A Matrix Factorization Algorithm for Music Recommendation using Implicit User Feedback

Maciej Pacula
MIT CSAIL
Cambridge, MA 02139, USA
mpacula@csail.mit.edu

Abstract

The goal of recommender systems is to make personalized product recommendations based on users' taste. As the Netflix challenge demonstrated, one of the the most effective way to build such systems is through matrix factorization. Matrix factorization algorithms utilize prior product feedback given by users to automatically build user and product profiles. A product can then be recommended to a user if the user's profile closely matches that of the product.

Unfortunately, most of the research in matrix factorization focuses on explicit feedback datasets, where users make their preferences known by directly rating subsets of available products on a fixed scale. In many real-worlds applications, however, such direct ratings are unavailable. Instead, implicit feedback must be used, such as browsing history and view counts. In this work we present a recommendation algorithm which uses implicit play frequency information to recommend artists to the users of the Last.fm[1] music website. We demonstrate how an existing algorithm for explicit feedback datasets can be adapted to work with implicit data. We further evaluate the performance of our recommendation algorithm on a Last.fm dataset, investigating the effects of regularization, normalization and the number of features on the quality of recommendations. We also discuss the viability of our approach in a real-world setting.

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

1 Introduction

Being able to make personalized recommendations is beneficial both for e-commerce sites and their users. Accurate recommendations improve customer experience by providing users with content they are likely to enjoy, and provide a potential for new purchases.

Recent years have seen a rise of interest in recommender algorithms, which can be broadly divided into two categories. *Content-based* approaches work by directly building profiles for users and products, often through explicit questionnaires and expert evaluation (Koren et al., 2009; Hu et al., 2008). An example of such a system is the internet radio Pandora, based on the Music Genome Project, where each song is characterized by an extensive feature set created by a music expert. The features, such as a song's genre, can then be matched with users' preferences in order to make recommendations. While sometimes successful, content-based approaches require access to a substantial amount of information that might be either unavailable or infeasible to obtain (Koren et al., 2009; Hu et al., 2008).

In constrast, *collaborative filtering* relies only on past user behavior to make recommendations, without the need to explicitly create profiles. Such past behavior can include product ratings, view counts and purchase history. One of the main appeals of collaborative filtering is that it is domain free, yet can address data aspects which are often elusive and difficult to profile with content-based systems (Koren et al., 2009).

Within the collaborative filtering domain, two approaches are prevalent: *neighborhood methods* and *latent factor models*. Neighborhood methods focus on inferring relationships between products and/or users, and using those relationships to make recommendations (Koren et al., 2009). For example, if a user gives a high rating to the movie *Star Wars*, it is reasonable to assume that they would also enjoy *Star Trek*, another sci-fi movie.

Latent factor models, on the other hand, attempt to characterize users and products by feature vectors automatically inferred from data (Koren et al., 2009; Hu et al., 2008). Such feature vectors describe users and products along multiple dimensions, somewhat similarly to explicit profiling in content-based systems, although the actual interpretation of various features is generally irrelevant and often impossible (Koren et al., 2009). As the Netflix challenge demonstrated, one of the the most effective way to build such systems is through *matrix factorization*, the focus of this work. Matrix factorization algorithms utilize prior item feedback given by users to automatically build user and product profiles. A product can then be recommended to a user if the user's profile closely matches that of the product.

Unfortunately, most of the research in matrix factorization focuses on high-quality explicit feedback datasets, where users make their preferences known by directly rating subsets of available items on a fixed scale (Hu et al., 2008). In many real-worlds applications, however, such direct ratings are unavailable. Instead, implicit feedback must be used, such as browsing history and view counts. In this work we present a recommendation algorithm which uses implicit play frequency information to recommend artists to the users of the Last.fm music website. We make the following contributions:

1. We demonstrate the inadequacy of an explicit feedback algorithm on an implicit feedback dataset

2. We propose a straightforward modification of an explicit feedback algorithm which significantly improves its performance on implicit data

3. We evaluate the modified algorithm with respect to regularization, number of features and other dimensions of interest

In addition, we briefly discuss the viability of our approach in a real-world setting, where the standard error measures such as the root mean square error (RMSE) are sometimes inadaquate.

