The Role of Unknown Interactions in Implicit Matrix Factorization — A Probabilistic View

Joey De Pauw
joey.depauw@uantwerpen.be
University of Antwerp
Antwerp, Belgium

Bart Goethals
bart.goethals@uantwerpen.be
University of Antwerp
Antwerp, Belgium
Monash University
Melbourne, Australia

ABSTRACT

Matrix factorization is a well-known and effective methodology for top-k list recommendation. It became widely known during the Netflix challenge in 2006, and since then, many adapted and improved versions have been published. A particularly interesting matrix factorization algorithm called iALS (for implicit Alternating Least Squares) adapts the method for implicit feedback, i.e. a setting where only a very small amount of positive labels are available along with a majority of unknown labels. Compared to the classical task of rating prediction, learning from implicit feedback is applicable to many more domains, as the data is more abundant and requires less effort to elicit from users. However, the sparsity, imbalance, and implicit nature of the signal also pose unique challenges to retrieving the most relevant items to recommend.

We revisit the role of unknown interactions in implicit matrix factorization. Traditionally, all unknowns are interpreted as negative samples and their importance in the training objective is then downweighted to balance them out with the known, positive interactions. Interestingly, by adapting a probabilistic view of matrix factorization, we can retain the unknown nature of these interactions by modelling them as either positive or negative. With this new formulation that better fits the underlying data, we gain improved performance on the downstream recommendation task without any computational overhead compared to the popular iALS method.

This paper outlines the key insights needed to adapt iALS to use logistic regression. Furthermore, a logistic version of the popular full-rank EASE model is introduced in a similar fasion. An extensive experimental evaluation on several real-world datasets demonstrates the effectiveness of our approach. Additionally, a discrepancy between the need for weighting between factorization and autoencoder models is discovered, leading towards a better understanding of these methods.

CCS CONCEPTS

• **Information systems** → *Learning to rank*; **Recommender systems**; *Collaborative filtering*.

KEYWORDS

recommender system, collaborative filtering , implicit feedback, matrix factorization

ACM Reference Format:
Joey De Pauw and Bart Goethals. 2024. The Role of Unknown Interactions in Implicit Matrix Factorization — A Probabilistic View. In *18th ACM Conference on Recommender Systems (RecSys '24), October 14–18, 2024, Bari, Italy*. ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3640457.3688100

1 INTRODUCTION

An important component, contributing to the success of most modern recommender systems, is collaborative filtering from implicit feedback data [1]. In this approach, the preferences of one user are estimated based on the collection of all users' past behaviour. The data is both easy to collect and abundant, as it is generated by users' interactions with the system. However, the data is also noisy and sparse. Unlike explicit feedback data, where low ratings can be interpreted as negative feedback, in binary implicit feedback, all items that a user interacted with are assumed to be positive and the vast remainder of the catalogue as unknown. This makes it challenging to train a model that can accurately predict the user preferences [1].

One of the most popular collaborative filtering methods, that can learn from implicit feedback data, is implicit Alternating Least Squares (iALS) [8, 17]. iALS, often also called WMF or WRMF, is a matrix factorization method that estimates the user preferences by factorizing the user-item interaction matrix into lower-rank user and item embedding matrices. In this method, all unknown interactions are interpreted as negative samples and their importance in the training objective is then downweighted to balance them out with the known, positive interactions. However, this approach does not fully capture the uncertainty in the data, as the unknown interactions could be either positive or negative.

To address this issue, we propose a new probabilistic interpretation of iALS called LogWMF. In this probabilistic interpretation, unknown interactions are modelled as either positive or negative, to better capture the uncertainty in the data. This new formulation allows us to gain improved performance on the downstream recommendation task without any computational overhead compared to iALS. An adaptation of the full-rank EASE model [24] with probabilistic unknowns (LogEASE) is also included for completeness. Compared to the closed-form solution of EASE, training LogEASE is less practical in a production setting. However, we gain a better understanding of the importance of weighting by studying its performance in an offline evaluation.

Section 2 presents the background of iALS and introduces the new probabilistic interpretation. Section 3 describes the experimental validation of our approach. Section 4 contains a review of related work, and finally, Section 5 concludes the paper.

2 MODELS

2.1 Background

We use m and n to denote the amount of users and items respectively and k for the embedding dimension. The matrices $P \in \mathbb{R}^{m\times k}$ and $Q \in \mathbb{R}^{n\times k}$ are the embedding matrices that factorize the binary interaction matrix $X \in \{0,1\}^{m\times n}$. The loss of iALS is given by:

$$\mathcal{L}^{(iALS)} = \sum_{u}^{m}\sum_{i}^{n} W_{u,i}\left(X_{u,i} - P_u Q_i^\top\right)^2 + \lambda\left(\|P\|_F^2 + \|Q\|_F^2\right) \tag{1}$$

where the weights $W_{u,i}$ are chosen to be uniform (α_0) for all missing interactions and one for all positive interactions.

iALS is typically solved with Alternating Least Squares (ALS), where the item factors are kept fixed when the user factors are updated and vice versa. This is possible because the loss is bi-convex in terms of the user or item factors. The update steps can even be derived in closed form for fast learning:

$$P_u = X_u\,\mathrm{dm}(W_u)Q\left(Q^\top \mathrm{dm}(W_u)Q + \lambda I\right)^{-1}$$
$$Q_i = X_{\cdot,i}^\top\,\mathrm{dm}(W_{\cdot,i})P\left(P^\top \mathrm{dm}(W_{\cdot,i})P + \lambda I\right)^{-1}$$

with $\mathrm{dm}(\cdot)$ constructing the diagonal matrix from the given vector. The update formulas are derived by taking the derivative of the loss with respect to the user or item factors and setting it to zero. Without loss of generality, in the following we only describe the optimization of the user factors. The item factors are learned analogously.

The efficiency and high scalability of this method arises from the structure of the weights. Since all missing interactions are assigned the same weight α_0, we can rewrite the weights as $W_u = X_u + \alpha_0(1 - X_u)$. Subsequently, the inner product $Q^\top \mathrm{dm}(W_u)Q$ can efficiently be computed as:

$$\alpha_0 Q^\top Q + Q^\top \mathrm{dm}\left(X_u(1-\alpha_0)\right)Q \tag{2}$$

Here, the first term is shared between all users and can be precomputed. The second term is only based on the positive interactions and can be computed efficiently by taking the outer product between the factors of Q corresponding to positive interactions. The total complexity of one iteration of learning all user and item embeddings with iALS is $O(pk^2 + (m+n)k^3)$ with p the total amount of positive interactions.

