Taxonomy-informed latent factor models for implicit feedback

Andriy Mnih
Gatsby Computational Neuroscience Unit
University College London
amnih@gatsby.ucl.ac.uk

ABSTRACT

We describe an approach based on latent factor models to the Track 2 task of KDD Cup 2011, which required learning to discriminate between highly rated and unrated items from a large dataset of music ratings. We take the pairwise ranking route, training our models to rank the highly rated items above the unrated items which are sampled from the same distribution. Using the item relationship information from the provided taxonomy to constrain item representations results in improved predictive performance. Providing the model with features summarizing the user rating history as it relates to the item being ranked leads to further improvement.

Categories and Subject Descriptors

I.2.6 [**Machine Learning**]: Engineering applications

General Terms

Experimentation, Algorithms

Keywords

Collaborative filtering, implicit feedback, machine learning

1. INTRODUCTION

Collaborative filtering deals with inferring complete user preferences from large but incomplete collections of preference expressions [8]. Users can express their preferences using explicit feedback, such as item ratings, or implicit feedback, such as item views or purchases [5]. Explicit feedback is more reliable than implicit feedback but is typically available in much smaller quantities. This is a consequence of requiring an effort on the part of the users to explicitly express their preferences through special actions, such as rating items. Implicit feedback, on the other hand, is much easier to obtain in large quantities because it is produced as a byproduct of natural use of the system. This easy availability makes modelling implicit feedback an important task, which until recently has received surprisingly little attention.

What makes implicit feedback fundamentally different from explicit feedback (e.g. like/dislike) is that users provide only positive examples, that is just the items they are interested in [5]. In other words, though users can express their interest in an item by selecting it, they have no way to express their disinterest. Note that not having expressed interest in an item is not the same as having expressed disinterest. After all, the set of items a user has expressed interest in is almost certainly incomplete due to the user's budgetary and time constraints as well as incomplete knowledge of the inventory. As a result, the set of items not selected by a user can contain both interesting and uninteresting items. It is this uncertainty about the unobserved items that makes implicit feedback more difficult to model than its explicit counterpart.

The simplest way to deal with this uncertainty is to ignore it and use all items not selected by the user as negative examples. Approximating the resulting dense binary user-item matrix with a product of two low-rank matrices [5, 10, 9] is the most popular approach to modelling implicit feedback. Such models are typically trained using alternating least squares in time linear in the number of positive examples but cubic in the number of latent factors.

A recently introduced approach, called Bayesian Personalized Ranking (BPR), makes more realistic assumptions about the unobserved items [11]. Instead of assuming that the unobserved items are uninteresting, it only assumes that they are less interesting than the observed items. The model is trained to rank the positive items above the negative ones in positive-negative item pairs, with the negative items sampled from the uniform distribution. Both users and items are represented using latent factor vectors which are learned using stochastic gradient ascent, ensuring excellent scalability. In this paper we apply BPR to the Track 2 task of KDD Cup 2011, modifying the training algorithm to use a popularity-based distribution of negative examples and extending the model to take advantage of the item taxonomy.

1.1 The task

In this paper we describe our approach to the Track 2 task of KDD Cup 2011, which deals with music recommendation using a dataset of item ratings collected from the Yahoo! Music service. The dataset consists of ratings assigned to music items of four types (tracks, albums, artists, and genres) by about 249K users of the site. The dataset set contains about 64M ratings on a scale from 0 to 100 for over 296K items, about 224K of which are tracks. See [4] for a more detailed description of the data.

The Track 2 task is to learn to discriminate between the highly rated (ratings 80 and above) and the unrated tracks for each user. The test set contains six tracks per user, three of which are highly rated and three are unrated. Note that because this rated/unrated breakdown is known, it is possible to discriminate between items in the test set even with a model that ranks items instead of classifying them.

This task is most naturally viewed as an implicit feedback modelling problem because the goal is to learn to discrim-

inate between the observed positive examples (the highly rated tracks) and the unobserved negative examples (the unrated tracks). However, the Track 2 task differs from the classic implicit feedback problem in two important ways. First, models are evaluated at discriminating between three positive items and three negative items that come from the same distribution. This makes it impossible to use item popularity for discrimination between positive and negative examples, which is not the case in the classic setting where negative examples are effectively uniformly distributed. Second, in addition to the identities of the positive items, which are also available in the classic setting, we have access to the item taxonomy and item ratings. We will show how to take advantage of these extra sources of information to improve model performance.

