Logistic Matrix Factorization for Implicit Feedback Data

Christopher C. Johnson
Spotify
New York, NY 10011
cjohnson@spotify.com

Abstract

Collaborative filtering with implicit feedback data involves recommender system techniques for analyzing relationships betweens users and items using implicit signals such as click through data or music streaming play counts to provide users with personalized recommendations. This is in contrast to collaborative filtering with explicit feedback data which aims to model these relationships using explicit signals such as user-item ratings. Since most data on the web comes in the form of implicit feedback data there is an increasing demand for collaborative filtering methods that are designed for the implicit case. In this paper we present Logistic Matrix Factorization (Logistic MF), a new probabilistic model for matrix factorization with implicit feedback. The model is simple to implement, highly parallelizable, and has the added benefit that it can model the probability that a user will prefer a specific item. Additionally, we show it to experimentally outperform the widely adopted Implicit Matrix Factorization method using a dataset composed of music listening behavior from streaming music company Spotify.

1 Introduction

Users of modern e-commerce services are exposed to a myriad of diverse product choices. Helping users sort through these choices and find what they are looking for can mean a huge increase in user satisfaction and company profit. For this reason, most major web businesses are investing substantial resources into building features to simplify and personalize the process of sorting through their product options. The goal of recommender systems is to analyze data associated with users and products in order to provide users with personalized recommendations. Recommender systems generally fall into two categories: content based and collaborative filtering. Content based strategies involve analyzing features directly associated with the users and products such as a user's age, sex, and demographic, or the release year and genre of a specific music album. Collaborative filtering based recommender systems examine users' past behavior in order to predict how users will act in the future. Perhaps the most famous example of this comes from the Netflix Prize [2] in which the movie streaming service Netflix[1] released a dataset of explicit movie ratings from 500,000 anonymized users and offered a $1 million prize to the first team of researchers able to improve the company's RMSE performance by 10%. The contest, which was eventually won by the BellKor team, stirred up a great deal of research in analyzing methods for collaborative filtering with explicit feedback data such as Netflix movie ratings. However, implicit feedback data such as clicks, page views, purchases, or media streams can be collected at a much larger and faster scale and without needing the user to provide any explicit sentiment.

[1] https://www.netflix.com

For this reason, there has been a recent surge of work focusing on the task of collaborative filtering with implicit as opposed to explicit feedback data [17].

In this paper we explore a new method for collaborative filtering with implicit feedback data that's based in part on the traditional matrix factorization strategy. We first describe the theoretical foundation of the model and then describe how it can be trained using an alternating gradient descent approach. Next, we show how the model training can be parallelized and scaled up to Millions of users and items using modern high performance computing frameworks such as Hadoop and Spark. Finally, we provide an experimental evaluation of our model using a dataset of music streaming counts from online music streaming service Spotify[2]. In this setting we compare our model to the frequently used Implicit Matrix Factorization (IMF) model [12] as well as a baseline popularity based strategy.

2 Problem Setup and Notation

The goal of collaborative filtering with implicit feedback data is to analyze a user's past behavior in order to predict how they will act in the future. Implicit feedback can come in many forms such as clicks, page views, or media streaming counts, but in all cases we will assume that we have a set of non-negative feedback values associated with each pair of users and items in our domain. More formally:

- $U = (u_1, \ldots, u_n)$: a group of n users
- $I = (i_1, \ldots, i_m)$: a group of m items
- $R = (r_{ui})_{n \times m}$: a user-item observation matrix where $r_{ui} \in \mathbb{R}_{\geq 0}$ represents the number of times that user u interacted with item i

Notice that we don't require the r_{ui} values to be integers but instead allow them to be any non-negative reals. This is to allow contextual or temporal weighting of observations. For example, in the case of a music streaming service such as Spotify we may choose to weight explicit clicks by a user higher than streams without clicks. Additionally, we may choose to weight more recent user streams higher than older streams as a user's taste may change slightly over time. Furthermore, it's important to note that in most cases R is a very sparse matrix as most users only interact with a small number of items in I. For any entries r_{ui} where user u does not interact with item i we place $0's$. It's important to note that that a value of 0 does not necessarily imply that the user does not prefer the item, but could simply imply that the user does not know about the item. Then, our goal is to find the top recommended items for each user for each item that they have not yet interacted with (where $r_{ui} = 0$).

