The Xbox Recommender System

Noam Koenigstein
Microsoft R&D
Herzliya, Israel

Nir Nice
Microsoft R&D
Herzliya, Israel

Ulrich Paquet
Microsoft Research
Cambridge, UK

Nir Schleyen
Microsoft R&D
Herzliya, Israel

ABSTRACT

A recent addition to Microsoft's Xbox Live Marketplace is a recommender system which allows users to explore both movies and games in a personalized context. The system largely relies on implicit feedback, and runs on a large scale, serving tens of millions of daily users. We describe the system design, and review the core recommendation algorithm.

Categories and Subject Descriptors

H.2.8 [**Database Applications**]: Data Mining

General Terms

Algorithms

Keywords

Recommender Systems, Collaborative Filtering, Xbox

1. INTRODUCTION

The distribution of items bought in a marketplace typically follow a power-law distribution, with popular items exponentially overwhelming less popular ones. As a result, many items in the tail of the distribution, which would have been suited to a particular user, are never exposed. There are a number of reasons for tailoring a user's experience in any marketplace, the most compelling of which are the need to increase user engagement, and the need to give exposure to tail items that would, under normal circumstances, be drowned by a few popular ones.

These can be addressed by a system that models and learns from user engagement. Recommender systems have been particularly effective at this task, to which the systems of MSN News, Amazon, Netflix, and the likes of others can testify. Each of these face unique challenges that are posed by the size and nature of the catalog, type of user feedback, rate of catalog change, scalability, and expected real-time system performance.

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

1.1 The Xbox dataset

Our dataset comprise of usage information from Xbox consoles. For every user we collect the interaction time with every item on our catalog (either a game or a movie). We have tens of millions of active users, almost all of which are playing Xbox games. Some of our users also enjoy our movies catalog and watch movies purchased using their Xbox consoles. We provide personalized recommendations for both domains using two identical recommendation engines as described in this paper.

The number of Xbox users increase on a daily basis. Similarly, we regularly update our games and movies catalog with the latest products available on the market. With our very large user base, we hardly encounter a "cold start" problem in the items domain, because within a few hours we already have sufficient information on every new item. Similarly, Xbox users generally purchase some games together with their consoles, so we rarely encounter "cold users" in the games domain. Only in the movies domain, we encounter the cold start problem for users that do not watch movies on their consoles. We still hope to address this problem with cross domain learning, however this is out of the scope of this paper. In this paper, we deal only with users that have at least one item in their usage history.

Unlike the well-studied Netflix prize problem, where the user's movie preferences are explicitly given by a five-star rating scale, our data largely comprises of implicit signals, like that of purchasing a game or watching a movie on the Xbox console. The absence of a negative signal poses a challenge to conventional solutions based on the factorization of a "ratings" matrix into two low rank matrices with one representing latent user features, and the other representing latent item features. We therefore augmented our implicit dataset with randomly generated negative signals to prevent trivial solutions as explained in Section 2.

This paper.

In this paper we wish to bring a complete overview of a real world large scale recommender system. We draw a distinction between the tasks of modeling (how hidden parameters are assumed to generate observed data), inference (finding the hidden parameters given the data), and utility (using the model's predictions in order to optimize a reward function). Section 2 is devoted to an overview of the system's architecture and some preprocessing steps to prepare the data. Section 3 explains the modeling and parameter inference, which happens *offline*, and gives an example of an *online* utility function. Finally, in Section 4 we evaluate the model and summarize in Section 5.

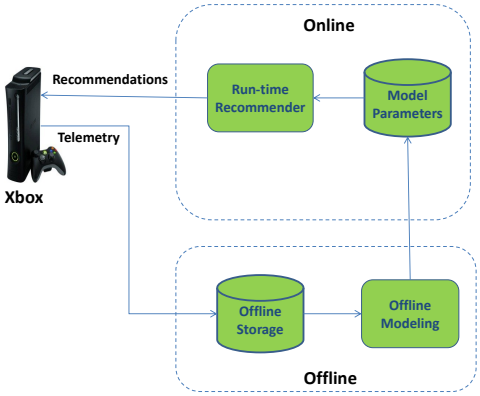

Figure 1: Xbox recommender system architecture

2. SYSTEM DESIGN AND PREPROCESSING

The architecture of the Xbox recommender system consists of an *offline* module and an *online* module (see figure 1). Feedback from the the Xbox consoles ("telemetry") is constantly sent back into the the *offline* module. In the *offline storage* we store information such as users' play time and movies watched by users. This information is then processed into an ownership dataset to be used by the *offline modeling* component for learning the model parameters.