2. BASIC LATENT-FACTOR MODEL

We represent both users and items using D-dimensional real-valued vectors, which will be learned automatically from the data. We will refer to entries in these vectors as *latent factors* to emphasize that they are learned. Though latent factors are sometimes called *features* in the collaborative filtering literature (e.g. in [12]), we will use the term *features* to refer only to the quantities precomputed using hand-designed rules and given to the learning algorithm as (fixed) inputs. For example, the number of items rated can be one of the features associated with a user. Though such hand-engineered features can sometimes be used to boost system performance, designing effective features is a difficult and time-consuming task, which is why our approach will rely mostly on learned representations.

2.1 The scoring function

Our approach to the problem is based on learning a scoring function for each user to capture their preferences. Since the task of interest is discrimination between positive and negative items, we would like to learn functions that assign higher scores to positive items than to negative ones. Once the scoring function for a user has been learned, we can rank any set of items by ordering them according to their scores.

We restrict our attention to scoring functions of the form

$$s_u(i) = U_u^\top V_i + b_i, \tag{1}$$

where U_u and V_i are the latent factor vectors for user u and item i respectively. b_i is the item-specific bias term that models the popularity of the item. We do not include a user-specific bias term because adding a constant to the function has no effect on the induced ranking. In spite of their simple parametric form, such scoring functions are very expressive because the factor vectors they operate on are learned from the data and can be as high-dimensional as necessary.

In order to learn scoring functions we need to introduce an objective function that links item scores to observations. Unfortunately, the classification error rate, which is the official Track 2 performance metric, is discontinuous which makes optimizing it difficult. We take the probabilistic approach instead and use the (penalized) log-likelihood as the objective function, which is much easier to optimize because it is smooth.

2.2 Classification and ranking objective functions

Our first model tries to classify items into positive and negative (for each user) by computing the probability that the item is positive based on its score. The probability of the item being positive is obtained from its score by applying the logistic function:

$$P(i \in \mathcal{P}_u) = \frac{1}{1 + \exp(-s_u(i))}, \tag{2}$$

where $\mathcal{P}_u$ is the set of positive items for user u. This model can be seen as a logistic regression classifier with the item factor vector serving as the input and the user factor vector as the weights.

While this model is appealing because of its simplicity, it is trying to solve a harder problem then necessary by estimating the probability of an item being positive. After all, in collaborative filtering we are typically interested in ranking items based on some criterion, e.g. potential interest to the user, as opposed to classifying them. Since ranking is in a sense an easier task than classification because there is no need to decide on a score threshold for separating the positive and the negative items, we also tried learning scoring functions by training a probabilistic ranking model. We take the pairwise ranking approach used by BPR [11] and model the probability that user u prefers item i to item j as the logistic function of the difference between the item scores:

$$P(i >_u j) = \frac{1}{1 + \exp(-(s_u(i) - s_u(j)))}. \tag{3}$$

The model is trained on positive item i / negative item j pairs. We describe how the negative items are generated in Sec. 3.3.

To make predictions with either model on the test set, we rank the items for each user according to their scores, and label the three top-scoring ones as positive and the remaining three as negative.

3. INCORPORATING TAXONOMY INFORMATION

One distinguishing feature of the dataset is that items come in several types: tracks, albums, artists, and genres. We are also given a taxonomy that links items of different types, which provides the album, the artist, and the list of genres for each track, as well as the artist and the list of genres for each album.

We used taxonomy information to parameterize track factor vectors in a more sophisticated way and to generate features that relate items to the user's rating history in a more direct manner than latent factors allow.

3.1 Taxonomy-based parameterization

Learning latent factor models that generalize well can be difficult because of the large number of free parameters that have to be estimated from the data. For example, a model from Sec. 2 with 100-dimensional latent vectors has over 54M parameters, which is of the same order as the number of positive examples in the training set. If we know that some of the parameters should take on similar values, we can improve generalization by incorporating this information into the model. We apply this principle to encourage tracks from the same album or by the same artist to have similar representation. We achieve this by defining new track latent

factor vectors in a hierarchical manner:

$$V_i^H = V_i + w_{al}^r V_{album(i)} + w_{ar}^r V_{artist(i)}, \quad (4)$$

where $album(i)/artist(i)$ refer to the album/artist for track i.[1] In other words, a track is now represented by a linear combination of its old unconstrained representation (V_i) and the representations of its album and artist. The contributions of the album and artist representations are modulated by w_{al}^r and w_{ar}^r, which are learned along with all other parameters. We also define new track biases in an analogous manner:

$$b_i^H = b_i + w_{al}^b b_{album(i)} + w_{ar}^b b_{artist(i)}, \quad (5)$$

and use V_i^H and b_i^H in place of V_i and b_i in Eq. 1.