3 Related Work

Following the Netflix prize [2], collaborative filtering has become a well studied topic in the domain of recommender systems. In most cases this has involved models that take as input a set of explicit user-item ratings and attempt to recommend users items that they haven't yet rated. Neighborhood based methods were some of the first collaborative filtering models for explicit feedback data and can be either user-oriented or item-oriented. User-oriented neighborhood methods search the domain of U to find users that have rated or interacted with similar items and then recommend items that similar users prefer [10]. Item-oriented neighborhood methods instead search the domain of I to find items similar to the items that a user has either rated highly or interacted with. Matrix Factorization models seek to find a low rank approximation to a sparse ratings matrix by minimizing a loss function on the known ratings [6]. These methods are often a popular choice for industry recommender systems due to their simplicity and superior performance to neighborhood based methods. Additionally, MF models have been shown to be efficiently parallelizable and highly scalable to large web scale datasets [7, 19, 23, 24]. Matrix Factorization models are a subset of a larger family of models known as latent factor models. Latent factor models attempt to

[2]https://www.spotify.com

uncover latent or unobserved factors to encode the users and items in U and I. Examples of other latent factor models include Probabilistic Latent Semantic Indexing (PLSA) [11] , Latent Dirchlet Allocation (LDA) [3], and Restricted Boltzmann Machines (RBM) [21], all of which have been applied to the domain of collaborative filtering.

In many real world scenarios, explicit feedback such as item ratings can be difficult to obtain and so there has recently been considerable attention spent on methods that can learn from implicit feedback datasets such as clicks, views, or purchases [17]. Hu et al. [12] propose treating implicit feedback data as binary preferences with a higher confidence placed on observations that include a greater number of interactions. They then formulate a Matrix Factorization problem that minimizes a weighted root mean squared error over the training data. Others have formulated the problem as One-Class Collaborative Filtering in which data is considered either positive or unlabeled. Pan et al. [18] employ a weighted Matrix Factorization approximation to One-Class Collaborative Filtering using negative sampling.

Many hybrid approaches have also been developed that utilize side information such as context, temporal information, or item content within traditional CF models. Context-aware collaborative filtering attempts to take into account the context with which users provided implicit feedback, such as the time of day, day of the week, or location. Karatzoglou et al. [13] propose a tensor factorization model that generalizes the traditional MF approach to incorporate contextual side information. Koren [14] proposes a CF model that takes into account how a user's taste drifts over time by modeling the temporal dynamics of the data. Another popular approach is to utilize item content such as metadata, text, or audio content to improve collaborative filtering models. Gunawardana et al. [9] propose Unified Botzmann Machines which are probabilistic models that encode content information into the collaborative filtering framework.

Probabilistic approaches to collaborative filtering attempt to encode the probability of a user choosing to interact with an item. Salakhutdinov et al. [20] model the conditional distribution of an explicit ratings matrix given user and item latent factor vectors as being distributed according to a Gaussian and solve for optimal vectors by maximizing the log-posterior. Mnih et al. [16] provide a probabilistic framework for the implicit case where they model the probability of a user choosing an item as being distributed according to a normalized exponential function. They avoid linear time computation and approximate the normalization to the distribution by traversing a tree structure. More recently, Goplan et al. [8] introduced a factorization model that factorizes users and items by Poisson distributions. In contrast to their work we propose a new probabilistic framework for the implicit case in which we model the probability of a user choosing an item by a logistic function.

4 Logistic MF

Our model takes a similar approach to [12] by factorizing the observation matrix R by 2 lower dimensional matrices $X_{n\times f}$ and $Y_{m\times f}$ where f is the number of latent factors. The rows of X are latent factor vectors that represent a user's taste while the columns of Y^T are latent factor vectors that represent an item's style, genre, or other implicit characteristics. However, while [12] minimizes the weighted RMSE between a binary based preference matrix and the product of U and V, we take a probabilistic approach.

Let $l_{u,i}$ denote the event that user u has chosen to interact with item i (user u prefers item i). Then, we can let the probability of this event occurring be distributed according to a logistic function parameterized by the sum of the inner product of user and item latent factor vectors and user and item biases.

$$p(l_{ui} \mid x_u, y_i, \beta_i, \beta_j) = \frac{\exp(x_i y_i^T + \beta_u + \beta_i)}{1 + \exp(x_u y_i^T + \beta_u + \beta_i)} \tag{1}$$

Here, the β_i and β_j terms represent user and item biases which are meant to account for variation in behavior across both users and items. Some users will have a tendency to interact with a diverse assortment of items in I while others will only interact with a small

subset. Similarly, some items will be very popular and so will have a high expectation of being interacted with across a broad audience while other items will be less popular and only apply to a niche group. The bias terms are latent factors associated with each user $u \in U$ and item $i \in I$ that are meant to offset these behavior and popularity biases.