In general, the *offline modeling* component treats the ownership information as an indication that the user liked the item. When processing the ownership information we employ some filters to remove games that were almost never played or movies that were not fully watched, but these are rare events that hardly effect the final recommendations. The ownership dataset comprises of implicitly generated "positive" only signals. Furthermore, it is hard to define credible ordering on those signals (e.g., the user liked a better than b), thus ordering algorithms such as [4, 3] are not useful. We therefore resolve to randomly generating negative signals as we describe below.

On average, an Xbox user has about 18 games and watched about 7 movies. Therefore, we can randomly pick a small number of items from our large catalog and assume that the reason the user do not own them is because she doesn't like them. For every user we pick the same number of "negative" items as the number of items she liked ("positive" items). By doing so, we cancel out the user bias as explained in Section 3. We draw the "negative" items with probabilities proportional to their popularity (training-set frequencies). This approach was highly popular in many of the solutions submitted to the second track in the KDD-Cup'11 [2] competition. Sampling according to popularity penalize popular items, which based on subjective internal evaluations found to increase user satisfaction.

The binary *like-dislike* matrix is then passed on to the *offline modeling*. A detailed description of the inference algorithm implemented in the *offline modeling* component is given in in 3.1.

Every few hours, the learned parameters are uploaded from the *offline modeling* component into the *model parameters* database. The *run-time recommender* in the *online* module is a real-time service that utilize the learned parameters in order to generate user specific recommendations. Upon a console-query for personalized recommendations, the *run-time recommender* chooses items according to the utility algorithm described in Section 3.2. These recommendations are then sent back to the console and presented to the user under the title "Picks For Me".

3. PREDICTION MODEL

The interaction between users and items lends itself to bilinear models. In them, each user m is represented by $\mathbf{u}_m \in \mathbb{R}^K$, each item n is represented by a similar vector $\mathbf{v}_n$, and the magnitude of their inner product $\mathbf{u}_m^T \mathbf{v}_n$ denotes the user's affinity to the item (personalization). The parameters $\mathbf{U} = \{\mathbf{u}_m\}_{m=1}^M$ and $\mathbf{V} = \{\mathbf{v}_n\}_{n=1}^N$ are unobserved, and should be inferred from data. This core set-up has been widely used in the Netflix prize competition, and we've adopted the Matchbox library of [5] for parameter inference.

The bilinear model has the property that items which often co-occur for a given user, have similar $\mathbf{v}$-vectors. Figure 2 shows $\mathbf{V}$ embedded in $\mathbb{R}^2$ for a number of games. In it, the similarity between vectors of sports games, for example, is clearly visible.

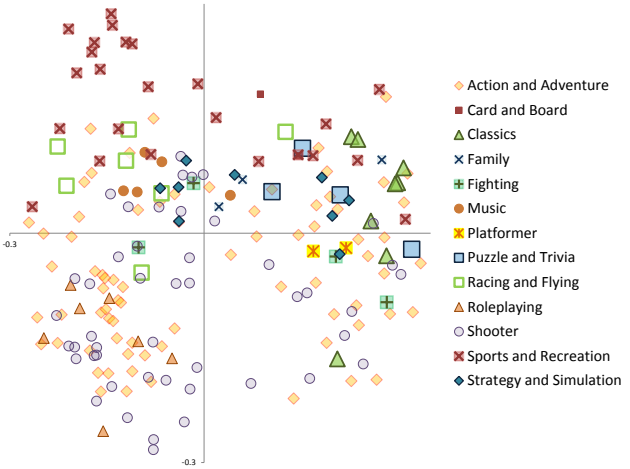

Figure 2: Game feature vectors embedded in $\mathbb{R}^2$, tagged by genre.

