Swarming to Rank for Recommender Systems

Ernesto Diaz-Aviles
diaz@L3S.de

Mihai Georgescu
georgescu@L3S.de

Wolfgang Nejdl
nejdl@L3S.de

L3S Research Center / University of Hannover, Germany

ABSTRACT

Recommender systems make product suggestions that are tailored to the user's individual needs and represent powerful means to combat information overload. In this paper, we focus on the item prediction task of Recommender Systems and present *SwarmRankCF*, a method to automatically optimize the performance quality of recommender systems using a Swarm Intelligence perspective. Our approach, which is well-founded in a Particle Swarm Optimization framework, learns a ranking function by optimizing the combination of unique characteristics (i.e., features) of users, items and their interactions. In particular, we build feature vectors from a factorization of the user-item interaction matrix, and directly optimize Mean Average Precision metric in order to learn a linear ranking model for personalized recommendations. Our experimental evaluation, on a real world online radio dataset, indicates that our approach is able to find ranking functions that significantly improve the performance of the system for the Top-N recommendation task.

Categories and Subject Descriptors: H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval–*Information Filtering*

General Terms: Algorithms, Experimentation, Measurement, Performance

Keywords: Collaborative Filtering; Matrix Factorization; PSO; Recommender Systems; Swarm Intelligence

1. INTRODUCTION

The information overload problem facing today's Web users has made the decision-making task really challenging, and in many cases complex to the user, who has to face an overwhelming set of options that outstrips her capability to survey them and reach a decision. How can recommendation systems help users to meet their particular information needs and preferences?

In practice this problem can be cast as a ranking problem, whose goal is to define an ordering among items (e.g., Web sites, news articles, songs, books, or movies) that ranks relevant ones in higher positions of the retrieved list.

In recent years supervised learning-based methods have been proposed to automatically learn an effective ranking model based on training data (e.g., [8, 6]). This task is referred to as "Learning To Rank for Information Retrieval". Learning to rank relies upon pre-engineered features that characterize the items to be ranked. Such a set of features can be difficult to compute and maintain.

In the absence of high quality *explicit features*, recommender systems can infer *latent factors* about users and items using matrix factorization algorithms for Collaborative Filtering (CF).

This paper presents an approach to learning ranking functions that exploits collaborative latent factors as features. To accomplish this, instead of manually creating an item feature vector, we factorize a matrix of user-item interactions, and use these collaborative latent factors as input to a Swarm Intelligence (SI) ranking method: SwarmRank [5]. We focus in this paper on the item prediction or Top-N recommendation problem in the context of recommender systems, which we consider a harder prediction problem than rating prediction, as only positive feedback is available.

SwarmRank was chosen because it is based on Particle Swarm Optimization (PSO) [11], a global non-linear optimization algorithm inspired in the behavior of biological organisms. PSO does not require, nor approximate, gradients of the cost function, and its resilience to local minima allows it to directly optimize non-smooth Information Retrieval (IR) measures, such as MAP (Mean Average Precision).

Contributions. The main contributions of this paper are as follows: (1) We present SwarmRankCF, a model suitable for the personalized item recommendation task, that seamlessly integrates a SI learning to rank method with a collaborative filtering approach, to derive a personalized ranking function optimized for the ranking task. (2) We present an empirical study on the public dataset *Last.fm Dataset – 1K users* obtained from Last.fm[1], a major social media Internet radio station, demonstrating the superior recommendation performance of SwarmRankCF.

2. BACKGROUND

In this section we introduce the task of learning to rank for IR and provide a background on ranking with swarm intelligence. We also include a brief review of particle swarm optimization and matrix factorization for recommender systems.

2.1 Learning to Rank for IR

Learning to rank for Information Retrieval [10] is a problem formalized as follows. In learning (training), a collection of queries and their corresponding retrieved documents are given. Furthermore, the labels (i.e., relevance judgments) of the document with respect to the queries are also provided. The relevance judgments, provided by human annotators, can represent ranks (e.g., categories in a total order). The objective of learning is to construct a ranking model, e.g., a ranking function, that achieves the best performance on test data.

[1] **Last.fm**: http://www.last.fm

RecSys'12, September 9–13, 2012, Dublin, Ireland.