This kind of hierarchical parameterization encourages the latent vectors (and biases) of tracks from the same album and/or by the same artist to have similar representations. It also encourages track representations to be similar to those of its album and artist. As a result, updating the latent vector V_i^H for a track will affect not only the track's representation but also the representations of its album and artist, as well as the representations of all other tracks sharing the album or the artist. Similarly, updating a representation for an album will also affect representations for all tracks off that album. Thus this kind of parameterization should allow effective pooling of information between all types of items, except for genres.[2]

3.2 Taxonomy-based features

There is another, more direct, way to use the item relationship information provided by the taxonomy. It involves identifying the items related to the current track (i.e. its album, artist, and tracks sharing the album or the artist) and then computing a set of features that summarize the user's rating history for these items. The resulting feature vector $f(u, i)$ is then used as an additional input to the scoring function:

$$s_u(i) = U_u^\top V_i + b_i + (W_u + W_i + W)^\top f(u, i). \quad (6)$$

Here W_u, W_i, and W are the user-specific, the item-specific, and the global feature weights respectively, which are learned jointly with all other model parameters.

Though this approach is conceptually simple, its effectiveness is heavily dependent on the choice of features. This sort of feature based approach has been quite successful in natural language processing [7] and computer vision [6]. Designing useful features however requires a reasonably good understanding of the problem and extensive experimentation.

We experimented with a number of features and found that while many of them reduced the training error, only a few of them helped on the test set. We also observed that some features help models with low-dimensional latent factor vectors considerably, but either hurt or have no effect on models with higher-dimensional factor vectors.

Our feature-based models used (some) of the following features:

[1] If the album or artist for a track is unknown we drop the corresponding term from Eq. 4.

[2] We also tried including a contribution from the genre representations in Eq. 4 but found that it did not improve model performance.

Table 1: The effect of the sampling distribution

Distribution of negative items	Validation ERR (%)
Uniform	9.274
Same as for positive	5.350

1. The rating given to the track's album by the user (divided by 100)
2. Is the album's rating 80 or higher?
3. The rating given to the track's artist by the user (divided by 100)
4. Is the artist's rating 80 or higher?
5. The fraction of the track's genres rated by the user (either directly or by rating an album or a track of that genre)
6. Has the user rated (directly or indirectly) any of the track's genres?
7. The fraction of user's genre ratings the track's genres account for
8. Has the user rated any other track from the album?
9. Has the user rated any other track by this artist?
10. The mean rating of the track's genres that have been rated by the user (divided by 100)

In general, we found that only the first four of these features improved model performance in all cases. As we show in Sec. 5, using too many features can easily lead to overfitting in models with a large number of latent factors.

3.3 Training

The models we described need access to both positive and negative examples for training. We used the user-item pairs from the training part of the dataset as positive examples. For each user/positive-item pair we generated a negative item by sampling from the empirical distribution of items in the training set. We did not ensure that the sampled negative item has not been rated by the user because we found that the required bookkeeping noticeably increased the running time without having much of an effect on the predictive performance.

Since the models are evaluated based on their ability to rank three positive items above three negative items, optimal performance is achieved when each of the positive items is assigned a score larger than the score of any of the three (randomly-sampled) negative items. We encourage our models to learn to rank the current positive item above the highest-scoring of three randomly chosen negative items by sampling three items and keeping the one with the highest score under the model as the negative item.

We trained all our models using stochastic gradient ascent. The algorithm iterated through the user/positive-item pairs in the training set in a random order, generating a negative item for each such pair before updating model parameters. Training a model that uses features and has 100D latent vectors takes about half a day on a several-year-old Xeon

Table 2: The effect of the training objective

Objective function	Validation ERR (%)
Classification (Eq. 2)	5.556
Ranking (Eq. 3)	5.350

Table 3: The effect of taxonomy-based parameterization and features

Features used	Taxonomy-based parameterization	Validation ERR (%)
—	No	6.017
—	Yes	5.350
Best 4 (1-4)	No	3.916
Best 4 (1-4)	Yes	3.708

machine. Training models without features is considerably faster.

Model parameters were regularized using L2 weight cost using the scheme explained in [13] as well as early stopping on the validation set.

