Towards Scalable and Accurate Item-Oriented Recommendations

Noam Koenigstein
School of Electrical Eng., Tel Aviv University
noamk@eng.tau.ac.il

Yehuda Koren
Google, Israel
yehuda.a.koren@gmail.com

ABSTRACT

Most recommenders research aims at personalized systems, which suggest items based on user profiles. However, in reality many systems deal with item-oriented recommendations. In such setups, given a single item of interest, the system needs to provide other related items, following patterns like "people who liked this also liked...".

While item-oriented systems are central in their importance, they have been approached so far using very basic tools. We identify several hurdles faced by standard approaches to the item-oriented task. First, the sparseness of observed activities prevents establishing reliable similarity relations for many item pairs. Second, we address a scalability challenge at the retrieval stage present in many real-world systems: Given an item inventory, which may encompass millions of items, it is desired to identify the most related item pairs in a sub-quadratic time. This work addresses these two challenges, thereby improving both accuracy and scalability of item-oriented recommenders. Additionally, we propose an empirical evaluation scheme for comparing the quality of different solutions with encouraging results.

Categories and Subject Descriptors

H.2.8 [**Database Management**]: Database Applications—*Data Mining*

Keywords

collaborative filtering, item-based recommendation

1. ITEM BASED RECOMMENDATIONS

Most research on recommender systems is focused on modeling relations between users and items. New items are recommended to users based on their past purchases or activities. However, in reality user profiles are often not available (e.g., when a user is new or not logged-in) or irrelevant (e.g., when the current, short-term interest is unrelated to longer term inclinations). Therefore, many industrial systems are based on item-oriented recommendations, where item suggestions are solely based on their relatedness to a small set of currently considered items. Such recommendations aim at highlighting alternative items or items that are often bought together. A well known example of such a recommender is Amazon's shopping cart recommender. Amazon's system capitalizes on customers impulse buying patterns [10] – i.e., when a customer is currently considering a specific book, the system suggests to her more books that are often sold together. This real-world popular recommendation setup is characterized by inferring the recommendations based on the user's current interest rather than on her long term activity history. In general, these item-oriented systems employ signals like "users who liked this product also liked...", which are directly mined from usage logs.

While evidentially used in practice, we are not aware of many published scientific works addressing collaborative-filtering item-oriented recommendations. This is not very surprising as the problem seems rather simplistic compared to personalized recommendations, which indeed may use item-item relations internally. After all, one could argue that simply counting co-usage patterns among items can almost directly deliver the required similarity scores in the form of item-item conditional purchase probabilities, or other pairwise similarity metrics (e.g., cosine and Jaccard similarities, etc.). However, we wish to highlight two shortcomings of such straightforward retrieval techniques. First, pointwise counting of item-item co-occurrences requires an adequate support for both items. Hence, similarities related to items subject to less user activity cannot be estimated reliably. Second, a naïve retrieval of most recommended items requires evaluating all possible items for each target item. Therefore, computing all related item pairs requires a quadratic time which is hard to scale as item inventories grow in size. The scalability issue could be alleviated by different pruning rules or by caching results, yet we suggest a more systematic solution which accounts for both scalability challenges as well accuracy challenges.

We present Euclidean Item Recommender (EIR) which is a new method for identifying related item-pairs while addressing the two aforementioned shortcomings. Instead of determining item-item relations by the limited number of co-occurrences, we represent item-item conditional probabilities through latent factor vectors which are learned using a "global " learning algorithm. Our method is utilizing the entire training data rather than just the pairwise information. Hence, we achieve a smooth estimation of item-to-item relations that facilitates more reliable recommendations even

RecSys'13, October 12–16, 2013, Hong Kong, China.
ACM 978-1-4503-2409-0/13/10.
http://dx.doi.org/10.1145/2507157.2508221.

for rarely co-purchased item pairs. Furthermore, the latent factor vectors are embedded in a Euclidean space, thereby allowing a fast retrieval of most similar items using *metric trees*, without requiring the consideration of all possible item pairs. Finally, another contribution of this work is the establishment of a modeling and experimentation scheme for item-oriented recommendation, which employs existing user preference datasets.