Alternatively, previous work pointed out that computing the exact optimum at each iteration is not necessary [2, 18, 19]. Indeed, every iteration the optimum is overwritten anyway, so using a sufficiently accurate approximation inside the iteration of ALS can speed up the computation with barely any loss in convergence speed.

Newton's method is a popular choice for approximate optimization. It requires the first and second derivative of the loss. These are also called the Jacobian ($\vec{j}$) and Hessian (H) matrices. One update step of Newton's optimization method consists of subtracting the inverse of the Hessian multiplied with the Jacobian:

$$P_u^\top = P_u^\top - H^{-1}\vec{j} \tag{3}$$

Finally, as described in [19], we can reach even better scalability by using subvector optimization. In this method, we optimize the factors piece by piece in small subsets of the weights determined by the *blocksize*. The *blocksize* is typically between 64 and 256 to leverage efficient use of low-level vector instructions. Its optimal value is hence mostly hardware dependent. We iterate by updating a specific block π of the embedding for all of the factors before switching to the next block. The formulas for the Jacobian and Hessian are found by computing the first and second partial derivatives:

$$\frac{\partial \mathcal{L}^{(iALS)}}{\partial P_{u,\pi}^\top} = \vec{j} = -Q_{\cdot,\pi}^\top\,\mathrm{dm}(X_u)\left(\vec{1}^\top - P_u Q^\top\right)^\top + \alpha_0 Q_{\cdot,\pi}^\top\,\mathrm{dm}(\vec{1}^\top - X_u)QP_u^\top + \lambda P_{u,\pi}^\top$$

$$\frac{\partial^2 \mathcal{L}^{(iALS)}}{\partial P_{u,\pi}\partial P_{u,\pi}^\top} = H = Q_{\cdot,\pi}^\top\,\mathrm{dm}\Big(X_u + \alpha_0\left(\vec{1}^\top - X_u\right)\Big)Q_{\cdot,\pi} + \lambda I$$

The complexity of one epoch is $O(pk|\pi| + (m+n)(k^2 + k|\pi|^2))$. For more details and a full overview of these methods, refer to [19].

2.1.1 *Maximum Likelihood Estimation.*

In this section, we show that the square loss objective of iALS can equivalent be written as a Maximum Likelihood Estimation (MLE). Though the two formulations are equivalent, the MLE formulation is easier to extend with the logistic function. In MLE, we maximize the likelihood of the data given the model. The likelihood of the data in this case is the product of the probabilities of each interaction. For linear models, probabilities are modelled with a Gaussian function: $g(x) = \exp\left(-x^2\right)$, leading us to the following MLE for iALS:

$$\arg\max_{P,Q} \prod_{u,i} g\left(1 - P_u Q_i^\top\right)^{X_{u,i}} g\left(P_u Q_i^\top\right)^{\alpha_0\cdot(1-X_{u,i})} \tag{4}$$

To facilitate optimization, we take the negative of the natural logarithm (ln) of the likelihood. This does not change the optimum since the logarithm is continuous and monotonically increasing. The equivalent loss to minimize is then the negative log-likelihood:

$$\mathcal{L}^{(MLE)} = -\sum_{u,i} \ln\left(g\left(1 - P_u Q_i^\top\right)^{X_{u,i}}\right) + \ln\left(g\left(P_u Q_i^\top\right)^{\alpha_0\cdot(1-X_{u,i})}\right) + \lambda\cdot\left(\|P\|_F^2 + \|Q\|_F^2\right)$$

Regularization is added to the loss to prevent overfitting. Note that this corresponds to placing a Gaussian prior on the weights in Equation (4). Finally, we can see that the logarithm cancels out the exponential function of the Gaussian g, and we can simplify the loss to Equation (1) as follows:

$$\mathcal{L}^{(MLE)} = \sum_{u,i} X_{u,i}\left(1 - P_u Q_i^\top\right)^2 + \alpha_0(1 - X_{u,i})\left(P_u Q_i^\top\right)^2 + \lambda\cdot\left(\|P\|_F^2 + \|Q\|_F^2\right)$$
$$= \mathcal{L}^{(iALS)} \qquad \square$$

2.1.2 *Regression vs Classification.*

Given that we are modelling binary labels in the interaction matrix X, we argue that the Gaussian function may not be the most appropriate choice. This can be seen by first noting that the Gaussian function corresponds to linear regression as shown above and as proven in the literature on generalized linear models [14]. Linear regression is useful for its simplicity and efficiency. However, if we consider the problem as a classification task, we encounter two issues.

First, the domain is not bounded between 0 and 1. This means that the model is enforced to predict values around the arbitrarily chosen targets of 0 and 1. With the downstream task where we construct a ranked list based on predicted relevance, it is perhaps counter-intuitive to also penalize positive values with a score above 1 and unknowns with a score below 0.

Second, the output is not interpretable as a probability. Indeed, the domain of the regression is not bounded, so the computed scores cannot be interpreted as probabilities. Even if we use the interpretation with the Gaussian functions of Equation (4), it is clear that the "probabilities" for positive and negative do not sum to one.

We can overcome these issue by phrasing the problem as a classification task, i.e. interpreting the labels as classes and modelling the probabilities of a sample belonging to each class. For binary classification, the logistic function is typically used:

$$\sigma(x) = \frac{1}{1 + \exp(-x)}$$

It projects real values to the interval $[0, 1]$ with a monotonically increasing sigmoid curve. Logistic regression uses the logistic function to model the probability of a sample belonging to the positive class. The probability of the negative class is simply $1 - \sigma(x)$.

2.2 Modelling Unknown Interactions

A simple, yet impractical approach to achieve implicit matrix factorization with logistic regression would be to directly use the logistic function for both the positive and unknown interactions. In this case, the unknowns are assumed to be negative interactions and their probability is modelled as $1 - \sigma(P_u Q_i^\top)$. However, the optimization of this loss scales with the size of the matrix $m \times n$, i.e. all combinations of users and items, which is not feasible for large datasets. The reason for this is that the efficient precomputation step outlined in Equation (2) of the iALS optimization is no longer possible when the logistic function is used to model negative interactions. A negative sampling approach is needed to make the optimization feasible as previously shown in [10].