Given this formulation, let the non-zero entries of our observation matrix $r_{ui} \neq 0$ represent positive observations and the zero entries $r_{ui} = 0$ represent negative observations. Additionally, similar to [12], we define our "confidence" in the entries of R as $c = \alpha r_{ui}$ where α is a tuning parameter. Then, let each nonzero element $r_{ui} \neq 0$ serve as c positive observations and each zero element $r_{ui} = 0$ serve as a single negative observation. Increasing α places more weight on the non-zero entries while decreasing α places more weight on the zero entries. We've found that choosing α to balance the positive and negative observations generally yields the best results. Additionally, it should be noted that other confidence functions can replace c. For example, to remove the power user bias that comes from a dataset where a small minority of users contribute the majority of the weight it can help to use a log scaling function such as:

$$c = 1 + \alpha \log(1 + r_{ui}/\epsilon)$$

By making the assumption that all entries of R are independent we derive the likelihood of our observations R given the parameters X, Y, and β as:

$$\mathcal{L}(R \mid X, Y, \beta) = \prod_{u,i} p(l_{ui} \mid x_u, y_i, \beta_u, \beta_i)^{\alpha r_{ui}} (1 - p(l_{ui} \mid x_u, y_i, \beta_u, \beta_i)) \tag{2}$$

Additionally, we place zero-mean spherical Gaussian priors on user and item latent factor vectors to help regularize the model and avoid over fitting to the training data.

$$p(X \mid \sigma^2) = \prod_u \mathcal{N}(x_u \mid 0, \sigma_u^2 I), \quad p(Y \mid \sigma^2) = \prod_i \mathcal{N}(y_i \mid 0, \sigma_i^2 I)$$

Taking the log of the posterior and replacing constant terms with a scaling parameter λ we arrive at the following.

$$\log p(X, Y, \beta \mid R) = \sum_{u,i} \alpha r_{ui}(x_u y_i^T + \beta_u + \beta_i) - (1 + \alpha r_{ui}) \log(1 + \exp(x_u y_i^T + \beta_u + \beta_i)) - \frac{\lambda}{2}\|x_u\|^2 - \frac{\lambda}{2}\|y_i\|^2 \tag{3}$$

Notice that placing zero mean Gaussian priors on the latent factor vectors simply amounts to ℓ_2-regularization of the user and item vectors. Then, our goal is to learn X, Y, and β that maximize the log posterior (3).

$$\arg\max X, Y, \beta \log p(X, Y, \beta \mid R) \tag{4}$$

A local maximum of the objective defined in (4) can be found by performing an alternating gradient ascent procedure. In each iteration we first fix the user vectors X and biases β and take a step towards the gradient of the item vectors Y and biases β. Next, we fix the item vectors Y and biases β and take a step towards the gradient of the user vectors X and biases β. The partial derivatives for the user vectors and biases are given by:

$$\frac{\partial}{\partial x_u} = \sum_i \alpha r_{ui} y_i - \frac{y_i (1 + \alpha r_{ui}) \exp(x_u y_i^T + \beta_u + \beta_i)}{1 + \exp(x_u y_i^T + \beta_u + \beta_i)} - \lambda x_u \tag{5}$$

$$\frac{\partial}{\partial \beta_u} = \sum_i \alpha r_{ui} - \frac{(1 + \alpha r_{ui}) \exp(x_u y_i^T + \beta_u + \beta_i)}{1 + \exp(x_u y_i^T + \beta_u + \beta_i)} \tag{6}$$

Each iteration is linear in the number of users $|U|$ and items $|I|$. Though linear time complexity might not seem prohibitive, for larger domains this can become a limitation. In situations where linear computation is not possible we sample fewer negative samples ($r_{ui} = 0$) and decrease α in response. This gives enough of an approximation to the full loss that near optimal vectors can be solved with much less computation time. Additionally, we've found that the number of iterations required for convergence could be reduced dramatically by choosing the gradient step sizes adaptively via AdaGrad [5]. Let x_u^t denote the value of x_u at iteration t and $g_{x_u}^t$ denote the gradient of (3) with respect to x_u at iteration t. At iteration t we perform the following AdaGrad update to x_u.