2 Related Work

Many successful matrix factorization algorithms are inspired by Sigular Value Decomposition of the user-product rating matrix $M_{i,j}$ defined as:

$$M_{i_j} = r_{i,j}$$

where $r_{i,j}$ is the observed rating of product j by user i (Koren et al., 2009). The matrix M is usually sparse, with the Netflix Challenge matrix having less than 2% of all possible ratings available (Funk, 2006). For m users and n products, a typical model decomposes M into two matrices U and P of dimensions $f \times m$ and $f \times n$, respectively, such that M is approximated by the product $U^T P$. In other words, a rating $r_{i,j}$ is approximated by a dot product of u_i and p_j in a joint latent factor space of dimensionality f (Koren et al., 2009):

$$\hat{r_{i,j}} = u_i^T p_j$$

Parameter estimation is accomplished by minimizing the regularized square error between observed and estimated ratings (Koren et al., 2009; Hu et al., 2008):

$$\min_{U,P} \sum_{\substack{r_{i,j} \\ \text{known}}} \left(r_{i,j} - u_i^T p_j\right)^2 + \lambda \left(\|u_i\|^2 + \|p_j\|^2\right) \tag{1}$$

A common parameter estimation algorithm, popularized by Simon Funk, is *stochastic gradient descent* (SGD). For each training sample, the algorithm predicts the rating $r_{i,j}$ and computes the associated error:

$$e_{i,j} = r_{i,j} - u_i^T p_j \tag{2}$$

The algorithm then uses this error to modify its parameters by a magnitude proportional to γ in the direction opposite to the gradient, yielding (Koren et al., 2009):

$$u_i \leftarrow u_i + \gamma \cdot (e_{i,j} \cdot p_j - \lambda \cdot u_i) \tag{3}$$

$$p_j \leftarrow p_j + \gamma \cdot (e_{i,j} \cdot u_i - \lambda \cdot p_j) \tag{4}$$

While this approach works well for explicit feedback datasets, it requires modification for implicit feedback data where ratings can be uncertain. To model uncertainty, confidence coefficients $c_{i,j}$ are introduced, transforming the minimization problem in Equation 1 into (Hu et al., 2008):

$$\min_{U,P} \sum_{\substack{r_{i,j} \\ \text{known}}} c_{i,j} \left(r_{i,j} - u_i^T p_j\right)^2 + \lambda \left(\|u_i\|^2 + \|p_j\|^2\right) \tag{5}$$

Hu et al. propose a confidence measure which is a linear function of the rating:

$$c_{i,j} = 1 + \alpha \cdot r_{i,j}$$

where α is a constant determined by cross-validation. While in principle this new formulation can be solved using Stochastic Gradient Descent with modified update rules (Equations 6 and 7), there are practical limitations that make SGD's use problematic (Hu et al., 2008). In particular, Hu's method incorporates unobserved ratings in the learning process by treating them as 0's with a low confidence, which makes the problem dense and gradient descent prohibitively expensive. Instead, Hu uses the method of *alternating least squares*, where Equation 5 is solved directly by fixing one of U, V and optimizing the other in turns until convergence (Koren et al., 2009; Hu et al., 2008). The resulting system works well, at the expense of a more complicated parameter estimation algorithm and increased runtime.

$$u_i \leftarrow u_i + \gamma \cdot (c_{i,j} \cdot e_{i,j} \cdot p_j - \lambda \cdot u_i) \tag{6}$$

$$p_j \leftarrow p_j + \gamma \cdot (c_{i,j} \cdot e_{i,j} \cdot u_i - \lambda \cdot p_j) \tag{7}$$

3 Implicit Music Feedback

Our work is based on implicit user feedback for a music dataset. For each user we know the number of times the user played songs by a given artist, but we do not have direct ratings for that artist. As a result, in order to use the matrix factorization approaches described earlier, ratings must be estimated from play count information. We define play frequency $freq$ for a given user i and artist j to be the user's play count for that artist normalized by user's total plays:

$$freq_{i,j} = \frac{count(i,j)}{\sum_{j'} count(i,j')} \tag{8}$$

We also adopt the notation $freq_k(i)$ to denote the frequency of the k-th most listened to artist for user i. As Figure 1 shows, play frequencies have a clear power law distribution. A rating for an artist with rank k is computed as a linear function of the frequency percentile:

$$r_{i,j} = 4 \cdot \left(1 - \sum_{k'=1}^{k-1} freq_{k'}(i)\right) \tag{9}$$

It follows that we assign a rating in the 3-4 range to the artists in the top 25% frequency percentile, a rating of 2-3 to the artists in the subsequent 25% percentile, and so on. The least frequently listened to artists have a rating close to 0.