Instead, we propose to use the logistic function to model the positive interactions and to model the unknowns as a product of the probabilities of the positive and negative classes. Formally:

$$\arg\max_{P,Q} \prod_{u,i} \sigma\left(P_u Q_i^\top\right)^{X_{u,i}} \cdot \left(\left(1 - \sigma\left(P_u Q_i^\top\right)\right) \sigma\left(P_u Q_i^\top\right)\right)^{\alpha_0 \cdot (1 - X_{u,i})}$$

There are multiple ways to interpret this formulation. First, the product of the positive and negative probabilities is maximal when both are equal to 0.5. In this sense, unknowns are modelled as equally likely to be positive as negative. Deviating from this balance is penalized, as this indicates that the model is too certain about the unknowns. This also motivates why weights are kept in the loss, as there are still many more unknown than positive interactions in the data and we want to balance out their impact on the learned weights.

Secondly, the new formulation can be interpreted as a three-class classification problem as shown in the top part of Figure 1. To make the probabilities of the three classes sum to one, we use the squared logistic function for the positives instead. This is equivalent to adding a weight to the positive samples, as the exponent becomes

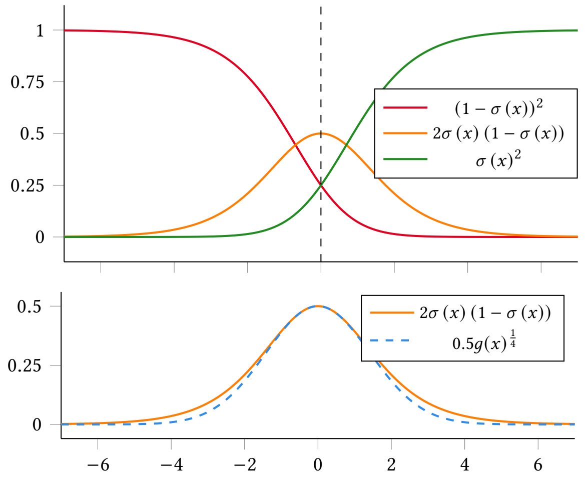

Figure 1: Plot of the three-class classification interpretation of the logistic function. The top plot shows the probabilities of the three classes: positive (green), unknown (orange), and negative (red). The bottom plot shows the approximation of the logistic function with the Gaussian function.

a scalar factor in the negative log-likelihood. It furthermore has no impact on the learned model as the same solution can be found under different optimal hyperparameters. With the three-class formulation, we can also model negatives explicitly as a separate class independent of the unknowns. For the scope of this paper however, we only consider binary implicit feedback data that is limited to positive interactions and unknowns.

Optimizing this loss as-is would again scale poorly, as it remains dependent on all combinations of users and items $m \times n$. However, we can now approximate the function used for the unknown class with a Gaussian function. The bottom part of Figure 1 illustrates the similarity between the two functions when a specific scalar and exponent are used. In practice the scalar has no impact on the loss and the exponent can be consumed by the hyperparamers so they can be safely ignored. Using the Gaussian function for approximation is inspired by the commonly used approximation of the logistic function by the cumulative distribution function (CDF) of a Gaussian distribution [4, 23]. In fact, the derivative of the gaussian CDF is the gaussian function and the derivative of $\sigma(x)$ is $\sigma(x)(1 - \sigma(x))$, which further supports the validity of this approximation. Conveniently, as shown in Section 2.1.1, the MLE with a Gaussian function is equivalent to a square loss.

To summarize, we find that when positive interactions are modelled with a sigmoid function, intuitively it makes sense that unknowns are modelled as a product of the positive and negative probabilities. This in turn can be approximated by a Gaussian function which equates to linear regression. With these insights we develop LogWMF in the next section.

2.3 LogWMF

Putting everything together, we define the the maximum likelihood estimation for LogWMF as follows:

$$\arg\max_{P,Q} \prod_{u,i} \sigma\left(P_u Q_i^\top\right)^{2X_{u,i}} g\left(P_u Q_i^\top\right)^{\alpha_0 \cdot (1 - X_{u,i})}$$

Similar to the trick of weighting with iALS, we only use the more complex logistic function for the positive interactions. The missing interactions are modelled as unknowns and approximated with the Gaussian function. For optimization, we use the equivalent negative log-likelihood loss with regularization added:

$$\mathcal{L} = -\sum_{u,i} 2X_{u,i} \ln\left(\sigma\left(P_u Q_i^\top\right)\right) + \alpha_0 \cdot (1 - X_{u,i}) \cdot \ln\left(g\left(P_u Q_i^\top\right)\right) + \lambda \cdot (\|P\|_F^2 + \|Q\|_F^2)$$

Compared to iALS, optimization with alternating least squares is not possible as there is no closed form solution for the user and item factors. We immediately derive the faster and more scalable subvector optimization with Newton's method instead.

$$\frac{\partial \mathcal{L}}{\partial P_{u,\pi}^\top} = \vec{j} = -Q_{\cdot,\pi}^\top \operatorname{dm}(X_u)\left(\vec{1}^\top - \sigma\left(P_u Q^\top\right)\right)^\top + \alpha_0 Q_{\cdot,\pi}^\top \operatorname{dm}(\vec{1}^\top - X_u) Q P_u^\top + \lambda P_{u,\pi}^\top$$

$$\frac{\partial^2 \mathcal{L}}{\partial P_{u,\pi} \partial P_{u,\pi}^\top} = H = Q_{\cdot,\pi}^\top \operatorname{dm}\Bigg(X_u \odot \sigma\left(P_u Q^\top\right) \odot \left(\vec{1}^\top - \sigma\left(P_u Q^\top\right)\right) + \alpha_0\left(\vec{1}^\top - X_u\right)\Bigg) Q_{\cdot,\pi} + \lambda I$$

The scalar multiplier of two in both $\vec{j}$ and H was omitted as it is divided away in the optimization step, Equation (3). Also notice that $Q_{\cdot,\pi}^\top \operatorname{dm}(\vec{1}^\top - X_u)Q$, which appears in J (and its subset in H), can be computed efficiently as described in Equation (2). Indeed, except for applying the logistic function to the predicted scores and reusing this result in the Hessian, this optimization requires no additional computation compared to the subvector optimization of iALS. It has the same training complexity $O(pk|\pi| + (m+n)(k^2 + k|\pi|^2))$. At inference time, a top-k list per user can be constructed directly with the dot product between user and item vectors. Since the logistic function is monotonically increasing, applying it would not change the relative order of the predictions. In summary, existing production systems based on iALS can easily be adapted to leverage LogWMF.