$$x_u^t = x_u^{t-1} + \frac{\gamma g_u^{t-1}}{\sqrt{\sum_{t'=1}^{t-1} {g_u^{t'}}^2}} \tag{7}$$

5 Scaling Up

Each iteration of the alternating gradient descent procedure involves computing the gradient for all latent factor vectors and then taking a step towards the positive direction of the gradient. Each of these gradients (5, 6) involves a sum of a set of functions that each depend on a single user and item. These summations can be performed in parallel and fit the MapReduce programming paradigm.

To scale up computation we employ a sharding technique similar to those used by other latent factor models [4, 7]. We first partition R into $K \times L$ blocks of K rows and L columns where $K \ll n$ and $L \ll m$ are parallelization factors. Additionally, we partition X into K blocks and Y into L blocks. Figure 1 gives a pictorial representation of the sharding of R. Each block depends on at most n/K users and m/L items so even if the full observation matrix R and set of user and item matrices X and Y cannot fit in memory, we can choose parallelization factors K and L such that each block can fit in memory. In the map phase, we partition all observations r_{ui}, user vectors x_u, and item vectors y_i from the same block to the same mapper. For each pair of users and items u and i we compute the following in parallel.

$$v_{ui} = \alpha r_{ui} y_i - \frac{y_i(1 + \alpha r_{ui}) \exp(x_u y_i^T + \beta_u + \beta_i)}{1 + \exp(x_u y_i^T + \beta_u + \beta_i)} \tag{8}$$

$$b_{ui} = \alpha r_{ui} - \frac{(1 + \alpha r_{ui}) \exp(x_u y_i^T + \beta_u + \beta_i)}{1 + \exp(x_u y_i^T + \beta_u + \beta_i)} \tag{9}$$

In the reduce phase we key off u (or i if performing an item iteration) such that each v_{ui} and b_{ui} that map to the same user u (or item i if performing an item iteration) are sent to the same reducer. It follows that $\frac{\partial}{\partial x_u} = \sum_i v_{ui}$ and $\frac{\partial}{\partial \beta_u} = \sum_i b_{ui}$ and so these summations can be efficiently aggregated in parallel in the reduce phase. Finally, once the partial derivates $\frac{\partial}{\partial x_u}$ and $\frac{\partial}{\partial \beta_u}$ have been computed we then update x_u and β_u according to (7).

6 Experimental Study

6.1 Dataset

We evaluated our model using a dataset consisting of user listening behavior from music streaming service Spotify. Our dataset is composed of the listening behavior for $|X| = 50,000$ uniformly random active users, limiting ourselves to the top $|I| = 10,000$ most popular artists on Spotify. For each user we tracked the number of times they listened to each artist in I over a period of 2 years from May, 2011 to June 2013. We define a "listen" to be any continuous stream of a song for more than 30 seconds. This is to remove a lot of the bias that comes from users sampling artists that they end up not liking. For each user-item pair in $U \times I$ we aggregated a tuple of the form `(<user>, <artist>, <listens>)` where

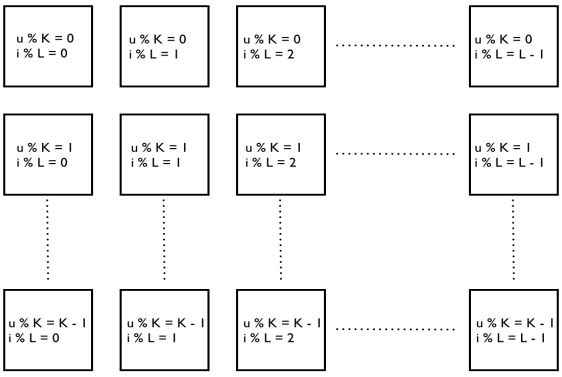

Figure 1: Sharding of users and items into $K \times L$ blocks for MapReduce implementation of model training.

<user> $\in U$, <artist> $\in I$, and $r_{ui} =$ <listens>. In total, R is composed of 86 million non-zero entries and 414 million zero entries. We chose to keep the dataset simple and not add additional contextual or temporal weighting.

