hijikata Y, Shimizu T, Nishida S (2009) Discovery-oriented collaborative filtering for improving user satisfaction. *Proc. of the 14th international conference on Intelligent user interfaces*, 67–76.

Hosanagar K, Fleder D, Lee D, Buja A (2013) Will the global village fracture into tribes? Recommender systems and their effects on consumer fragmentation. *Mgmt. Sci.* 60(4):805–823.

Hu Y, Koren Y, Volinsky C (2008) Collaborative filtering for implicit feedback datasets. *Proceedings of IEEE International Conference on Data Mining*, 263–272.

Jacobson K, Murali V, Newett E, Whitman B, Yon R (2016) Music personalization at spotify. *Proceedings of the 10th ACM Conference on Recommender Systems*, 373–373.

Jannach D, Adomavicius G (2017) Price and profit awareness in recommender systems. *arXiv preprint arXiv:1707.08029* .

Jannach D, Lerche L, Gedikli F, Bonnin G (2013) What recommenders recommend–an analysis of accuracy, popularity, and sales diversity effects. *Proc. of Int. Conf. on User Modeling, Adaptation, and Personalization*, 25–37 (Springer).

Koren Y, Bell R, Volinsky C (2009) Matrix factorization techniques for recommender systems. *Computer* 42(8):30–37.

Kotkov D, Konstan JA, Zhao Q, Veijalainen J (2018) Investigating serendipity in recommender systems based on real user feedback. *Proceedings of the 33rd annual acm symposium on applied computing*, 1341–1350.

Krishnan A, Sharma A, Sankar A, Sundaram H (2018) An adversarial approach to improve long-tail performance in neural collaborative filtering. *Proceedings of the 27th ACM International Conference on Information and Knowledge Management*, 1491–1494.

Lee D, Hosanagar K (2019) How do recommender systems affect sales diversity? a cross-category investigation via randomized field experiment. *Information Systems Research* 30(1):239–259.

Li M, Gan T, Liu M, Cheng Z, Yin J, Nie L (2019) Long-tail hashtag recommendation for micro-videos with graph convolutional network. *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*, 509–518.

Liang D, Krishnan RG, Hoffman MD, Jebara T (2018) Variational autoencoders for collaborative filtering. *Proceedings of the 2018 world wide web conference*, 689–698.

Lin W, Alvarez SA, Ruiz C (2002) Efficient adaptive-support association rule mining for recommender systems. *Data Mining and Knowledge Discovery* 6(1):83–105.

Lin Z, Goh KY, Heng CS (2017) The demand effects of product recommendation networks. *Mis Quarterly* 41(2):397–426.

Liu S, Zheng Y (2020) Long-tail session-based recommendation. *Fourteenth ACM conference on recommender systems*, 509–514.

Menon AK, Jayasumana S, Rawat AS, Jain H, Veit A, Kumar S (2020) Long-tail learning via logit adjustment. *arXiv preprint arXiv:2007.07314* .

Mobasher B, Dai H, Luo T, Nakagawa M (2001) Effective personalization based on association rule discovery from web usage data. *3rd Int. Wkshp. on Web Info. and Data Mgmt*, 9–15 (ACM).

Murakami T, Mori K, Orihara R (2007) Metrics for evaluating the serendipity of recommendation lists. *Proc. of Annual Conf. of the Japanese Society for Artificial Intelligence*, 40–46 (Springer).

Nakagawa M, Mobasher B (2003) A hybrid web personalization model based on site connectivity. *Proceedings of WebKDD*, 59–70.

Niemann K, Wolpers M (2013) A new collaborative filtering approach for increasing the aggregate diversity of recommender systems. *Proc. of the 19th ACM SIGKDD Int. Conf. on Knowl. Discovery and Data Mining*, 955–963 (ACM).

Paraschakis D, Nilsson BJ, Holländer J (2015) Comparative evaluation of top-n recommenders in e-commerce: An industrial perspective. *2015 IEEE 14th International Conference on Machine Learning and Applications*, 1024–1031.

Park YJ (2013) The adaptive clustering method for the long tail problem of recommender systems. *IEEE Trans. on Knowledge and Data Engineering* 25(8):1904–1915.

Parra D, Brusilovsky P (2015) User-controllable personalization: A case study with setfusion. *International Journal of Human-Computer Studies* 78:43–67.

Pathak B, Garfinkel R, Gopal RD, Venkatesan R, Yin F (2010) Empirical analysis of the impact of recommender systems on sales. *J of Mgmt. Info. Sys.* 27(2):159–188.

Pessemier EA (1978) Stochastic properties of changing preferences. *Am. Econ. Rev.* 68(2):380–385.

Qin J (2021) A survey of long-tail item recommendation methods. *Wireless Communications and Mobile Computing* 2021.

Qin J, Zhang Q, Wang B (2020) Recommendation method with focus on long tail items. *Journal of Computer Applications* 40(2):454–458.

Rabowsky B, Morrison L (2020) What's new in recommender systems. URL `https://aws.amazon.com/blogs/media/whats-new-in-recommender-systems/`.