As a side note, previous work has shown that the loss of iALS can be simplified by computing the term for missing values over all user-item combinations, rather than only the missing ones [2]. This results in a loss where the labels X are scaled by a factor depending on α_0. For linear regression, this leads to a rescaled loss that has the same optimal solution. However, the optimum will be found with different, rescaled versions of the optimal hyperparameters. With the addition of the logistic function however, we need the labels X to remain binary, and as such, this simplification cannot be applied.

2.4 Regularization Scaling

For simplicity of notation, we used a uniform regularization scalar λ for all user and item factors. In practice, it is often found that scaling the regularization strength based on the number of positive interactions and/or unknowns can lead to better performance. The optimization is otherwise identical, except a specific regularization value λ_u or λ_i is used for each user or item factor.

A first simple strategy, which we refer to as *frequency scaling*, is to multiply the fixed regularization constant λ with the number of positive interactions of a user (or item) to the power ν:

$$\lambda_u = \lambda \cdot \left(\sum_i X_{u,i}\right)^\nu$$

Alternatively, the unknowns can also be taken into account with their corresponding weight α_0 in the *weighted scaling* strategy:

$$\lambda_u = \lambda \cdot \left(\alpha_0 \cdot n + \sum_i X_{u,i}\right)^\nu$$

The hyperparameter ν is typically optimized between 0 and 1 with a grid search. In both strategies, a value of 0 means that the regularization strength is independent of X and reduces back to λ.

2.5 Weighted EASE and LogEASE

EASE [24] is a popular recommendation model because of its simple formulation and closed form solution. It computes a linear regression model with the following loss:

$$\mathcal{L}^{(EASE)} = \|X - XB\|_F^2 + \lambda \|B\|_F^2 \qquad \text{s.t. } \operatorname{diag}(B) = 0$$

Where B is a square full-rank matrix of item-item weights and its diagonal is set to zero to prevent learning predictions based on self-similarity. The closed-form solution of EASE leads to fast training times because no iteration is needed. However, for this solution to work, the loss needs to remain sufficiently simple. For example, adding item-specific regularization or weights breaks the closed form solution. In this section we show that the subvector optimization of iALS can be adapted to efficiently learn a weighted and probabilistic version of EASE in two steps.

First, note that the square matrix B of EASE can be viewed as an item embedding matrix with the corresponding 'user embeddings' X. Hence, if we want the learned item embeddings $Q^\top$ to correspond to the item-item weights B of EASE, we set the embedding size to full rank ($k = n$), and instead of learning P, we fix the user embeddings to X.

Secondly, the diagonal constraint needs to be added. For the subvector optimization procedure this can be done by adding the constraint to Newton's method. Recall that weights are updated per item i and block π independently. The Jacobian $\vec{j}$ and Hessian H of $Q_{i,\pi}$ w.r.t. the loss are used to perform one update step $Q_{i,\pi}^\top = Q_{i,\pi}^\top - \vec{x}$ with $H\vec{x} = \vec{j}$. If we assume a feasible start where the diagonal elements are zero, enforcing the constraint boils down to ensuring the update direction $\vec{x}$ also has a zero in the right position when $i \in \pi$. For this, we expand the linear system of equations H to include the constraint as follows:

$$\begin{bmatrix} H & I_{\cdot,d} \\ I_d & 0 \end{bmatrix} \begin{bmatrix} x \\ w \end{bmatrix} = \begin{bmatrix} \vec{j} \\ 0 \end{bmatrix}$$

Here, I is the identity matrix of size $|\pi|$ and d is the index corresponding to the diagonal element $Q_{i,i}$ within the block. The dual variable w does not need to be update in the case of a feasible start, and thus, simply solving this system for $\vec{x}$ makes sure the diagonal elements remain zero. With this adaptation, we can learn a weighted version of EASE with the subvector optimization of iALS. This model is referred to as Weighted EASE (WEASE). Similarly, we also adapt our LogWMF model to a full rank version called LogEASE.

3 EXPERIMENTS

To assess the performance of the proposed models, we conducted empirical experiments on two widely studied benchmarks: the MovieLens 20M (ML20M) [6], and the Million Song Dataset (MSD) [3]. Liang et al. [12] established these benchmarks for top-k item recommendation with implicit feedback and they have been used in numerous studies to evaluate the performance of recommendation algorithms. By using the same configuration, which is summarized in Section 3.1, we ensure a fair comparison with existing methods.

3.1 Experimental Setup

The benchmark of [12] consists of two datasets:

ML20M [6]: The MovieLens 20M dataset contains 20 million ratings of movies. During preprocessing only positive (> 3) ratings are retained as implicit feedback and inactive users and items are filtered. The final benchmark contains 136.677 users, 20.108 items and 10 million interactions.

MSD [3]: This data contains user-song play counts. Play counts are binarized and interpreted as implicit preference data. After preprocessing, the dataset contains 571.355 users, 41.140 items and 33.6 million interactions.

In the benchmark, first a validation and test set are split from the data. The validation set is used to tune the hyperparameters of the algorithms, while the test set is used to independently evaluate the performance. Strong generalization is used, meaning that the users in these sets are distinct from those in the training set. For the MovieLens 20M dataset, 10.000 users are randomly moved into the validation set and 10.000 to the test set. For the Million Song Dataset, they contain 50.000 users each. Then, to evaluate the quality of the recommendations, 80% of the interactions of each user are given as history to the model and the remaining 20% are used as ground truth to compute the metrics.

The evaluation metrics used, are Recall@20, Recall@50, and NDCG@100. Recall@k measures the proportion of relevant items that are ranked in the top-k positions, while NDCG@k is a ranking metric that additionally considers the position of the relevant items in the top-k positions. The reported results are averages over 5 runs on the test set. Each method is found to converge well-within 16 epochs, which we used as the maximum amount. All optimal parameters with respect to the Recall@20 metric are listed in Table 1 as determined by a grid search on the validation set. For more details on the datasets and the preprocessing steps, refer to [12] and the source code of our algorithms and experimental setup: https://github.com/JoeyDP/LogWMF.

3.2 Results

Table 2 shows the results of the experiments sorted by Recall@20 scores. It contains results by previous works that use the benchmark of [12], along with our own results for LogWMF and LogEASE, and reproduced results of iALS. Considering that the performance of most recommendation models is highly dependent on finding the right hyperparameters [21], we believe this to be the most fair comparison. Hyperparameter tuning often requires an extensive understanding of the underlying algorithms [20, 21]. Additionally,

Table 1: Best hyperparameters based on Recall@20. Values with a box are fixed for the algorithm and not optimized.