In practice some items are more popular than others, for which the personalization term is offset with a bias b_n for each item. We could equally add user-specific biases, but omit it in order to regress a users usage list which has the same number of positive and negative items. The probability that user m is going to like (or dislike) item n is

$$p(l_{mn}|\mathbf{u}_m, \mathbf{v}_n, b_n) = \Phi\left(l_{mn}(\mathbf{u}_m^T \mathbf{v}_n + b_n)\right) \quad (1)$$

where $l_{mn} = 1$ for the user liking and -1 for the user disliking the item. $\Phi(z) = \int_{-\infty}^{z} \mathcal{N}(x; 0, 1)\mathrm{d}x$ is the Gaussian cumulative density function, and acts as a link function that maps its argument to a value in the $(0, 1)$ interval.

Our working assumption is that there is sufficient usage data to infer $\mathbf{v}_n$. Alternatively, we can further regress against user or item-specific meta-data, or learn a K-dimensional representation for each meta-data atom if required [5].

Given the data $\mathcal{D} \doteq \{l_{mn}\}$ and parameters $\theta \doteq \{\mathbf{U}, \mathbf{V}, \mathbf{b}\}$,

the likelihood for observing the data is

$$p(\mathcal{D}|\theta) = \prod_{(m,n)} p(l_{mn}|\mathbf{u}_m, \mathbf{v}_n, b_n) \ .$$

Our interest lies in how well the data supports the parameters, i.e. the posterior probability of θ given the data. This is given by Bayes' theorem,

$$p(\theta|\mathcal{D}) = \frac{p(\mathcal{D}|\theta)p(\theta)}{p(\mathcal{D})} \ , \tag{2}$$

which incorporates a prior distribution on the model parameters. In our application, $p(\theta) \doteq p(\mathbf{U})p(\mathbf{V})p(\mathbf{b})$ factorizes as Gaussian distributions over the features and biases.

Bayesian inference comes into its own right when the posterior density $p(\theta|\mathcal{D})$ is used for predictions. Instead of using a single parameter estimate $\hat{\theta}$, which needs to be carefully regularized to avoid overfitting when data is sparse [1], we rather average the likelihood function over all parameters that plausibly explain the data. In other words,

$$p(l_{ij} = 1|\mathcal{D}) = \int p(l_{ij} = 1|\theta)\, p(\theta|\mathcal{D})\, \mathrm{d}\theta \tag{3}$$

gives the predictive distribution that user i is going to like item j.

Both the posterior and predictive distributions, given in (2) and (3), require the computation of analytically intractable integrals. The integrals can be evaluated stochastically through Monte Carlo methods, or approximated deterministically through a relaxation into an optimization problem.

3.1 Parameter inference

We approximate the posterior distribution with a one that factorizes with

$$p(\theta|\mathcal{D}) \approx q(\theta) = \prod_m q(\mathbf{u}_m) \prod_n q(\mathbf{v}_n)\, q(b_n) \ .$$

The choice is largely driven by practical convenience, as $q(\theta)$ is "factorized enough" to give rise to computationally tractable optimization problem and algorithm, and it allows us to tractably compute the (approximate) predictive distribution,

$$p(l_{ij}|\mathcal{D}) \approx \int p(l_{ij}|\mathbf{u}_i, \mathbf{v}_j, b_j)\, q(\mathbf{u}_i)\, q(\mathbf{v}_j)\, q(b_j)\, \mathrm{d}\{\mathbf{u}_i, \mathbf{v}_j, b_j\} \ , \tag{4}$$

where we've replaced $p(\theta|\mathcal{D})$ with $q(\theta)$ in (3). The factorization also ensures that only the matching user and item's approximations play a role, and other random variables are independently averaged away in (4).

We learn $q(\mathbf{v}_n)$ for each item; this is chosen as a K-dimensional factorizing Gaussian distribution $q(\mathbf{v}_n) = \prod_k q(v_{nk})$. As a result, we not only have a mean estimate of each item's (and user's) feature vector, but also encode uncertainty about its location through the variances of $q(\mathbf{v}_n)$. The variance of $q(\mathbf{v}_n)$ is typically small for very popular items, but larger for items that are less frequently played, watched, or bought. As a consequence, popular games (with more "well determined" parameters) will have a stronger effect on determining a user's feature vector, than games for which the model is less certain.