The formulation in Equation 9 appears to transform the problem into the domain of explicit feedback, and makes it tempting to directly use unmodified algorithms from that domain. In fact, as we show in the Experiments section, such an approach does indeed quickly converge to a reasonable global error, but the result is of no practical use due to the pecularities of implicit music datasets. We describe these pecularities, and how we deal with them, below.

3.1 The Implicit Problem

One of the most significant differences between explicit and implicit feedback datasets is the distribution of ratings. In an explicit setting, the average user only rates a reasonably small subset of products, and the ratings are not heavily skewed towards one end or the other. This is in contrast to implicit music datasets and ratings derived from play frequencies, where most artists have a rating close to 0 - a consequence of a power law distribution (see Figure 1).

Such a skewed distribution has an impact on parameter estimation. A naive approach, i.e. directly optimizing Equation 1, is susceptible to getting stuck in a local minimum where, for each user, all available products have a low rating. While this is a reasonable approximation of the distribution, it is of little practical use, since in recommender systems we are interested in identifying products a user might actually like. We demonstrate this problem in the Experiments section, where we apply a Stochastic Gradient Descent algorithm, designed for explicit datasets, to implicit ratings derived from a Last.fm dataset.

3.2 Percentile Normalization

In order to deal with the skewed rating distribution, we propose a normalization scheme which amplifies the error (Equation 2) for highly rated artists. We accomplish this, for each user i, by assigning all artists to four non-overlapping bins based on the artists' rating (which in turn is based on the frequency percentile, see Equation 9):

$$B_i^1 = \{j : 4 \geq r_{i,j} > 3\} \tag{10}$$
$$B_i^2 = \{j : 3 \geq r_{i,j} > 2\} \tag{11}$$
$$B_i^3 = \{j : 2 \geq r_{i,j} > 1\} \tag{12}$$
$$B_i^4 = \{j : 1 \geq r_{i,j} > 0\} \tag{13}$$

We then normalize errors within each bin by the square root of the bin's size. Formally, we define the *percentile-normalized error* as:

$$e_{i,j}^{norm} = \frac{r_{i,j} - u_i^T p_j}{\sqrt{\|B_i^t\|}} \tag{14}$$

where B_i^t is the bin to which artist j belongs for user i. The normalization ensures that each individual bin's contribution to the total squared error has approximately the same weight, regardless of the bin's size. To make this clearer, suppose we always make a constant error ϵ on all samples. The total squared error for all samples in a bin t is then:

$$\sum_{r_{i,j} \in B_i^t} \left(\frac{r_{i,j} - u_i^T p_j}{\sqrt{\|B_i^t\|}} \right)^2 = \sum_{r_{i,j} \in B_i^t} \left(\frac{\epsilon}{\sqrt{\|B_i^t\|}} \right)^2 = \|B_i^t\| \cdot \frac{\epsilon^2}{\|B_i^t\|} = \epsilon^2$$

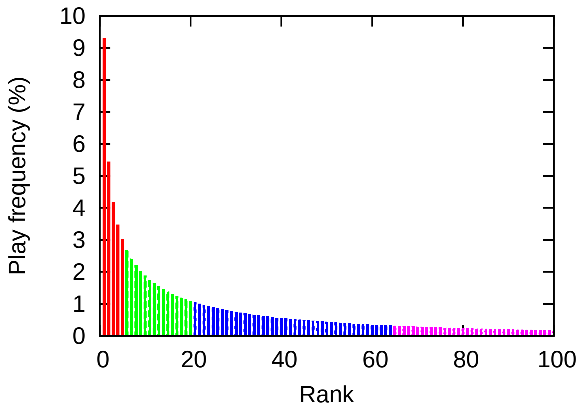

Figure 1: Artist play frequencies as a function of rank averaged over 47,000 Last.fm users. Frequencies have a clear power law distribution (with a negligible standard deviation between users), with the top 25% of all plays belonging to only 5 artists out of an average of 500 artists per user. The colors correspond to subsequent percentiles of size 25%. For clarity, only the first 100 top ranked artists are shown.

That is, the total error is the same for all bins, regardless of their size.