In retrieval (test phase), given a query, the learned ranking function is applied, returning a ranked list of documents in descending order of their relevance scores. Suppose that $Q = \{q_1, \cdots, q_{|Q|}\}$ is the set of queries and $D = \{d_1, \cdots, d_{|D|}\}$ the set of documents, the training set is created as a set of query-document pairs, $(q_i, d_j) \in Q \times D$, upon which a relevance judgement (e.g., a label) indicating the relationship between q_i and d_j is assigned by an annotator. Suppose that $Y = \{y_1, \cdots, y_{|Y|}\}$ is the set of labels and $y_{ij} \in Y$ denotes the label of query-document pair (q_i, d_j). A feature vector $\phi(q_i, d_j)$ is created from each query-document pair $(q_i, d_j), i = 1, 2, \cdots, |Q|; j = 1, 2, \cdots, |D|$. The training set is denoted as $T = \{(q_i, d_j), \phi(q_i, d_j), y_{ij}\}$. The ranking model is a real valued function of features:

$$f(q, d) = \vec{p_g} \cdot \phi(q, d) \quad (1)$$

where $\vec{p_g}$ denotes a weight vector. In ranking, for query q_i the model associates a score to each of the documents d_j as their degree of relevance with respect to query q_i using $f(q_i, d_j)$, and sort the documents based on their scores.

2.2 Ranking with Swarm Intelligence

Particle Swarm Optimization

The particle swarm optimization algorithm (PSO) is a population-based probabilistic optimization algorithm inspired by the social behavior of biological organisms, specifically the ability of groups of species of animals to work as a whole in locating desirable positions in a given area. In a PSO algorithm, a population of agents called $particles$ move through the solution space of an optimization problem, updating their velocity according to the information collected by the group called $swarm$. PSO algorithms are global non-linear optimization algorithms and do not require nor approximate gradients of the cost function.

In every iteration, each particle is attracted to the best solution that it has found individually, and toward the best solution that any particle in their $neighborhood$ has found. In PSO, a neighborhood is defined for each individual particle as the subset of particles which is able to communicate with. The algorithm updates the entire swarm at each time step by updating the velocity and position of each particle in every dimension. An in-depth introduction to PSO is given in [11].

SwarmRank: SI-Based Learning to Rank

$SwarmRank$ [5] is a PSO-based method to handle the task of learning to rank for IR. The goal of SwarmRank is the discovery of good ranking formulas more adapted to the particularities of a specific collection, which are also able to generalize well beyond the training data. The procedure examines important information retrieval features extracted from query-document pairs instances and learns a linear function that combines them optimally.

SwarmRank is an iterative process that learns linear ranking functions of the form of Eq. (1), where $\vec{p_g}$ is the weight vector corresponding to the best global solution found by the $swarm$ in PSO. The function represents a linear combination of the query-document pairs feature vectors which associates a real value to each query-document pair as their degree of relevance.

SwarmRank takes as an input the query-document pairs and their corresponding feature vectors. As an instance of PSO, it quantifies the optimality of a solution by means of a fitness function, which serves as a criteria to update the particle and global best solutions. As output, the procedure returns a linear ranking function whose coefficients are given by the dimension values corresponding to the global best solution vector found by the swarm.

Fitness Function. Since the particles represent solutions in terms of weight vectors to be used in a document ranking function, the fitness function measures the quality of the ranking generated by a given particle. SwarmRank takes advantage of the non-linear optimization capabilities of PSO and directly optimizes a non-smooth IR measure, namely Mean Average Precision–MAP.

2.3 Recommender Systems

Let $U = \{u_1, \ldots, u_{|U|}\}$ and $I = \{i_1, \ldots, i_{|I|}\}$ be the sets of all users and all items, respectively. Suppose we are dealing with interactions between these two entities, and for some user $u \in U$ and item $i \in I$ we observe a *relational score* r_{ui}. Thus each instance of the data is a tuple (u, i, r_{ui}), which for example, in the movie recommendation case might correspond to an explicit "rating" given by user u to movie i, or in the case of music recommendation to a "weight" implicitly derived from user u's listening patterns, e.g., how many times the user u has listened to song i. Collaborative Filtering (CF) is a technique used by some recommender systems, whose goal is to make automatic predictions (filtering) about the interests of a user by collecting preferences or taste information from many users (collaborating). Typical CF organizes these tuples into a sparse matrix $\mathbf{R}$ of size $|U| \times |I|$, using (u, i) as index and r_{ui} as entry value. The task of the recommender system is to estimate the score for the missing entries.