The approximation $q(\theta)$ is found by an algorithm called "Expectation Propagation", which has its roots in old and well-studied approximations[1] in statistical physics; the reader is referred to [5] for further algorithmic details.

[1]Expectation Propagation solves for the saddle point of a Bethe free energy with weak consistency constraints.

3.2 Retrieval of Recommendation

After training the model and learning parameter approximations for $\{\mathbf{U}, \mathbf{V}, \mathbf{b}\}$, we are free to choose a utility to optimize. As an example, the default task of retrieving recommendation for a specific user m can be formulated as follows. Let $\mathcal{Q} \doteq \{q(\mathbf{v}_n),\, q(b_n)\}$ be the set of all item densities, over which we maximize

$$\arg\max_{q_n \in \mathcal{Q}} \int p(l_{mn}|\mathbf{u}_m, \mathbf{v}_n, b_n)\, q(\mathbf{u}_m)\, q(\mathbf{v}_n)\, q(b_n)\, \mathrm{d}\{\mathbf{u}_m, \mathbf{v}_n, b_n\} \ ,$$

to find the items that a user will like with largest predictive probability.

In practice the above expression is analytic if either of the feature distributions is a point mass, or can otherwise be approximated with an analytic function as explained in [5].

4. RESULTS

We evaluate our system using a classification task inspired by the KDD-Cup'11 [2] competition. In Section 4.2 we also give a more traditional mean rank metric.

4.1 Recommendations as a classification task

Results here are based on 10,000 Xbox users with at least 2 games and at most 50 games. Here, we ignored Xbox consoles with more than 50 games because these may belong to larger organizations rather than individuals. Since traditional evaluation (e.g., root mean squared error) do not generalize well to the case of implicit data, we choose a rather new evaluation metric that was recently introduced in the second track of the KDD-Cup'11 [2] competition: For each test user we kept one item in the test set, and trained on the rest of the items. After training, we randomly picked another item for every user which is not present in the training or the test items of that user. The goal is to differentiate the item that the user owned from the item that was randomly picked. The randomly picked items were chosen in two methods: uniformly from the items set, and non-uniformly with probabilities proportional to each item popularity (similar to the process we use to generate implicit negative signals in Section 2). Namely, we define a classification task for every user and measure the overall precision on all users:

$$Precision = \frac{1}{|M|} \sum_{(m,n)\in\mathbf{T}} \mathbf{1}\left[p\left(l_{mn} = 1|\mathcal{D}\right) > p\left(l_{mk} = 1|\mathcal{D}\right)\right], \tag{5}$$

where $\mathbf{T}$ is the test-set, $p\left(l_{mn} = 1|\mathcal{D}\right)$ is the probability that user m likes item n (3), $p\left(l_{mk} = 1|\mathcal{D}\right)$ is the probability for user m to like the randomly sampled item k, and $\mathbf{1}\left(\cdot\right)$ is an indicator function. Note that since each user has just one item in the test-set, there is no real meaning to the recall measurement in this setting.

We had several objectives in choosing this evaluation task: First, it is closely related to the methodology we use in training the model (Section 2). More importantly, we believe this evaluation better relate to real life scenarios: As explained in [2], the proposed metric is related to the common recommendation task of predicting the items that the user is likely to own (rather than predicting a rating value to items the user already owns). In fact, this metric extends the evaluation to the truly missing entries and measures the true generalization power of the algorithm. Finally, there is an advantage to using this metric with negative examples drawn according to their popularity (non-uniformly), because it discourages known trivial solutions where most popular items are always suggested regardless of the user taste. In this way we focus

Domain	Uniform		Non-Uniform	
	Baseline	Xbox	Baseline	Xbox
Games	84.34%	89.21%	60.21%	75.56%
Movies	65.08%	79.27%	50.88%	69.93%

Table 1: Evaluation results for Xbox personalization recommender for movies and games. We present precision results against a baseline that always chooses the more popular item. We used both uniformly sampled items (over the entire items set) and non-uniformly sampling according to items popularity.

our evaluation on the true personalization power of the algorithm, rather than learning biases.