3.3 Parameter Estimation

We estimate the user and product matrices U and P using a variant of the Stochastic Gradient Descent algorithm modified to work with the normalized error. For each rating $r_{i,j}$ and its associated user and artist vectors u_i and p_j, we compute the normalized error $e_{i,j}^{norm}$ and update parameters according to:

$$u_i \leftarrow u_i + \gamma \cdot \left(e_{i,j}^{norm} \cdot p_j - \lambda \cdot u_i \right) \tag{15}$$
$$p_j \leftarrow p_j + \gamma \cdot \left(e_{i,j}^{norm} \cdot u_i - \lambda \cdot p_j \right) \tag{16}$$

We repeat the process for all ratings until convergence, or until a predefined maximum number of iterations.

4 Experiments

We report results on a Last.fm dataset. The dataset contains approximately 47,000 users with play counts for all artists they ever listened to. Since Last.fm does not itself publish datasets, we acquired the data on a per-user basis by walking the social graph, starting with 400 "recently active users" as the seed. The resulting dataset contains

1,559,512 unique artists and 27,440,327 individual play counts, and has a user-artist density of approximately 0.037%.

We split the dataset into 95% training and 5% test, and use the root mean square error (RMSE) as the evaluation criterion:

$$RMSE = \frac{\sum_{r_{i,j} \text{ known}} \left(r_{i,j} - u_i^T p_j\right)^2}{|\{r_{i,j} \text{ known}\}|} \qquad (17)$$

4.1 Unnormalized SGD

We report the performance of a Stochastic Gradient Descent algorithm, unmodified from the explicit feedback setting, applied to implicit Last.fm data. Unless noted otherwise, we used $\gamma = 0.0002$, $\lambda = 0.8$ and $f = 20$. Figure 2 shows the convergence of the error on the training and test sets. While the final training and test errors, 0.582793 and 0.709069, respectively, are reasonably low, they are misleading due to the high skewness of the ratings. We broke the RMSE down by percentiles, finding that SGD achieved an RMSE of 2.736185 for artists in bin 1 (play frequency percentile 25%), 1.877455 for artists in bin 2 (play frequency percentile 25%-50%), and 0.497912 for bins 3 and 4 (bottom 50%). Clearly, unmodified SGD overfits the low ratings due to the skewness of the distribution, making its use as a recommendation algorithm impractical.

We were unable to overcome the strong bias for low ratings by varying the parameters of the algorithm. Figure 3 shows that the cumulative RMSE does not change when increasing the number of features from $f = 8$ to $f = 128$. Similarly, RMSE errors within each bin were not affected by f either. We conducted similar evaluations for various learning rates λ and regularization factors γ with similar results. This demonstrates that the low-rating overfitting behavior is difficult to overcome without modifications to the algorithm.

4.2 Percentile-Normalized SGD

We report the performance of the Stochastic Gradient Descent algorithm using the percentile-normalized error, as described in Section 3.3. Unless noted otherwise, we used $\gamma = 0.0002$, $\lambda = 0.8$ and $f = 20$, the same as in the explicit case.

Figure 4 shows that the percentile-normalized

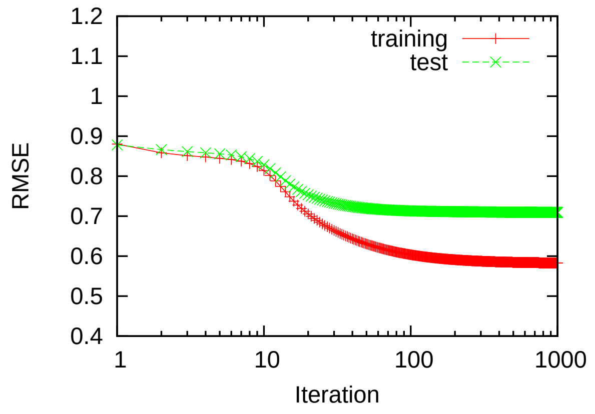

Figure 2: Convergence profile of a Stochastic Gradient Descent algorithm designed to work with explicit feedback on an implicit Last.fm dataset. Parameters are $\gamma = 0.0002$, $\lambda = 0.8$ and $f = 20$. Final errors are 0.582793 training and 0.709069 test.

SGD does not exhibit the low-rating overfitting behavior. While it has a greater cumulative RMSE compared to unnormalized SGD (0.772473 vs. 0.709069), it performs significantly better on the top 25% (1.599034 vs. 2.736185), top 25%-50% (1.051795 vs. 1.877455), with only a slight degradation on the bottom 50% (0.736469 vs. 0.497912).