6.2 Evaluation Metric

One of the difficulties with evaluating collaborative filtering models with implicit feedback datasets is that the data only encodes positive feedback. If a user doesn't listen to a particular artist it doesn't necessarily mean that they dislike the artist but could instead mean that they've never heard of the artist. In fact, if a user has only listened to a particular artist once or twice this often indicates that the user tried out the artist but ultimately decided that they didn't like them. On the other hand, if a user listens to an artist a large number of times then this is a strong positive signal that the user likes the artist. Due to the lack of negative feedback, we choose a recall based evaluation metric known as Mean Percentage Ranking (MPR) [12] that evaluates a user's satisfaction with an ordered list of recommended items.

We removed a uniform random subset of 10% of the entries in R as a test set (R_{test}) and trained on the remaining 90%. We distinguish between entries in R_{test} as opposed to R using a superscript r^t_{ui} We chose to remove random entries in R as opposed to random listens so as avoid the obvious bias that users will tend to revisit the same artists. In other words, we wanted to avoid a scenario that favored a simple baseline model that always predicted the items the user already listened to. By removing full entries, the test set consists entirely of user-item pairs for which the training set has no listening data. The goal of collaborative filtering is to predict the top new items for a user and not the top items the user has already interacted with. So, by limiting the test set to new user-item pairs we are able to define an evaluation metric more inline with the problem we are solving.

For each user $u \in U$ we generated a ranked list of the items in I sorted by preference. Let $rank_{ui}$ denote the percentile ranking of item i for user u. $rank_{ui} = 0\%$ signifies that i is predicted as the highest recommended item for u. Similarly, $rank_{ui} = 100\%$ signifies that i is predicted as the lowest recommended item for u. The percentile ranking is then evenly distributed among the remaining items in the list by steps of $\frac{100\%}{|I|}$. Our basic measure of quality is the expected percentile ranking of a user listening to the artists in the test set, which can be defined as:

$$MPR = \frac{\sum_{ui} r^t_{ui} rank_{ui}}{\sum_{ui} r^t_{ui}} \tag{10}$$

Lower values of MPR are more desirable as they indicate that the user listened to artists higher in their predicted lists. Conversely, higher values of MPR indicate that users listened

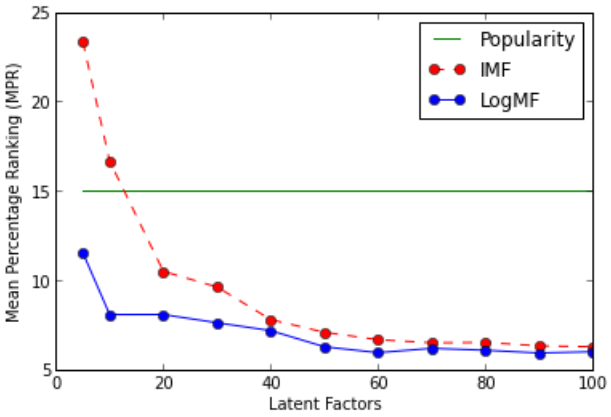

Figure 2: MPR for popularity baseline, IMF, and Logistic MF using streaming count data for 50k users and the top 10k artists on Spotify

to artists lower in their predicted lists. It should also be noted that randomly produced lists would have an expected MPR of 50%.

6.3 Results

We evaluated our model using the MPR evaluation metric for a differing number of latent factors ranging from 5 to 100. We found that increasing the number of latent factors beyond 100 did not improve performance on our dataset. Additionally, we also implemented the Implicit Matrix Factorization (IMF) model defined in [12] for the same set of varying latent factors. For both models, we tuned α and λ by cross validation, though in both cases choosing an α that roughly balanced the positive and negative observations yielded the best performance. We ran both models until the change in MPR was minimal, which ultimately amounted to around 10 to 20 iterations for IMF and 30 to 40 iterations for LogisticMF (depending on the number of latent factors). In addition to these 2 models we also evaluated against a popularity baseline in which all ranked lists were based on the globally most popular artists on Spotify over the training period. Since the majority of users tend to generally listen to the same set of artists, a popularity baseline already performs quite well on the test set. That said, we feel that music tastes are in general more personal and niche than movie or television viewing and that a popularity based model is less applicable in the domain of music than in movies. The most popular movies and shows tend to have a much wider audience reach than the most popular artists and so we believe that personalized recommendations are even more important in our domain than other frequently evaluated domains such as Netflix.