2. RELATED WORK

Item-item neighborhood methods are well studied in the academic literature [10, 13]. We share with these works the need to derive pairwise relations between items. However, our research is item-oriented rather than user-oriented. Unlike previous work, we do not model the users in our dataset. We focus on modeling the item-item relations directly.

Some prior works [9, 12] used latent factors for representing item-item relations. This work also follows this path with several distinctions. First, we aim at item-oriented recommendation dictating a different cost function and training method. In addition, we emphasize fast retrieval which is facilitated by embedding the factor vectors in a Euclidean space, rather than the inner-product space used by others.

Our usage of Euclidean embedding shares some similarities with a recent work on Euclidean matrix factorization [7]. However we model item-item relations and not user-item relations. Beyond this, a fundamental distinction between the works is that we address the implicit feedback case and not explicit ratings, which is in fact the common case in most systems where explicit rating data is not present. Implicit feedback requires a different formulation based on maximizing likelihood rather than squared error minimization [6]. In addition, we propose a novel way of incorporating biases into the model without hindering the effectiveness of the fast retrieval.

3. MODELING PAIRWISE RELATIONS

We represent item-item relations through their conditional probabilities. That is, given items i and j we will estimate $P(j|i)$, the conditional probability that a user consuming item i will consume item j as well. The conditional probabilities will be learned by embedding all items in a low-dimensional Euclidean space. An item k will be represented by a d-dimensional vector $\mathbf{y}_k \in \mathbb{R}^d$, and a scalar bias b_k. The latent item vectors are designed to capture item similarities and the biases capture popularity patterns independently of other co-consumed items. To this end, we define the conditional probability $P(j|i)$ by the multinomial distribution

$$P(j|i) = \frac{\exp(-\|\mathbf{y}_i - \mathbf{y}_j\|^2 + b_j)}{\sum_k \exp(-\|\mathbf{y}_i - \mathbf{y}_k\|^2 + b_k)}. \quad (1)$$

Note that we assure $\sum_j P(j|i) = 1$. Finally, given a training set $\mathcal{D}$, containing item pairs of co-consumed (or co-liked) items, we seek to learn model parameters that maximize the log-likelihood of the training set:

$$\text{log-likelihood}\{\mathcal{D}\} \stackrel{\text{def}}{=} \sum_{(i,j)\in\mathcal{D}} \log P(j|i)\,. \quad (2)$$

3.1 Optimization process

Learning proceeds by stochastic gradient ascent. Given a training pair (i, j) we update each parameter θ (a latent vector component or a bias) by

$$\Delta\theta = \eta\frac{\partial P(j|i)}{\partial\theta} = \eta\left[\frac{\partial}{\partial\theta}\Big(-\|\mathbf{y}_i - \mathbf{y}_j\|^2 + b_j\Big) + \sum_k P(k|i)\frac{\partial}{\partial\theta}\Big(\|\mathbf{y}_i - \mathbf{y}_k\|^2 - b_k\Big)\right], \quad (3)$$

where η is the learning rate. However, such a training scheme would be too slow in practice as each update rule requires summing over all items. We thus resort to sampling the weighted sum in (3) based on the importance sampling idea proposed by Bengio and Senécal [2]; see also Aizenberg et al. [1] for a related usage of the technique.

With importance sampling we draw items according to a *proposal distribution*. In our case we assign each item a probability proportional to its empirical frequency in the training set (fraction of train pairs containing the item), and denote this proposal distribution by $P(i|\mathcal{D})$. Items are sampled with replacement from $P(i|\mathcal{D})$ into a list $\mathcal{L}$. Using $\mathcal{L}$, we approximate $P(k|i)$ for each $k \in \mathcal{L}$ with the weighting scheme

$$w(k|i) = \frac{\exp(-\|\mathbf{y}_i - \mathbf{y}_k\|^2 + b_k)/P(k|\mathcal{D})}{\sum_{l\in\mathcal{L}} \exp(-\|\mathbf{y}_i - \mathbf{y}_l\|^2 + b_l)/P(l|\mathcal{D})}. \quad (4)$$