In addition, as we show in Figures 5, 6 and 7, percentile-normalized SGD responds well to variation of parameters such as the number of features f, the learning rate γ and the regularization factor λ, potentially enabling fine-tuning for specific applications and requirements. Such fine-tuning is impossible for unnormalized SGD due to its strong overfitting behavior.

5 Practical Considerations

While the experimental performance of the percentile-normalized SGD is promising, RMSE is limited as an evaluation metric. To evaluate the real-world performance of our algorithm, we used it to make rcommendations to a small number of friends. The feedback we gathered was mixed. While the algorithm did recommend artists that our friends enjoyed, it also made a lot of spurious recommendations. For example, it would often recommend little known foreign artists.

Upon a closer analysis, we found that a great majority of spurious recommendations was of a partic-

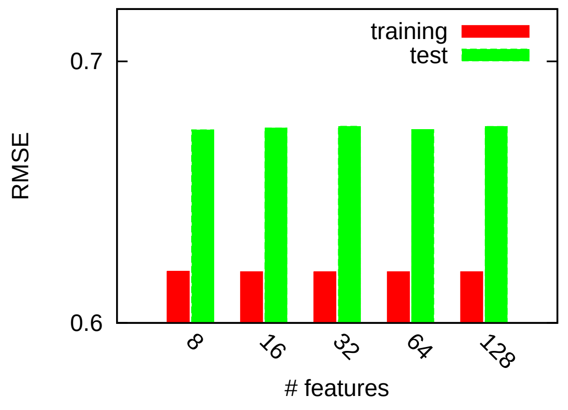

Figure 3: RMSE as a function of features for a Stochastic Gradient Descent algorithm designed to work with explicit feedback on an implicit Last.fm dataset. Parameters are $\gamma = 0.0002$, $\lambda = 0.8$. The number of features has a negligible effect on the error, both global (shown) and within bins (not shown).

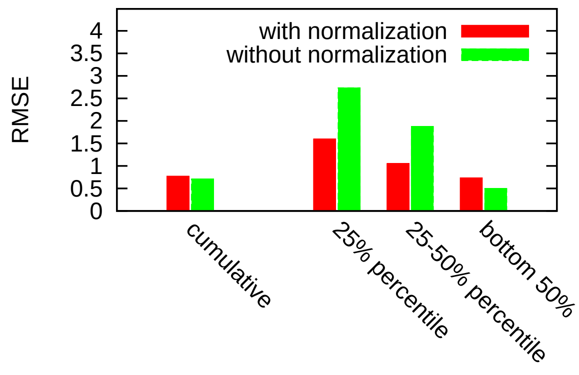

Figure 4: Cumulative and percentile RMSE attained by SGD with and without the percentile-normalized error. Parameters for both algorithms are $\gamma = 0.0002$, $\lambda = 0.8$ and $f = 20$. SGD with normalized error has a greater cumulative RMSE (0.772473 vs. 0.709069), but performs significantly better on the top 25% (1.599034 vs. 2.736185), top 25%-50% (1.051795 vs. 1.877455), with only a slight degradation on the bottom 50% (0.736469 vs. 0.497912).

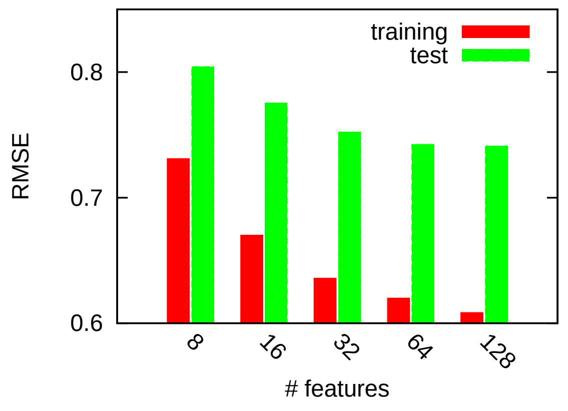

Figure 5: Training and test errors of the percentile-normalized Stochastic Gradient Descent algorithm for different numbers of features f. The performance of the algorithm improves as the number of features increases.