Dataset	Algorithm	α_0	λ	ν
ML20M	LogWMF	0.003	1	0.25
	iALS	0.1	0.003	1
	MF	$\boxed{1}$	0.1	1
	MF (uniform reg)	$\boxed{1}$	30	$\boxed{0}$
	LogEASE	0.01	10	0
MSD	LogWMF	0.003	0.01	1
	iALS	0.1	0.03	1
	MF	$\boxed{1}$	0.1	1
	MF (uniform reg)	$\boxed{1}$	30	$\boxed{0}$
	LogEASE	0.05	10	0

in reproducing algorithms, implementation errors or misinterpretations of the original method, can also cause major differences in the measured performance [7]. By comparing with the best results of previous works, we ensure that the algorithms are evaluated under the best possible conditions and that there is no bias where the proposed method is tuned better than the baselines.

First, we observe that LogWMF and LogEASE outperform their linear counterparts (iALS and EASE) on both datasets. On the MSD datasets, LogEASE is even the best performing algorithm of all the baselines reported in [20]. Though the difference in metrics is not large, we can still draw two conclusions from this result. On the one hand, this validates the proposed approach. Introducing the logistic function and modelling the missing values as unknowns work well together to improve on the linear model. On the other hand it is striking that, despite only needing to add a single application of the logistic function to the training procedure, the algorithm is better able to capture the underlying data.

Secondly, we experimented with different ways of scaling the regularization terms. The results of previous work are based on *weighted scaling*, where the weights α_0 is included in the formula (see Section 2.4). This leads to smoother, more uniform regularization values compared to *frequency scaling*, which is purely based on the occurrence counts. For iALS, on the ML20M dataset, we found that the weighted scaling performed better, while on the MSD dataset the frequency scaling performed better. For LogWMF on the other hand, the frequency scaling always performed best and the difference between the two strategies was much smaller compared to iALS.

3.2.1 Ablation Study. The plots in Figure 2 dive deeper into how different versions of matrix factorization perform under different embedding dimensions. In addition to the iALS and LogWMF models that are explained in Section 2, we also compare with standard a Matrix Factorization model (MF) where no weighting is applied to the unobserved values. For this model there are two variants: one with uniform regularization (no scaling), and one with frequency scaling. The results in Figure 2 were obtained on the test set with the hyperparameters of Table 1 as determined on the validation set with an embedding dimension of 4096 for ML20M and 8192 for the MSD dataset similar to [20].

Table 2: Quality results on the ML20M and MSD benchmark sorted by Recall@20 scores.

Dataset	Method	Recall@20	Recall@50	NDCG@100	Result from
ML20M	RecVAE [22]	0.414	0.553	0.442	[22]
	H+Vamp (Gated) [11]	0.413	0.551	0.445	[11]
	RaCT [13]	0.403	0.543	0.434	[13]
	LogWMF	**0.401**	**0.538**	**0.432**	**our result**
	LogEASE	**0.396**	**0.528**	**0.426**	**our result**
	Mult-VAE [12]	0.395	0.537	0.426	[12]
	LambdaNet [5]	0.395	0.534	0.427	[22]
	iALS [8, 17]	0.395	0.532	0.425	[20] and reproduced
	EASE [24]	0.391	0.521	0.420	[24]
	CDAE [27]	0.391	0.523	0.418	[12]
	Mult-DAE [12]	0.387	0.524	0.419	[12]
	SLIM [16]	0.370	0.495	0.401	[12]
	WARP [26]	0.314	0.466	0.341	[22]
	Popularity	0.162	0.235	0.191	[24]
MSD	**LogEASE**	**0.336**	**0.430**	**0.392**	**our result**
	EASE [24]	0.333	0.428	0.389	[24]
	LogWMF	**0.315**	**0.414**	**0.374**	**our result**
	iALS [8, 17]	0.311	0.409	0.369	our result
	iALS [8, 17]	0.302	0.403	0.360	[20]
	RecVAE [22]	0.276	0.374	0.326	[22]
	RaCT [13]	0.268	0.364	0.319	[13]
	Mult-VAE [12]	0.266	0.364	0.316	[12]
	Mult-DAE [12]	0.266	0.363	0.313	[12]
	LambdaNet [5]	0.259	0.355	0.308	[22]
	WARP [26]	0.206	0.302	0.249	[22]
	CDAE [27]	0.188	0.283	0.237	[12]
	Popularity	0.043	0.068	0.058	[24]

Due to the hyperparameters not being tuned independently of the embedding dimension, it is possible for some models to achieve slightly higher results on the lower dimensions. We ran a grid search for each dimension independently with LogWMF and one variant of iALS and found that the optimal hyperparameters tend to be the same or very close for all dimensions. Combined with the fact that we are mostly interested in the highest possible performance of each method, which is often achieved with the highest reported embedding dimension, we chose to reduce the computational overhead and complexity of our experiments by only tuning the hyperparameters once. Only for the MF model with uniform regularization on the ML20M Dataset did this cause a significant difference. Namely, with $\lambda = 100$ instead of 30, the model would produce a smooth curve that is more in line with the other models, however, slightly worse on the highest embedding dimension, where the hyperparameters are determined. In any case, this does not impact the conclusions drawn from the results.

Notice that in order of increasing model complexity, we first have the simplest MF (Uniform Reg.), followed by MF that adds frequency scaling to the regularization. Then iALS extends this with weights for unobserved values, and finally LogWMF adds the logistic function to the model. The results show that, in general, complexer models are able to achieve better performance, which is no surprise. Having more ways to adapt to the dataset by tuning the corresponding hyperparameters, typically allows for a better fit. On the ML20M dataset, this pattern is clearly visible for all metrics. On the MSD dataset however, we find that for the embedding dimension of 8192, the simpler MF model with frequency scaling outperforms the more complex iALS model with weighted scaling and it is on-par with iALS with frequency scaling. Note that the former is a special case of the latter with α_0 fixed to 1 and so it cannot be better. LogWMF consistently outperforms all other models on both datasets, which confirms the results from the previous section.