Figure 2 depicts the MPR evaluation for LogisticMF and IMF for a differing number of latent factors, as well as the popularity baseline. The popularity baseline achieved an MPR of 14.9% which is already a huge improvement over a purely random model which has an expected MPR of 50%. Both LogisticMF and IMF eventually converged to a similar MPR at around 100 latent factors (6.27 for IMF and 5.99 for LogisticMF), both of which were significant improvements over the popularity baseline. The most noticeable improvement of LogisticMF to IMF comes from its ability to outperform IMF under a fewer number of latent factors. For example, in using just 10 latent factors IMF achieved an MPR of 16.9% which is just above the popularity baseline. On the other hand, LogisticMF achieved an MPR of 8.065%. The ability to perform well using less latent factors is a desirable characteristic for several reasons.

1. The curse of dimensionality tells us that as we increase the number of latent factors we will in turn need more training data to effectively learn near optimal vectors. In situations where we do not have access to a large amount of training data, increasing the number of latent factors may not yield performance improvements.

Daft Punk		Bob Dylan		Mumford & Sons	
LogisticMF	**IMF**	**LogisticMF**	**IMF**	**LogisticMF**	**IMF**
Justice	Muse	Neil Young	Simon & Garfunkel	Jerry Douglas	Florence and The Machine
The Chemical Brothers	Kings Of Leon	Buffalo Springfield	The Rolling Stones	Ben Howard	MGMT
Fatboy Slim	Gorillaz	Lou Reed	The Beach Boys	The Lumineers	Lana Del Rey
Gorillaz	The Killers	The Animals	David Bowie	Noah And The Whale	The Killers
The Prodigy	Jay-Z	Simon & Garfunkel	Neil Young	Radical Face	Kings Of Leon
Deadmau5	Foo Fighters	Johnny Cash	The Cure	Of Monsters And Men	Band of Horses
Basement Jaxx	MGMT	Paul Simon	R.E.M.	The Tallest Man On Earth	Gotye
Röyksopp	Kanye West	The Kinks	Bruce Springsteen	Ray LaMontagne	Snow Patrol
Ratatat	Florence and The Machine	The Velvet Underground	Elvis Presley	Edward Sharpe & The Magnetic Zeros	Daft Punk
Moby	Coldplay	Van Morrison	Dire Straits	The Shins	Foo Fighters

Table 1: Top Related Artists lists sorted by cosine similarity (11) between item vectors.

2. Recommendation features, such as nearest neighbor searches, can be performed faster and using less memory if using less latent factors. A common practice for collaborative filtering based recommender systems is to first learn a set of item vectors and then update user vectors more frequently in an online fashion while keeping the item vectors static. Recommendations can then be computed in real time by performing an approximate nearest neighbor (ANN) search over the space of item vectors in I. ANN search requires keeping a data structure such as a k-d tree or locality sensitive hashing (LSH) based trees in RAM. The size of these data structures is proportional to both the number of items as well as the dimensionality of the latent factor vectors and so reducing the number of latent factors can greatly reduce the size of these objects.

6.4 Qualitative Analysis

In addition to our quantitative evaluation we also explored a qualitative analysis. One of the benefits to using latent factor models is that it places all items in I into a lower dimensional space where we can compute similarities between items. We used cosine similarity to measure the similarity between item vectors in a latent factor space. Given two vectors a and b the cosine similarity between the vectors is defined as:

$$sim_{ab} = \frac{\sum_i a_i b_i}{\sqrt{\sum_i a_i^2}\sqrt{\sum_i b_i^2}} \tag{11}$$

The top related items for a given item a can be defined as the top items $b \in I$ ranked by sim_{ab}. We trained both LogisticMF and IMF using the same Spotify listening history dataset from our quantitative analysis using 60 latent factors. Next, we computed the top related items for several artists in I using the learned item vectors. Table 1 lists the top related items for 3 artists for which we feel that LogisticMF qualitatively outperformed IMF, though we leave it to the reader to have their own opinion.

7 Summary and Discussion

In this paper, we presented a new probabilistic latent factor model for collaborative filtering with implicit feedback data. Our model is simple to implement, highly scalable, and has been shown to outperform the widely used IMF model [12] using a dataset composed of user listening behavior from Spotify. Additionally, our model has the added benefit that it encodes the probability that a user will prefer an item in the domain of I. At Spotify, we've scaled this model up to 40 million users and 20 million songs and use it to power various music recommendation features including Discover, Radio, and Related Artists.