Consequently, the approximated gradient ascent step given a training pair (i, j) will be

$$\Delta\theta = \eta\left[\frac{\partial}{\partial\theta}\Big(-\|\mathbf{y}_i - \mathbf{y}_j\|^2 + b_j\Big) + \sum_{k\in\mathcal{L}} w(k|i)\frac{\partial}{\partial\theta}\Big(\|\mathbf{y}_i - \mathbf{y}_k\|^2 - b_k\Big)\right]. \quad (5)$$

As mentioned in [2], it is desirable that the size of the set $\mathcal{L}$ grows as the training process proceeds because at later training phases more delicate parameter adjustments are needed. Hence, we employ a simple rule for controlling the sample size ($|\mathcal{L}|$) based on the fitness of the current estimate. Given a training pair (i, j), we keep sampling items into $\mathcal{L}$ until the following condition is satisfied:

$$\sum_{k\in\mathcal{L}} P(k|i) > \alpha \cdot P(j|i). \quad (6)$$

The adaptive sampling automatically lets the sample size grow when parameters are nearing final values and the correct paired item is getting a relatively high probability. In our implementation we used $\alpha = 3$. We impose a minimal size of 5 on the sample size. For efficiency we also limit the maximal sample size to 500.

4. FAST RETRIEVAL

Retrieval time of recommendations is a key factor when designing real-world large-scale systems that need to address tens of thousands of queries per second [8]. A major design goal of EIR is enabling fast pairing of items. In this setting, the task of efficiently retrieving recommendations requires finding items that the user is most likely to purchase given the item she is currently considering. Namely, we wish to find an item j that maximizes:

$$\max_{j\neq i} P(j|i) \quad \Leftrightarrow \quad \max_{j\neq i} -\|\mathbf{y}_i - \mathbf{y}_j\|^2 + b_j. \quad (7)$$

	Items	Users	Training	Test
Netflix	17,749	478,488	53,414,617	941,614
MSD	384,526	1,019,296	45,078,691	2,037,890
YMusic	433,903	497,881	40,362,704	984,162
Books	340,536	103,723	1,068,842	66,306

Table 1: The number of items, users, training-set examples and test-set examples in each dataset.

It is easy to see that by ignoring biases we are left only with the squared Euclidean distance between the two items vectors, and the retrieval is reduced to a simple nearest neighbor task. Nevertheless, biases are a key contributor to recommendations accuracy, and should not be dismissed. We therefore propose a simple transformation to reduce the problem in (7) to that of a simple Euclidean search. For each item vector $\mathbf{y}_j$, we define a concatenated item vector $\hat{\mathbf{y}}_j$ as follows:

$$\hat{\mathbf{y}}_j = [\mathbf{y}_j^\top, \sqrt{M_b - b_j}]^\top \in \mathbb{R}^{d+1}, \tag{8}$$

where M_b is the maximum bias ($M_b = \max_j b_j$). We also define a concatenated query vector as follows:

$$\bar{\mathbf{y}}_i = [\mathbf{y}_i^\top, 0]^\top. \tag{9}$$

It can be easily shown that (7) is equivalent to:

$$\max_{j \neq i} P(j|i) \quad \Leftrightarrow \quad \min_{j \neq i} \|\bar{\mathbf{y}}_i - \hat{\mathbf{y}}_j\|^2. \tag{10}$$

Therefore, this transformation facilitates a variety of Euclidean nearest neighbor algorithms e.g. Metric Trees, or Locality Sensitive Hashing (LSH).

5. EMPIRICAL STUDY

5.1 Dataset Construction

We evaluate our algorithm using four different datasets: Netflix [3], the Million Song Dataset (MSD) [4], Ziegler's books' reviews[1] (Books) [16], and Yahoo! Music (YMusic) [5]. Our work is focused on implicit ratings, however of the four aforementioned datasets only the MSD dataset is implicit. Therefore, we simulated imlicit data from the explicit datasets as follows: In Netflix, we first filtered only the ratings with a value ≥ 4. We then produced a dataset of "co-liked" movies – namely, movies that were liked by the same users. For the YMusic dataset, we repeated the same process for ratings ≥ 80. In the books dataset we simply used all the data-entries in order to create a dataset of co-consumed books that were read and reviewed by the same users (regardless of the review). Finally, in all datasets we generated item pairs as follows: For each user we created a random cyclic order of the items consumed by her. Then, the final dataset is comprised from all the pairs of consecutive items.