4. IMPLEMENTATION DETAILS

Except when explicitly stated otherwise, all reported results were obtained by training on user-item pairs with ratings 50 and higher, with the contribution from pairs with ratings between 50 and 79 downweighted by a factor of 4. The empirical distribution used for generating negative items was based on pairs with ratings 50 and higher. Features described in Sec. 3.2 were computed based on pairs with ratings 10 and above, as we discovered that including pairs with ratings between 10 and 49 to compute features improved predictive accuracy.

We generated negative samples from the empirical distribution of items in the training set by picking a training case uniformly at random and taking the associated item as the sample. This is much more efficient than the naive approach that precomputes the empirical probabilities of the items and then generates samples using those probabilities in time linear in the number of items.

In order to compute features from Sec. 3.2 efficiently, we stored all the necessary information about user ratings in hash maps[3] before starting training. Hash maps allow us to compute features for a user-item pair in time that is effectively constant in the number of training set ratings for the user.

5. RESULTS

We created a validation set in order to monitor model performance during training as well as to allow quick model evaluation without using the KDD Cup website. For each user present in the test set, we randomly selected three highly rated tracks from the training set as positive examples and moved them to the validation set.[4] The negative examples were generated by sampling from the empirical distribution of tracks rated 80 or higher in the training set. The scores on the resulting validation set were highly correlated with the corresponding test scores, with the validation scores being about 0.45% higher. In other words, a model with a validation error rate (ERR) of 5.45% will have a test ERR of about 5%.

[3]We also tried using binary trees but found them slower, though more memory-efficient, than hash maps.

[4]Users with fewer than three such tracks were not included in the validation set.

Table 4: The interplay between the number of latent factors and the usefulness of features

Features used	Number of latent factors	Validation ERR (%)
All	0	7.186
Best 4 (1-4)	0	11.654
All	100	3.977
Best 4 (1-4)	100	3.708

Unless stated otherwise, our models used 100-dimensional factor vectors and the taxonomy-based parameterization from Sec. 3.1, but did not use features. The feature-based models were trained using the best four features (features 1-4) from Sec. 3.2. Early stopping on the validation set was performed to prevent overfitting. We report only the validation ERR (in percent) for these models because we had no access to the official test set.

First, we looked at the effects of the objective function and the distribution of the negative items on model performance. As can be seen from Table 2, the pairwise ranking approach outperforms the classification approach, though not drastically. The effect of sampling negative items from the empirical distribution of positive items instead of the uniform distribution (used by BPR) is much more dramatic. As shown in Table 1 it reduces the error rate by a factor of 1.7. As a result, we performed all the remaining experiments using the ranking objective with the negative items sampled from the empirical distribution.

We investigated the benefits of taxonomy-based parameterization and taxonomy-based features by training four models, one for each possible parameterization / features combination. The scores shown in Table 3 indicate that the features help considerably more than the parameterization. The two uses of taxonomy information appear to be complementary though, as using them both results in the best-performing model.

To explore the interplay between the features used and the latent vector dimensionality we trained models with different numbers of latent factors using only the best four features or all features. Table 4 shows the results for models with 100-dimensional latent vectors and models with no latent factors. Using more features seems to hurt models with many latent factors and help models without (or with few) latent factors, which suggests that the models with a lot of features and latent factors are beginning to overfit. We have also observed that models without features benefit from higher-dimensional latent vectors (e.g. 200D and higher) while the models with features do not, which seems to support the overfitting theory.

To determine the contribution of different types of feature weights, we trained three models, each of which used only one type of feature weights (global, user-specific, or item-specific). The results shown in Table 5 suggest that the user-specific feature weights do most of the work, though the item-specific and the global weights also help.

6. OUR BEST SUBMISSION

Our best submission combined predictions from the fol-

Table 5: The contribution of different feature weights

Feature weights used	Validation ERR (%)
All	3.708
Global only	4.051
Item only	4.003
User only	3.830

lowing models:

1. Ranker without features with 500 latent factors
2. Ranker without features with 1000 latent factors
3. Ranker without features with 2000 latent factors
4. Ranker with features 1-10 with 1000 latent factors
5. Ranker with features 1-6,10 with 100 latent factors
6. Ranker with features 1-5 with 100 latent factors
7. Classifier with features 1-5 with 100 latent factors
8. Classifier with features 1-5 with 500 latent factors

The following learning rates were used: 10^{-2} for user representations and item biases, 3×10^{-2} for item representations, 10^{-4} for artist/album contribution weights (w^r_{ar} / w^r_{al}), 10^{-2} for user/item feature weights, and 10^{-4} for global feature weights. The weight cost for user/item representations was 10^{-4} in models without features and 10^{-3} in models with features. The weight cost values for user/item-specific feature weights and global feature weights were 3×10^{-3} and 10^{-3} respectively.