Rendle S, Freudenthaler C, Gantner Z, Schmidt-Thieme L (2009) BPR: Bayesian personalized ranking from implicit feedback. *Proc. of 25th Conf. on Uncertainty in Artif. Intel.*, 452–461.

Resnick P, Iacovou N, Suchak M, Bergstrom P, Riedl J (1994) Grouplens: an open architecture for collaborative filtering of netnews. *Proceedings of the 1994 ACM Conference on Computer Supported Cooperative Work*, 175–186 (ACM).

Ribeiro MT, Ziviani N, Moura ESD, Hata I, Lacerda A, Veloso A (2015) Multiobjective pareto-efficient approaches for recommender systems. *ACM Trans. on Intel. Sys. and Tech.* 5(4):1–20.

Ricci F, Rokach L, Shapira B (2022) *Recommender Systems Handbook* (Springer).

Sarwar B, Karypis G, Konstan J, Riedl J (2001) Item-based collaborative filtering recommendation algorithms. *Proc. of the 10th Int. Conf. on WWW*, 285–295 (ACM).

Semerci O, Gruson A, Edwards C, Lacker B, Gibson C, Radosavljevic V (2019) Homepage personalization at spotify. *Proceedings of the 13th ACM conference on recommender systems*, 527–527.

Shi L (2013) Trading-off among accuracy, similarity, diversity, and long-tail: a graph-based recommendation approach. *Proc. of the 7th ACM Conf. on Recommender Systems*, 57–64 (ACM).

Sreepada RS, Patra BK (2020) Mitigating long tail effect in recommendations using few shot learning technique. *Expert Systems with Applications* 140:112887.

Tan PN, Kumar V, Srivastava J (2002) Selecting the right interestingness measure for association patterns. *Proc. of the 8th ACM Inter. Conf. on Knowledge Discovery and Data Mining*, 32–41.

Tan TF, Netessine S, Hitt L (2017) Is Tom Cruise threatened? an empirical study of the impact of product variety on demand concentration. *Information Systems Research* 28(3):643–660.

Tian M, Mehrotra R, Maystre L, Lalmas M (2019) Homepage and search personalization at spotify. *DMRN+ 14: Digital Music Research Network One-day Workshop 2019*.

Wang Y, Wu J, Wu Z, Yuan H, Zhang X (2014) Popular items or niche items: Flexible recommendation using cosine patterns. *IEEE ICDM Wkshp on Domain Driven Data Mining*, 205–212.

Wen B, Deng S, Chen H (2020) Knowledge-enhanced collaborative meta learner for long-tail recommendation. *China Conf. on Knowledge Graph and Semantic Computing*, 322–333 (Springer).

Wickramaratna K, Kubat M, Premaratne K (2009) Predicting missing items in shopping carts. *IEEE Transactions on Knowledge and Data Engineering* 21(7):985–998.

Wu J, Zhu S, Liu H, Xia G (2012) Cosine interesting pattern discovery. *Info. Sci.* 184(1):176–195.

Wu Z, Cao J, Wu J, Wang Y, Liu C (2014) Detecting genuine communities from large-scale social networks: a pattern-based method. *The Computer Journal* 57(9):1343–1357.

Xiong H, Tan PN, Kumar V (2006) Hyperclique pattern discovery. *Data Mining and Knowledge Discovery* 13(2):219–242.

Ye R, Hou Y, Lei T, Zhang Y, Zhang Q, Guo J, Wu H, Luo H (2021) Dynamic graph construction for improving diversity of recommendation. *Proceedings of the 15th ACM Conference on Recommender Systems*, 651–655.

Yin H, Cui B, Li J, Yao J, Chen C (2012) Challenging the long tail recommendation. *Proceedings of the VLDB Endowment* 5(9):896–907.

Zhang C, Hong X (2021) Challenging the long tail recommendation on heterogeneous information network. *2021 International Conf. on Data Mining Workshops (ICDMW)*, 94–101.

Zhang M, Hurley N (2008) Avoiding monotony: improving the diversity of recommendation lists. *Proc. of the 2008 ACM Conf. on Recommender Systems*, 123–130 (ACM).

Zhang M, Hurley N (2009) Novel item recommendation by user profile partitioning. *Proc. of the 2009 Int. Joint Conf. on Web Intelligence and Intelligent Agent Technology*, 508–515.

Zhang Y, Cheng DZ, Yao T, Yi X, Hong L, Chi EH (2021) A model of two tales: Dual transfer learning framework for improved long-tail item recommendation. *Proceedings of the Web Conference 2021*, 2220–2231.

Zhang YC, Séaghdha DÓ, Quercia D, Jambor T (2012) Auralist: introducing serendipity into music recommendation. *Proc. of the 5th ACM Int. Conf. on Web Search and Data Mining*, 13–22.

Zhou T, Kuscsik Z, Liu JG, Medo M, Wakeling JR, Zhang YC (2010) Solving the apparent diversity-accuracy dilemma of recommender systems. *Proc. of the Nat. Acad. of Sci.* 107(10):4511–4515.

Ziegler CN, McNee SM, Konstan JA, Lausen G (2005) Improving recommendation lists through topic diversification. *Proc. of the 14th Int. Conf. on WWW*, 22–32 (ACM).