Matrix Factorization

Low dimensional linear factor models [9] are one of the most effective approaches for CF. Factor-based algorithms consider that only a small number of *latent* factors can influence the preferences. Their prediction is a real number $\hat{r}_{ui}$ per user item pair (u, i). Some of the most successful realizations of latent factor models are based on *matrix factorization*. In its basic form, matrix factorization estimates a matrix $\hat{\mathbf{R}}$ of size $|U| \times |I|$ by the product of two low-rank matrices $\mathbf{U} : |U| \times k$ and $\mathbf{I} : |I| \times k$:

$$\hat{\mathbf{R}} := \mathbf{U}\ \mathbf{I}^T\ , \quad (2)$$

where k is a parameter corresponding to the rank of the approximation. Each row $\mathbf{U}_u$ in $\mathbf{U}$ and row $\mathbf{I}_i$ in $\mathbf{I}$ can be considered as a feature vector describing a user u and an item i, correspondingly. Thus the final prediction is the linear combination of the factors:

$$\hat{r}_{ui} = \mathbf{U}_u \cdot \mathbf{I}_i^T \quad (3)$$

Singular value decomposition (SVD) provides the best approximation of $\hat{\mathbf{R}}$ to $\mathbf{R}$ with respect to least-square, which is usually extended using regularization to prevent overfitting [9].

3. SI FOR RECOMMENDER SYSTEMS

In this section we present our collaborative ranking approach for learning to rank. The goal of learning to rank is to automatically learn a ranking model from training data, such that the model can sort objects (e.g., documents, songs, movies) according to their degrees of relevance, preference, or importance as defined in a specific application.

In the general learning to rank scenario, a retrieval function is learned from rankings of documents associated to queries. Each document is represented by a vector of predefined features that characterize it, for example, text documents are usually characterized by their term frequency, inverse document frequency, document length and other low-level content characteristics, as well as high-level content features as BM25 scores. In the case of web pages, hyperlink features are also used, such as PageRank or HITS.

In CF, on the other hand, item and user features can be learned based on user-item interactions (e.g, ratings or relevance scores)

Algorithm 1 : SwarmRankCF

Input: (i) Matrix $\mathbf{R}$ of user-item interactions and (ii) The number of factors k

Output: A ranking function $f(u,i) = \vec{p_g} \cdot \phi(u,i)$

1: Extract k collaborative features (i.e., latent factors) $\mathbf{I}$ by factorizing matrix $\mathbf{R}$, e.g, by applying SVD (Section 2.3).

2: Apply SwarmRank to learn a ranking function $f(u,i)$, by optimizing Mean Average Precision (MAP), using U as queries Q, items I as documents D, the item feature vectors $\mathbf{I}$ as $\phi(u,i)$, and $\mathbf{R}$ as the relevance scores Y (Section 2.1).

3: **return** $f(u,i)$

using a matrix factorization method, which learns for each item (columns of $\mathbf{R}$), a k-dimensional feature vector (rows of $\mathbf{I}$), where each row of $\mathbf{U}$ is a feature vector that captures the latent factors of the users, and can be considered as a linear predictor, predicting the entries in the corresponding row of $\mathbf{R}$, based on inner products in a k-dimensional space (Section 2.3).

We propose to learn a collaborative ranking model using SwarmRank. To this end, we cast the item recommendation task as a learning to rank problem, where users U can be considered as queries Q, items I as documents D, feature vectors $\phi(u,i)$ as $\mathbf{I}$ (i.e, the latent factors)[2]. Finally, the relevance scores Y are given by $\mathbf{R}$, that corresponds to ratings or implicit feedback information that measures user u's preferences for item i. The approach, named *SwarmRankCF* is summarized in Algorithm (1).

4. EXPERIMENTS AND EVALUATION

We evaluate the recommendation performance of our approach, SwarmRankCF, on a public dataset obtained from *Last.fm*, a major social media Internet radio station. In our evaluation, we empirically compared the recommendation quality of SwarmRankCF to: (i) *PureSVD*: a matrix factorization algorithm based on SVD and proposed for Top-N recommendation tasks [4], (ii) *Most Popular*: a non-personalized method that recommends the most popular artists to every user, and (iii) *Random*: a baseline method, that recommends random artists to users.

Dataset

We conducted experimentation, parametrization and algorithmic fine tuning on the "real-world" public dataset *Last.fm Dataset – 1K users* [3], which represents the whole listening habits, until May, 2009, for nearly 1,000 users.

Specifically, we used in our evaluation a 5-core subset of the dataset, i.e., every user listened to at least 5 songs and each song was listened by at least 5 users. The 5-core statistics are as follows: 242,103 user-item interactions (transactions), 844 unique users and 37,315 unique items (artists).

Evaluation Methodology

We performed a time sensitive split of the dataset S into two sets: a training set S_{train} and a testing set S_{test}. Consider we make the split at time t_{split}, then we put into S_{train} the individual training examples (i.e., user rankings) with timestamps less than t_{split}. Into S_{test}, we put the user rankings with timestamps greater than t_{split}. The recommenders are trained on S_{train} and then their performance is measured on S_{test}.