Our results can also be placed in the context of the theoretical work by Jin et al. [9]. They studied a closed-form solution for the simplest matrix factorization model, and discovered that the difference between MF and a full-rank autoencoder boils down to how they scale the eigenvalues of the user-item matrix to compute their recommendations. In light of this work, it is possible that the more complex factorization models have more favourable ways of scaling the eigenvalues, which could be what allows them to perform better. However, since there is no exact formula for their global optimum, we cannot make the same derivation to compare these methods.

3.2.2 *WEASE and LogEASE.*

In a final set of experiments, we take a closer look at the use of unobserved weighting in full-rank models. To the best of our knowledge, no efficient implementation for Weighted EASE (WEASE) has been found before. The main bottleneck is needing to compute the inverse of a rank n matrix for each

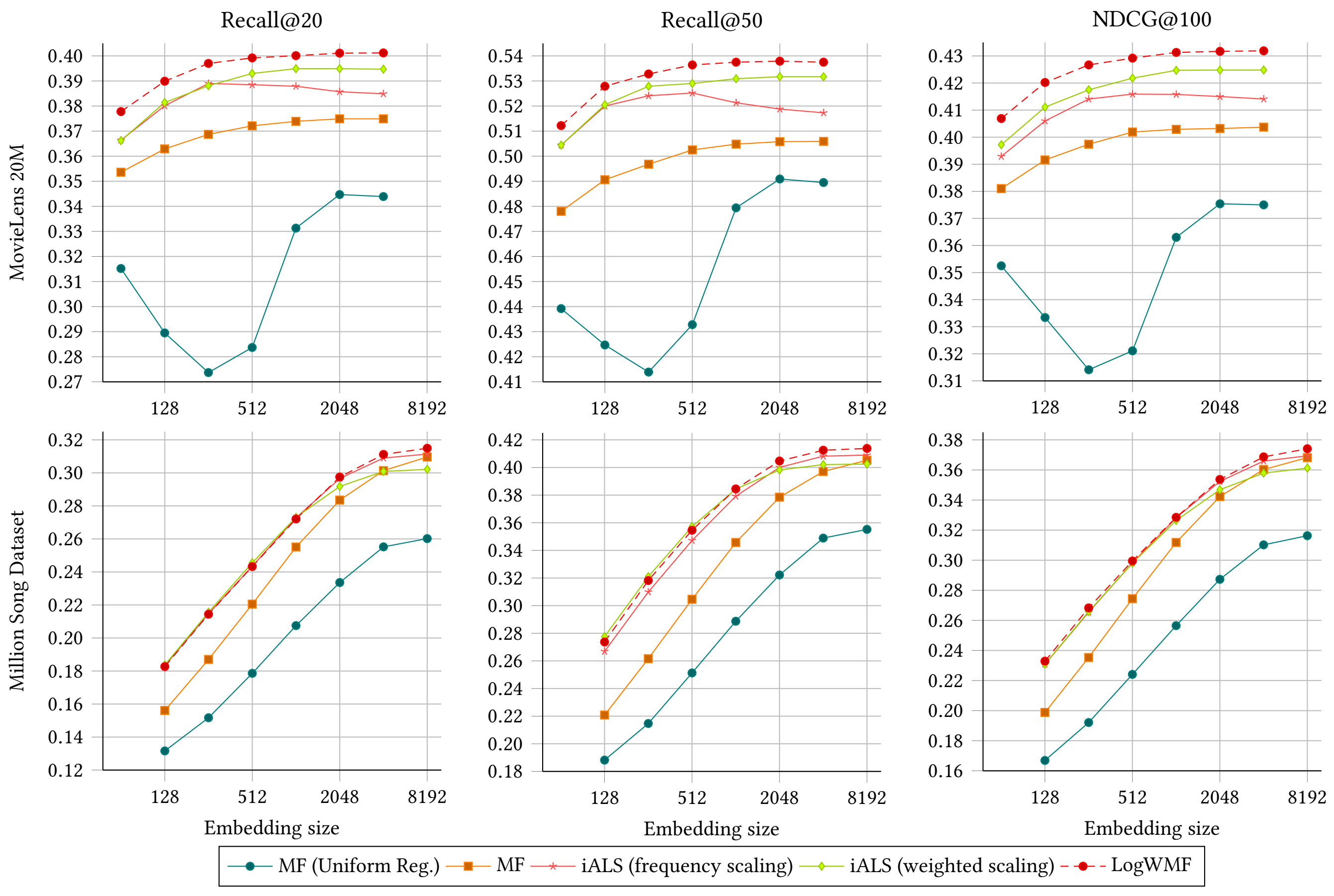

Figure 2: Results on the test set for different embedding dimensions.

item independently if the closed-form solution is used. By adapting the subvector optimization procedure of iALS as shown in Section 2.5, we found a scalable algorithm for WEASE and LogEASE.

Figure 3 shows the results of running EASE, WEASE and LogEASE on the validation set with different values for the unobserved weight α_0. By inspecting the results on the validation set, we can draw conclusions about which hyperparameters are selected by the grid search. First, we observe that the values for EASE and WEASE with $\alpha_0 = 1$ are very close for all metrics on both datasets. This validates our implementation of WEASE since it converges towards the same optimum as the closed form EASE. Secondly, we find that the full-rank WEASE model achieves its optimal performance with an unobserved weight of 1 on both datasets. As this corresponds to not downweighting the unobserved interactions at all, we can conclude that adding weights to the loss of EASE is actually detrimental for its performance. Considering that weighting plays such an important role in matrix factorization methods, it is a surprising result that the same does not hold for linear full-rank autoencoders. For LogEASE however, we find that the optimal value for α_0 is 0.01 on the ML20M dataset and 0.05 on the MSD dataset.

Based on these results we cannot formulate a conclusive explanation for this phenomenon yet. Further experimentation is needed to pinpoint the exact difference between matrix factorization and full-rank autoencoders that causes the difference in need for weighting. Our working theory is that either the full-rank nature, the diagonal constraint or the fixed user embeddings already fulfil the same role in the loss as the weighting does in matrix factorization, leading it to be obsolete and even detrimental when added.

Similarly, we found that item-specific regularization with frequency scaling and weighed scaling also did not improve results. Both for EASE and LogEASE the optimal results are found with $\nu = 0$ on both datasets. Notice that item-specific regularization in this context differs from the one used in [25]. Where we use a separate regularization value for each item embedding vector, the original paper scales each embedding dimension differently, i.e. the rows of B rather than its columns. The latter is easier to compute as it does not require a different inverse for each item.