Each model was trained using the above parameters until its validation set ERR started increasing. Then the model was finetuned by reducing all learning rates by a factor of 10 (except for the artist/album contribution weight learning rate, which was set to 0) and training until the validation ERR started increasing again. The finetuning process was repeated one more time for Models 1-3 after removing the observations with ratings lower than 80 from the training set. The feature weights for model 4 were learned before other model parameters and were kept fixed while learning the remaining parameters.

Model scores were combined linearly using the weights learned by minimizing the ranking loss (Eq. 3) on the validation set. The resulting ensemble achieved the score of 2.9337 on the leaderboard.

7. DISCUSSION

In this paper we have described several improvements to the Bayesian Personalized Ranking model [11] that make it perform much better on Track 2 of KDD Cup 2011. They include using a data-driven distribution to generate negative examples and taking advantage of the item taxonomy to better parameterize item representations. We also used the taxonomy to provide the model with features summarizing the user's rating history as it relates to a particular item.

There is a number of directions we did not have time to explore. Since BPR is based on a pairwise ranking approach [2] introduced in the "learning to rank" literature, it might be worthwhile to consider the more recent ranking methods developed in that field. LambdaRank [1] as well as the listwise ranking methods [14, 3] seem especially promising.

Using a linear function to map features to item score contributions is the simplest choice but probably not the best one. More powerful mappings such as neural networks might be able to make better use of the features.

8. ACKNOWLEDGMENTS

I would like to thank Yee Whye Teh for helpful discussions and feedback on a draft of this paper.

9. REFERENCES

[1] C. J. C. Burges, R. Ragno, and Q. V. Le. Learning to rank with nonsmooth cost functions. In *NIPS*, pages 193–200, 2006.

[2] C. J. C. Burges, T. Shaked, E. Renshaw, A. Lazier, M. Deeds, N. Hamilton, and G. N. Hullender. Learning to rank using gradient descent. In *ICML*, pages 89–96, 2005.

[3] Z. Cao, T. Qin, T.-Y. Liu, M.-F. Tsai, and H. Li. Learning to rank: from pairwise approach to listwise approach. In *ICML '07*, pages 129–136, 2007.

[4] G. Dror, N. Koenigstein, Y. Koren, and M. Weimer. The Yahoo! Music Dataset and KDD-Cup'11. In *KDD-Cup Workshop*, 2011.

[5] Y. Hu, Y. Koren, and C. Volinsky. Collaborative filtering for implicit feedback datasets. In *Proceedings of the 2008 Eighth IEEE International Conference on Data Mining*, pages 263–272, 2008.

[6] D. Lowe. Object recognition from local scale-invariant features. In *Computer Vision, 1999. The Proceedings of the Seventh IEEE International Conference on*, volume 2, pages 1150 –1157 vol.2, 1999.

[7] C. D. Manning and H. Schütze. *Foundations of statistical natural language processing*. MIT Press, Cambridge, MA, USA, 1999.

[8] B. Marlin. Collaborative filtering: A machine learning perspective. Master's thesis, University of Toronto, 2004.

[9] R. Pan and M. Scholz. Mind the gaps: weighting the unknown in large-scale one-class collaborative filtering. In *KDD*, pages 667–676, 2009.

[10] R. Pan, Y. Zhou, B. Cao, N. N. Liu, R. M. Lukose, M. Scholz, and Q. Yang. One-class collaborative filtering. In *ICDM*, pages 502–511, 2008.

[11] S. Rendle, C. Freudenthaler, Z. Gantner, and S.-T. Lars. BPR: Bayesian personalized ranking from implicit feedback. In *UAI '09*, pages 452–461, 2009.

[12] R. Salakhutdinov and A. Mnih. Probabilistic matrix factorization. In *Advances in Neural Information Processing Systems*, volume 20, 2008.

[13] R. Salakhutdinov and N. Srebro. Collaborative filtering in a non-uniform world: Learning with the weighted trace norm. In *NIPS*. 2010.

[14] F. Xia, T.-Y. Liu, J. Wang, W. Zhang, and H. Li. Listwise approach to learning to rank: theory and algorithm. In *ICML '08*, pages 1192–1199, 2008.