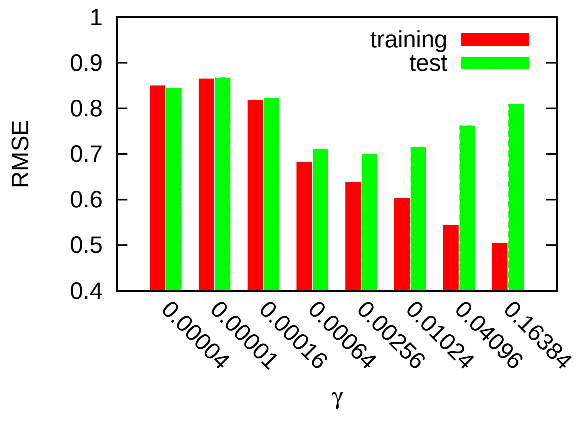

Figure 6: Training and test errors of the percentile-normalized Stochastic Gradient Descent algorithm for different learning rates γ. The (limited) overfitting behavior can be controlled by varying γ.

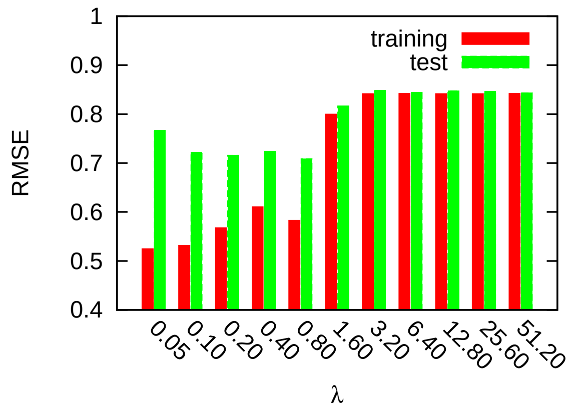

Figure 7: Training and test errors of the percentile-normalized Stochastic Gradient Descent algorithm for different regularization parameters λ.

ular type. Namely, the system would recommend artists that very few Last.fm users listened to. We suspect the feature vectors p_j for such artists overfit the few users who did listen to them, and did not represent the artists' true qualities. This intuition is supported by the update rules in Equations 15 and 16 - a given artist's feature vector is only updated for each observed rating.

To overcome this behavior, we changed the algorithm to only recommend artists with a total of more that 2000 plays by at least 20 users. This ensured that the feature vectors p_j for those artists did not considerably ovefit any particular user. While we set these thresholds arbitrarily, we discovered that they significantly improved the quality of recommendations.

6 Conclusions

We have demonstrated that a standard Stochastic Gradient Descent algorithm designed for explicit feedback datasets does not work with implicit music data. We proposed a variant of the SGD algorithm that uses the *percentile-normalized error*, and demonstrated its superior performance on a large Last.fm dataset. We further evaluated the performance of our algorithm as a function of the number of features, the learning rate and the normalization factor, showing that it can be fine-tuned if required. We also performed a limited evaluation of the algorithm in a real world setting, discovering problems not evident from RMSE-only evaluations.

There is a number of areas that warrant future work. First, it would be worthwile to investigate how well our approach generalizes to other implicit datasets. Second, we would be interested in a larger real-world evaluation of the percentile-normalized SGD, where recommendations are made to real Last.fm users who are then asked for feedback. Finally, our approach could be used with other matrix factorization algorithms such as Alternating Least Squares, by substituing the percentile-normalized error.

A Implementation details

We have made our implementation available under the GPLv3 license on the author's website[2]. Per Last.fm's Terms of Service, we were unable to distribute the dataset with the code. However, we provided the tools to automatically build such a dataset given a Last.fm API key which can be obtained at `http://www.last.fm/api`.

References

[Koren et al. 2009] Yehuda Koren, Robert Bell, Chris Volinsky. August 2009. *Matrix Factorization Techniques for Recommender Systems*. Computer, v.42 n.8, p. 30-37

[Hu et al. 2008] Yifan Hu, Yehuda Koren, Chris Volinsky. 2008. *Collaborative Filtering for Implicit Deedback Datasets*. Proc. IEEE International Conference on Data Minining (ICDM 08), IEEE CS Press, pp. 263-272

[Funk 2006] Simon Funk. December 2006. *Netflix Update: Try This at Home.* *`http://sifter.org/~simon/journal/20061211.html`*

[2] `http://people.csail.mit.edu/mpacula/classes/6.867/final_project.tar.gz`