4 RELATED WORK

In 2014, Johnson C. proposed a similar logistic matrix factorization model for implicit feedback data [10]. In their method called Logistic MF, unknowns are modelled as negatives and the model is trained with gradient descent and negative sampling to keep it scalable. The optimization procedure of LogWMF has two major advantages over Logistic MF. First, Newton's optimization method typically

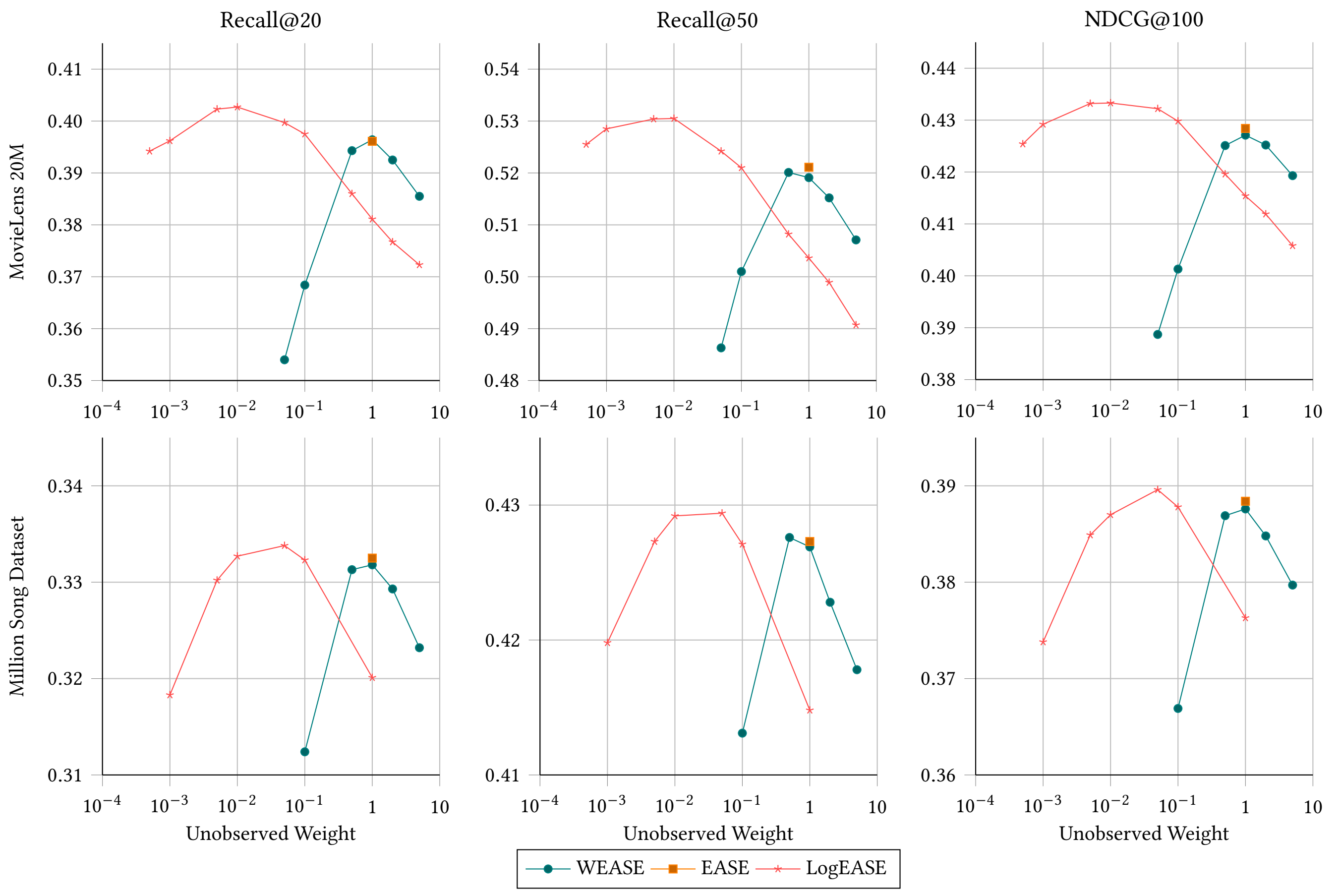

Figure 3: Results on the validation set for full-rank models with different unobvserved weights.

converges faster and does not require hyperparameters such as a learning rate to tune. Second, being able to take all negatives into account without needing to sample them also reduces the complexity of our method, although sampling could still be used to speed up the training process further.

Another difference is the need for bias terms in Logistic MF, which are are found to have little impact in iALS and LogWMF. Looking at the loss function that is optimized, the need for unregularized bias terms makes sense because the majority of the data is missing and the logistic function in Logistic MF requires the model weights to produce a negative number for those samples. However, the regularization on the user and item factors pulls them towards zero to prevent overfitting. Bias terms are not regularized and can therefore be used to make modelling the unknowns easier in Logistic MF. In iALS and LogWMF, the unknowns are modelled by a dot product of zero between factors, which is the same as the regularization target. The only use for bias terms would be to model the difference in popularity/activity between users and items, which is already captured by the magnitude of the embedding vectors.

A second related model is called Probabilistic MF by Mnih and Salakhutdinov [15]. They propose a probabilistic matrix factorization model for rating prediction, which is a different problem than the implicit feedback setting. The model is based on a Gaussian distribution over the ratings, where the mean is estimated by the dot product between user and item factors. Due to the difference in the tasks of rating prediction and top-k recommendation from implicit feedback, the approach does not directly carry over. However, they did also find a major improvement when using 'adaptive priors', similarly to using regularization scaling in our case.

The effectiveness of approximating the logistic function by the cumulative distribution function of a Gaussian distribution (or vice-versa) [4] was already demonstrated in previous work. For example, H. Steck [23] used this approximation to convert a single score to an approximated rank in a list. By decoupling the dependence of the ranking on the scores of all items, the method was able to approximately optimize ranking metrics such as NDCG in the loss.

5 CONCLUSIONS

In this paper we presented a probabilistic interpretation of implicit matrix factorization for recommendation. In this new formulation, unknowns are modelled as a product of positive and negative probabilities, which better fits their role in the data. Indeed, ultimately we are interested in recommending unseen items to a user, and hence assuming all of them are negative is too strict in a probabilistic setting. Furthermore, we derived an efficient optimization procedure for this new model, called LogWMF for Logistic Weighted Matrix Factorization. The key to its efficiency, is to approximate the probabilities of unknown interactions with a Gaussian. This allows for subvector optimization with Newton's method. Experiments on two benchmark datasets showed that LogWMF is able to outperform its linear counterpart iALS on the downstream recommendation task.