References

[1] Bell R & Koren Y. & Volinsky C. (2007) Modeling Relationships at Multiple Scales to Improve Accuracy of Large Recommender Systems *Proc. of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining.*

[2] Bennet J. & Lanning S. (2007) The Netflix Prize *KDD Cup and Workshop*.

[3] Blei D. & Ng A. & Jordan M. (2003) Latent Dirchlet Allocation *Journal of Machine Learning Research*.

[4] Das A. & Mayur D. & Garg A. & Rajaram S. (2007) Google News Personalization: Scalable Online Collaborative Filtering *Proc. of the 16th International Conference on the World Wide Web*, pp. 271-280.

[5] Duchi J. & Hazan E. & Singer Y. (2011) Adaptive Subgradient Methods for Online Learning and Stochastic Optimization *Journal of Machine Learning Research*.

[6] Funk S. (2006) Netflix Update: Try this at Home *http://sifter.org/ simon/journal/20061211.html*.

[7] Gemulla R. & Nijkamp E. & Haas P. & Sismanis Y. (2011) Large Scale Matrix Factorization with Distributed Stochastic Gradient Descent *Proc. of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*.

[8] Goplan P. & Hofman J. & Blei D. (2013) Scalable Recommendation with Poisson Factorization *arXiv preprint arXiv:1311.1704*.

[9] Gunawardana A. & Meek C. (2009) A Unified Approach to Building Hybrid Recommender Systems *Proc. of the 3rd ACM Conference on Recommender Systems (RecSys)*, pp. 117-124.

[10] Herlocker J. & Konstan J. & Borchers A. & Riedl J. (1999) An Algorithmic Framework for Performing Collaborative Filtering *Proc. of the 22nd ACM SIGIR Conference on Information Retrieval*, pp. 230-237.

[11] Hofmann T. (2007) Latent Semantic Models for Collaborative Filtering *ACM Transactions on Information Systems*, pp. 89-115.

[12] Hu Y. & Koren Y. & Volinsky C. (2008) Collaborative Filtering for Implicit Feedback Datasets *8th IEEE International Conference on Data Mining*, pp. 263-272.

[13] Karatzoglou A. & Amatriain X. & Baltrunas L. & Oliver N. (2010) Multiverse Recommendation: N-dimensional Tensor Factorization for Context Aware Collaborative Filtering Proc. of the 4th ACM Conference on Recommender Systems (RecSys), pp. 79-86.

[14] Koren Y. (2009) Collaborative Filtering with Temporal Dynamics *Proc. of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*.

[15] Linden G. & Smith B. & York J. (2003) Amazon.com Recommendations: Item to Item Collaborative Filtering *IEEE Internet Computing*, pp. 76-80.

[16] Mnih A. & Yee W. (2011) Learning Item Trees for Probabilistic Modeling of Implicit Feedback *CoRR*.

[17] Oard D. & Kim J. (1998) Implicit Feedback for Recommender Systems *Proc. of the 5th DELOS Workshop on Filtering and Collaborative Filtering*, pp. 31-36.

[18] Pan R. & Zhou Y. & Cao B. & Liu N. & Lukose R. & Scholz M. & Yang Z. (2008) One Class Collaborative Filtering *IEEE International Conference on Machine Learning*, pp. 502-511.

[19] Recht B. & Re C. (2013) Parallel Stochastic Gradient Algorithms for Large Scale Matrix Completion Mathematical Programming Computation, pp. 201-226.

[20] Salakhutdinov R. & Mnih A. (2007) Probabilistic Matrix Factorization iÂdvances in Neural Information Processing Systems (NIPS).

[21] Salakhutdinov R. & Mnih A. & Hinton G. (2007) Restricted Boltzmann Machines for Collaborative Filtering *International Conference on Machine Learning (ICML)*.

[22] Sarwar B. & Karypis G. & Konstan J. & Ridel J. (2001) Item Based Collaborative Filtering Recommendation Algorithms *Proc. of the 10th International Conference on the World Wide Web*, pp. 285-295.

[23] Yu H. & Hsieh C. & Si S. & Dhillon I. (2012) Scalable Coordinate Descent Approaches to Parallel Matrix Factorization for Recommender Systems IEEE International Conference on Data Mining.

[24] Zhou Y. & Wilkinson D. & Schreiber R. & Pan R. (2008) Large Scale Parallel Collaborative Filtering for the Netflix Prize *AAIM*, pp. 337-348.