[2]Please note that since we are learning a linear ranking function, the model cannot make use of the user u features, because such features are the same for all items of user u within his rankings.

To evaluate the recommenders we used a variant of the *all-but-1* protocol [1], also known as the leave-one-out holdout method. In particular, we followed a similar schema as the one described in [4].

Our goal is to evaluate the system performance when it suggests Top-N items to a user. For example, recommending the user a few specific artists which are supposed to be the most attractive to him. That is, to find the relative position of these interesting items within the total order of items ranked for a specific user.

For each user u we aggregate his rankings in the test set S_{test}, by accumulating the item frequencies across those rankings in order to produce a single total ranking. The items are again sorted in descending order of their accumulated frequencies. We take one item at random from the the top-10 of the aggregated ranking and hide it. In total, we have $|U|$ hidden items.

Then, for each hidden item i, we randomly select 100 additional items from the test set S_{test}. Notice that most of those items selected are probably not interesting to user u.

We predict the scores for the hidden item i and for the additional 100 items, forming a ranking by ordering the 101 items according to their scores. The best expected result is that the interesting item i to user u will precede the rest 100 random items.

Finally, we generate a Top-N recommendation list by selecting the N items with the highest score. If the test item i is in the Top-N, then we have a *hit*, otherwise we have a *miss*. We measure the quality by looking at the *recall* metric.

More formally, in our recommender systems setting, recall is defined as:

$$\text{recall} := \frac{\sum_{u \in U} hit(u)}{|U|}\ , \tag{4}$$

where $hit(u)$ is a binary function that returns 1, if the hidden item i is present in u's Top-N list of recommendations, and 0 otherwise. A recall value of 1.0 indicates that the system was able to always recommend the hidden item, whereas a recall of 0.0 indicates that the system was not able to recommend any of the hidden items. Since the precision is forced by taking into account only a restricted number N of recommendations, there is no need to evaluate *precision* or *F1* measures, i.e., for this kind of scenario precision is just the same as recall up to a multiplicative constant.

Experimental Setting

For SwarmRankCF, we used SVD as factorization method to learn the collaborative feature vectors. Furthermore, we set $k = 46$ as the number of latent factors for SwarmRankCF and PureSVD. The rationale behind using the particular value of $k = 46$ is to have an experimental setting with a dimensionality similar to the one used in non-personalized learning to rank tasks, e.g., the number of features in the benchmark dataset LETOR [10]. SwarmRankCF has been implemented in the Java programming language using the optimization framework provided by GenOpt [12]. Specifically, in our experiments we used the PSO with Constriction Coefficient algorithm (PSOCC) with parameter values that are most commonly found in the literature (e.g., [11]). The parameter setting for SwarmRankCF is summarized in Table 1.

Results

Recall was evaluated for different recommendation list sizes: Top-N, where $N \in \{1, 5, 10, 20\}$. The results are presented in Figure 1. SwarmRankCF clearly delivers the best recommendations, achieving significantly better results than the other methods (two-sample t-test, $p < 0.025$). As expected, SwarmRankCF and PureSVD largely outperform the non-personalized baselines. The reader should note

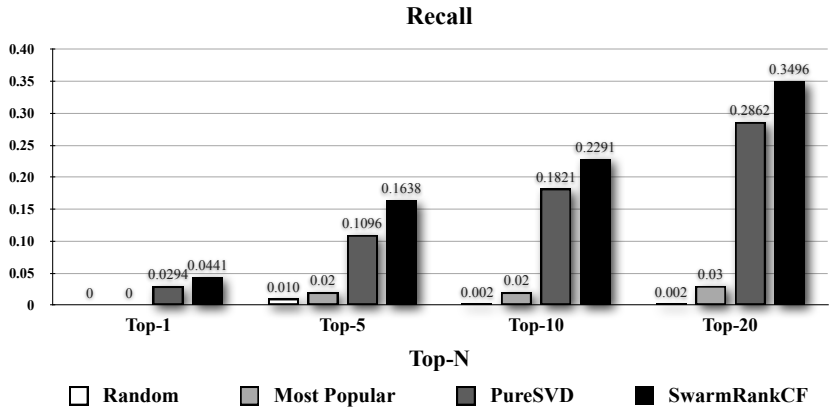

Figure 1: Recommendation performance in terms of recall measure.