We evaluate this classification task against a simple baseline that always predicts that the user prefers the more popular item. Table 1 summarize the results. We see that the baseline algorithm performs relatively well when the negative items are uniformly distributed (84.3% and 65.1% in the games and movies domains respectively). However, these results are attributed only to learning popularity of items (biases), without any personalization. When negative signals are drawn according to their popularity, we better observe the precision gained by our personalization recommender over the baseline: an improvement of about 15% and 19% in the games and movies domains respectively. The overall results of the games recommender are better than those of the movies recommender. This is the result of the higher sparsity in the movies dataset (many more items, much less implicit ratings).

Note that in general, if we can sample negative items exactly according to their popularity, we expect that the base precision on the non-uniform test-set will be about 50% (as it is in movies). However, in the games domain some items are so popular that it is impossible to generate negative examples in exactly the same distribution as the positive items[2]. Because we do not assign negative items to users that also has those items in their positive items list, the resulting distribution is still somewhat biased towards the popular items and the baseline algorithm still achieves 60.2% precision on this task.

4.2 Mean Rank

We also present a more "traditional" evaluation using the *mean rank* metric. As before, for each user we keep one game in a test-set, and use the rest of the items to train the model. We then rank the item in the test set against 8 thousand other games from our catalog according to $p(l_{mk} = 1|\mathcal{D})$ - the probability that the user m will like an item k. We measured the mean rank of the test item for every user in the test set.

Figure 3 presents the mean rank (on a log scale) vs. the number of items in users history. Here we used 100,000 users (here we also included users with history longer than 50 items). An interesting observation is that the mean rank values increase with the users history length. Most of our users seem to own several very popular games. As users buy more games, they are more likely to own long tail items that are harder to model, which explains the line trend in Figure 3.

[2]Because the most popular Xbox games are owned by most of our users.

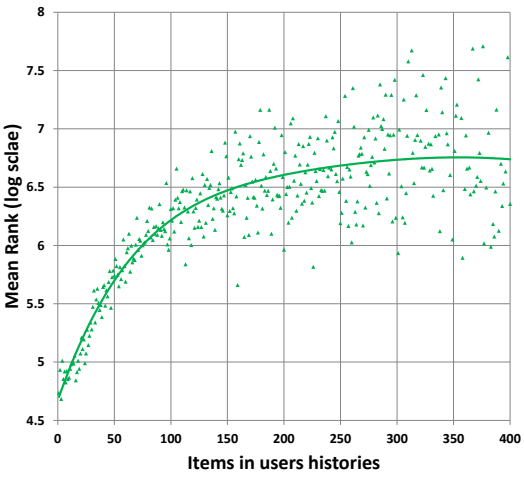

Figure 3: Mean-rank values vs. items in users history.

5. SUMMARY

A recent addition to Microsoft's Xbox Live Marketplace is a recommender system that allows users to explore personalized content tailored individually to their taste. This paper gives a complete overview of Xbox's recommender system in terms of system design, architecture, modeling, inference and utilization. We hope this example of real world large scale recommender system will give insights that would benefit the RecSys community academia and industry alike.

6. ACKNOWLEDGMENTS

The Xbox recommendation system was made possible through the hard work of everyone in the Recommendation Group in Microsoft Entertainment Services. The Matchbox library [5] was developed by the Machine Learning Group in Microsoft Research, Cambridge, UK.

7. REFERENCES

[1] G. Dror, N. Koenigstein, and Y. Koren. Yahoo! music recommendations: Modeling music ratings with temporal dynamics and item taxonomy. In *Proc. 5th ACM Conference on Recommender Systems (RecSys'11)*, 2011.

[2] G. Dror, N. Koenigstein, Y. Koren, and M. Weimer. The Yahoo! music dataset and KDD Cup '11. *Journal Of Machine Learning Research*, 17:1–12, 2011.

[3] U. Paquet, B. Thomson, and O. Winther. A hierarchical model for ordinal matrix factorization. *Statistics and Computing*, 21, 2011.

[4] S. Rendle, C. Freudenthaler, Z. Gantner, and L. Schmidt-Thieme. Bpr: Bayesian personalized ranking from implicit feedback. In *UAI '09: Proceedings of the 25th Conference on Uncertainty in Artificial Intelligence*, 2009.

[5] D. H. Stern, R. Herbrich, and T. Graepel. Matchbox: large scale online Bayesian recommendations. In *WWW*, pages 111–120, 2009.