We split our datasets into train and test subsets by randomly choosing a subset of users and placing all their item pairs in the test-set. Table 1 summarize the final datasets statistics.

5.2 Baselines

We compared performance against traditional item-item similarity measurements:

- *Empirical Conditional Probability (ECP):* Empirical measurement of the conditional probability $P(j|i)$ defined as

$$P(j|i)_{empirical} = \frac{n_{i,j}}{n_i + 1},$$

where n_i is the total number of occurrences of item i in the dataset, and $n_{i,j}$ is the number co-occurrences of i and j together (counts were smoothed by adding 1).

- *Jaccard Similarity:* Jaccard similarity is a well know similarity measure defined as

$$Jaccard(i,j) = \frac{n_{i,j}}{n_i + n_j - n_{i,j}}.$$

- *Cosine Similarity:* Cosine similarity is another widely used similarity measure defined as

$$Cosine(i,j) = \frac{n_{i,j}}{\sqrt{n_i n_j}}.$$

Note that ranking based on these similarity measures differ in the way of normalizing the co-counts ($n_{i,j}$).

5.3 Accuracy Results

We measure performance in terms of Mean Percentile Rank (MPR). This metric was earlier used in studies of implicit feedback datasets within the context of personalized recommendations [6, 14]. In our item-oriented context, it is defined as follows: For each test pair of related items (i, j) we sample N additional random items. We rank every item k (the N random ones and j) with respect to $P(k|i)$ based on our model. Then, we compute the percentile rank of item j within this ranking. Ranks are averaged over all test pairs. Accordingly, percentile ranks closer to zero indicate better rankings. We used $N = 200$ in our experiments, though results are insensitive to changes in N given the large number of test pairs.

We trained the EIR model with $d = 50$ dimensions. Table 2 presents the MPR results of EIR against the baselines. We were encouraged by the fact that our algorithm outperformed the baselines on the Netflix, MSD and the Books datasets. On the YMusic dataset, our algorithm was second to ECP with a very small difference between the two.

Figure 1 depicts the mean percentile rank of the different algorithms vs. the support (popularity) of the conditioned item i in $P(j|i)$. While all the algorithms perform well on popular items, EIR has a clear advantage in the long tail. This is explained by the fact the EIR employs global optimization that utilize the entire training data rather than just the pairwise information. Among the different baseline algorithms, ECP is most similar to EIR. This is to be expected, as ECP is merely an empirical estimate of $P(j|i)$ – the objective of EIR.

5.4 Fast Retrieval Results

We evaluate the fast retrieval capabilities of EIR using metric trees [11, 15]. Metric trees are binary space-partitioning trees widely used for the task of indexing Euclidean datasets. The tree construction is very fast and space efficient, and the search employs the depth-first branch-and-bound algorithm similar to that of [15]. We quantify the improvement in retrieval time for the task described in (7) when using the

[1] `www.informatik.uni-freiburg.de/~cziegler/BX/`

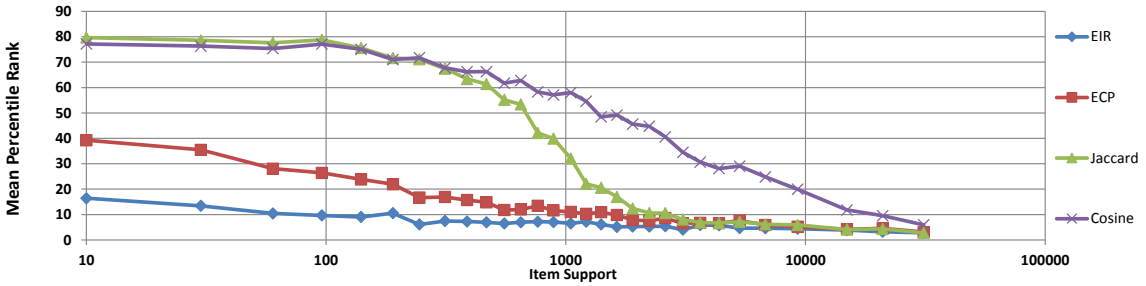

Figure 1: Mean Percentile Rank (MPR) vs. the support of the conditioned item i in $P(j|i)$ in the MSD dataset (lower is better).