Additionally, a similar adaptation called LogEASE is derived based on the popular full-rank EASE method, and finally, a weighted version of EASE called WEASE is studied. LogEASE outperforms EASE on both datasets and requires tuning of the unobserved weight parameter. WEASE on the other hand is found to not benefit from the weighting at all, for which the exact reason remains an open question for future research.

For future work, we would like to investigate the following three directions: (1) Train with explicit negative interactions, such as skips in music or explicit dislikes in a upvote/downvote system. LogWMF takes a three-class classification approach that is well-suited for this type of data. (2) Investigate the effect of weighting in WEASE by relating it to the differences with matrix factorization. Most prominently are the full-rank, the diagonal constraint and the sparsity pattern of the fixed user embeddings. (3) Study and compare the learned embeddings of iALS and LogWMF to gain a better understanding of the differences in performance and where further improvements can be made.

ACKNOWLEDGMENTS

This work was supported by the Research Foundation — Flanders (FWO) [11E5921N to J. De Pauw].

REFERENCES

[1] Charu C Aggarwal et al. 2016. *Recommender systems*. Vol. 1. Springer.

[2] Immanuel Bayer, Xiangnan He, Bhargav Kanagal, and Steffen Rendle. 2017. A generic coordinate descent framework for learning from implicit feedback. In *Proceedings of the 26th international conference on world wide web*. 1341–1350.

[3] Thierry Bertin-Mahieux, Daniel PW Ellis, Brian Whitman, and Paul Lamere. 2011. The million song dataset. (2011).

[4] Christopher M Bishop. 1995. *Neural networks for pattern recognition*. Oxford university press.

[5] Christopher Burges, Robert Ragno, and Quoc Le. 2006. Learning to rank with nonsmooth cost functions. *Advances in neural information processing systems* 19 (2006).

[6] F Maxwell Harper and Joseph A Konstan. 2015. The movielens datasets: History and context. *Acm transactions on interactive intelligent systems (tiis)* 5, 4 (2015), 1–19.

[7] Balázs Hidasi and Ádám Tibor Czapp. 2023. The effect of third party implementations on reproducibility. In *Proceedings of the 17th ACM Conference on Recommender Systems*. 272–282.

[8] Yifan Hu, Yehuda Koren, and Chris Volinsky. 2008. Collaborative filtering for implicit feedback datasets. In *2008 Eighth IEEE international conference on data mining*. Ieee, 263–272.

[9] Ruoming Jin, Dong Li, Jing Gao, Zhi Liu, Li Chen, and Yang Zhou. 2021. Towards a better understanding of linear models for recommendation. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*. 776–785.

[10] Christopher C Johnson et al. 2014. Logistic matrix factorization for implicit feedback data. *Advances in Neural Information Processing Systems* 27, 78 (2014), 1–9.

[11] Daeryong Kim and Bongwon Suh. 2019. Enhancing VAEs for collaborative filtering: flexible priors & gating mechanisms. In *Proceedings of the 13th ACM conference on recommender systems*. 403–407.

[12] Dawen Liang, Rahul G Krishnan, Matthew D Hoffman, and Tony Jebara. 2018. Variational autoencoders for collaborative filtering. In *Proceedings of the 2018 world wide web conference*. 689–698.

[13] Sam Lobel, Chunyuan Li, Jianfeng Gao, and Lawrence Carin. 2019. Towards amortized ranking-critical training for collaborative filtering. *arXiv preprint arXiv:1906.04281* (2019).

[14] Peter McCullagh. 2019. *Generalized linear models*. Routledge.

[15] Andriy Mnih and Russ R Salakhutdinov. 2007. Probabilistic matrix factorization. *Advances in neural information processing systems* 20 (2007).

[16] Xia Ning and George Karypis. 2011. Slim: Sparse linear methods for top-n recommender systems. In *2011 IEEE 11th international conference on data mining*. IEEE, 497–506.

[17] Rong Pan, Yunhong Zhou, Bin Cao, Nathan N Liu, Rajan Lukose, Martin Scholz, and Qiang Yang. 2008. One-class collaborative filtering. In *2008 Eighth IEEE international conference on data mining*. IEEE, 502–511.

[18] István Pilászy, Dávid Zibriczky, and Domonkos Tikk. 2010. Fast als-based matrix factorization for explicit and implicit feedback datasets. In *Proceedings of the fourth ACM conference on Recommender systems*. 71–78.

[19] Steffen Rendle, Walid Krichene, Li Zhang, and Yehuda Koren. 2021. IALS++: Speeding up matrix factorization with subspace optimization. *arXiv preprint arXiv:2110.14044* (2021).

[20] Steffen Rendle, Walid Krichene, Li Zhang, and Yehuda Koren. 2022. Revisiting the performance of ials on item recommendation benchmarks. In *Proceedings of the 16th ACM Conference on Recommender Systems*. 427–435.

[21] Faisal Shehzad and Dietmar Jannach. 2023. Everyone'sa winner! on hyperparameter tuning of recommendation models. In *Proceedings of the 17th ACM Conference on Recommender Systems*. 652–657.

[22] Ilya Shenbin, Anton Alekseev, Elena Tutubalina, Valentin Malykh, and Sergey I Nikolenko. 2020. Recvae: A new variational autoencoder for top-n recommendations with implicit feedback. In *Proceedings of the 13th international conference on web search and data mining*. 528–536.

[23] Harald Steck. 2015. Gaussian ranking by matrix factorization. In *Proceedings of the 9th ACM Conference on Recommender Systems*. 115–122.

[24] Harald Steck. 2019. Embarrassingly shallow autoencoders for sparse data. In *The World Wide Web Conference*. 3251–3257.

[25] Harald Steck, Maria Dimakopoulou, Nickolai Riabov, and Tony Jebara. 2020. Admm slim: Sparse recommendations for many users. In *Proceedings of the 13th international conference on web search and data mining*. 555–563.

[26] Jason Weston, Samy Bengio, and Nicolas Usunier. 2011. Wsabie: Scaling up to large vocabulary image annotation. (2011).

[27] Yao Wu, Christopher DuBois, Alice X Zheng, and Martin Ester. 2016. Collaborative denoising auto-encoders for top-n recommender systems. In *Proceedings of the ninth ACM international conference on web search and data mining*. 153–162.