Parameter	Value
SI Algorithm	PSO with Constriction Coefficient
Swarm size	50
Neighborhood Topology	Local Best ($lbest$)
Neighborhood Size	10
Cognitive Acceleration	$c_1 = 2.05$
Social Acceleration	$c_2 = 2.05$
Constriction Coefficient	$\chi = 0.72984$
# Dimensions	46 (one per latent feature)

Table 1: SwarmRankCF Parameter Settings.

that both SwarmRankCF and PureSVD use the latent factors learned by factorizing the user-item matrix using SVD, the strong performance of SwarmRankCF may be attributed to its better ability to model ranking semantics based on those features.

5. RELATED WORK

In recent years, the task of learning to rank has drawn a lot of interest in machine learning and several methods have been applied to learn ranking functions and promising results have been obtained. Typical methods include Ranking SVM [8, 7], RankBoost [6] and RankNet [2]. These three methods minimize loss functions that are loosely related to the evaluation measures such as MAP and NDCG. Given the nature of our approach, we are able to directly optimize such measures. All aforementioned algorithms, rely upon a manually selected set of features to build the ranking models. In contrast, our approach uses collaborative latent factors, learned directly from the user-item interactions, as feature vectors to characterize the user and items.

In the field of recommender systems, a lot of attention was directed to the rating prediction task, motivated by the Netflix challenge (e.g., [9]), standard sparse regression and classification methods proved to be successful tackling the challenge, but they cannot be directly applied to the item recommendation problem, as only positive observations are made. State-of-the-art methods for item recommendation are based on matrix factorization models that infer latent factors from the user-item interaction matrix, which we exploit to learn ranking functions for item prediction.

6. CONCLUSION AND FUTURE WORK

In this paper, we proposed SwarmRankCF to address the item recommendation task in the context of recommender systems. SwarmRankCF is a collaborative learning to rank algorithm based on swarm intelligence. While learning to rank algorithms use hand-picked features to represent items, we learn such features based on user-item interactions, and apply a PSO-based optimization algorithm that directly maximizes Mean Average Precision. Our experimental study demonstrates the recommendation performance gain of SwarmRankCF for the Top-N recommendation task.

For future work, we plan to extend SwarmRankCF to handle stream data. Most learning to rank algorithms work in batch mode, but social media streams, require ranking models to be updated online. Swarm intelligence can help to evolve models when new data enters the system. Another future research direction is to learn the latent factors using swarm intelligence as well, and at the same time optimize the ranking measure.

Acknowledgments This work was funded, in part, by the European Commission FP7 under grant agreements No.247829 and No.287704 for the M-Eco and CUBRIK projects, respectively, and by the NTH School (Niedersächsische Technische Hochschule) for IT Ecosystems.

7. REFERENCES

[1] J. S. Breese, D. Heckerman, and C. M. Kadie. Empirical analysis of predictive algorithms for collaborative filtering. In *Conference on Uncertainty in Artificial Intelligence*, 1998.

[2] C. Burges, T. Shaked, E. Renshaw, A. Lazier, M. Deeds, N. Hamilton, and G. Hullender. Learning to Rank Using Gradient Descent. In *Proceedings of the ICML*, 2005.

[3] O. Celma. *Music Recommendation and Discovery in the Long Tail*. Springer, 2010.

[4] P. Cremonesi, Y. Koren, and R. Turrin. Performance of recommender algorithms on top-n recommendation tasks. In *Proceedings of the ACM RecSys conference*, 2010.

[5] E. Diaz-Aviles, W. Nejdl, and L. Schmidt-Thieme. Swarming to rank for information retrieval. In *Proceedings of the ACM GECCO conference*, 2009.

[6] Y. Freund, R. Iyer, R. E. Schapire, and Y. Singer. An efficient boosting algorithm for combining preferences. *J. Mach. Learn. Res.*, 4:933–969, 2003.

[7] R. Herbrich, T. Graepel, and K. Obermayer. Large Margin Rank Boundaries for Ordinal Regression. In *Advances in Large Margin Classifiers*, 2000.

[8] T. Joachims. Optimizing search engines using clickthrough data. In *Proceedings of the ACM KDD conference*, 2002.

[9] Y. Koren, R. Bell, and C. Volinsky. Matrix factorization techniques for recommender systems. *Computer*, 42:30–37, August 2009.

[10] T.-Y. Liu. *Learning to Rank for Information Retrieval*. Springer, 2011.

[11] R. Poli, J. Kennedy, and T. Blackwell. Particle swarm optimization. *Swarm Intelligence*, 1(1):33–57, 2007.

[12] M. Wetter. Generic Optimization Program – GenOpt. *User Manual,User Manual Version 3.1.0. Lawrence Berkeley National Laboratory.*, 2011.