	EIR	ECP	Jaccard	Cosine
Netflix	**5.825%**	6.57%	9.728%	8.376%
MSD	**6.602%**	13.663%	35.287%	49.18%
YMusic	2.837%	**2.536%**	9.795%	9.545%
Books	**27.342%**	35.902%	53.896%	61.011%

Table 2: Comparing MPR of EIR against common baselines (lower is better).

	Netflix	MSD	YMusic	Books
Speedup	25.036	21.422	31.219	11.218

Table 3: Speedup values for EIR with 50 dimensions

metric tree compared to a naïve search as follows:

$$\text{Speedup} = \frac{\text{Retrieval Time Using Naive Search}}{\text{Retrieval Time Using Tree}}. \quad (11)$$

The speedup values for each dataset are presented in Table 3.

6. SUMMARY

We tackle *item-oriented* recommendations, where the goal is to find items related to the current item. Our approach is centered around a method embedding items and their biases within a Euclidean space in a way that preserves co-consumption patterns. A unique feature of our embedding is its being Euclidean thereby facilitating fast item retrieval with readily available data structures suitable to the Euclidean space. We demonstrated the efficacy of the method in terms of both accuracy and indexing time.

7. REFERENCES

[1] N. Aizenberg, Y. Koren, and O. Somekh. Build your own music recommender by modeling internet radio streams. In *WWW*, 2012.

[2] Y. Bengio and J.-S. Senécal. Quick training of probabilistic neural nets by sampling. In *Proc. 9th International Workshop on Artificial Intelligence and Statistics (AISTATS'03)*, 2003.

[3] J. Bennett and S. Lanning. The netflix prize. In *Proc. KDD Cup and Workshop*, 2007.

[4] T. Bertin-Mahieux, D. P. Ellis, B. Whitman, and P. Lamere. The million song dataset. In *Proceedings of the 12th International Conference on Music Information Retrieval (ISMIR 2011)*, 2011.

[5] G. Dror, N. Koenigstein, Y. Koren, and M. Weimer. The yahoo! music dataset and kdd-cup'11. *Journal Of Machine Learning Research*, 18:3–18, 2012.

[6] Y. Hu, Y. Koren, and C. Volinsky. Collaborative filtering for implicit feedback datasets. In *ICDM*, pages 263–272, 2008.

[7] M. Khoshneshin and W. N. Street. Collaborative filtering via euclidean embedding. In *RecSys*, pages 87–94, 2010.

[8] N. Koenigstein, P. Ram, and Y. Shavitt. Efficient retrieval of recommendations in a matrix factorization framework. In *CIKM*, 2012.

[9] Y. Koren. Factor in the neighbors: Scalable and accurate collaborative filtering. *TKDD*, 4(1), 2010.

[10] G. Linden, B. Smith, and J. York. Industry report: Amazon.com recommendations: Item-to-item collaborative filtering. *IEEE Distributed Systems Online*, 4(1), 2003.

[11] F. P. Preparata and M. I. Shamos. *Computational Geometry: An Introduction*. Springer, 1985.

[12] S. Rendle, C. Freudenthaler, and L. Schmidt-Thieme. Factorizing personalized markov chains for next-basket recommendation. In *WWW*, pages 811–820, 2010.

[13] B. Sarwar, G. Karypis, J. Konstan, and J. Riedl. Item-based collaborative filtering recommendation algorithms. In *Proceedings of the 10th international conference on World Wide Web*, WWW '01, pages 285–295. ACM, 2001.

[14] H. Steck. Training and testing of recommender systems on data missing not at random. In *KDD*, pages 713–722, 2010.

[15] J. K. Uhlmann. Satisfying general proximity/similarity queries with metric trees. *Inf. Process. Lett.*, 40(4):175–179, 1991.

[16] C.-N. Ziegler, S. M. McNee, J. A. Konstan, and G. Lausen. Improving recommendation lists through topic diversification. In *Proceedings of the 14th international conference on World Wide Web*, WWW '05, pages 22–32, 2005.