Context-unaware Collaborative Filtering

CS 281 Final Project

Kyle Sargent Ştefan Spătaru Denis Turcu

Abstract

The context-unaware collaborative filtering problem is to extrapolate hypothetical ratings of unobserved user−item pairs from observed pairs, without any additional user or item information. Context-unaware collaborative filtering essentially reduces to completing a partially filled set of tabular data using only the available data already in the table. In this project, we study two state-of-the-art models for context-unaware collaborative filtering, neural collaborative filtering (NCF, see [**?**]) and Restricted Boltzmann Machines (RBM). We implement and test them on various datasets and analyze their performance through the lens of their differing probabilistic assumptions and structure.

1. Introduction

Collaborative filtering problems are highly important for a variety of applications, most of which deal with better user experience. This can reflect in saving the users' time by pointing them to items that best fit their preferences when they need suggestions. In many of these applications the available data includes users' past behavior and characteristics outside the scope of the specific problem, for example age, geographical location, friends list etc. However, we are interested in models that do not interfere with users' privacy, making predictions only based on the previous interactions with the desired item type. This led us to studying and evaluating models that specifically deal with problem: complete a sparsely populated set of tabular data.

At a high level, the most sophisticated current methods work by first constructing concrete features for users and/or items, and then building discriminative or generative models on top of those features. Our main goal is to study, implement, test and, probably most important, to compare such models on different datasets.

This paper starts by introducing the relevant research that has been done on this topic so far, and further clarifies the setup of the problem. Immediately following this, we dive into the theoretical support of these models and recall the appropriate and necessary mathematical notation needed to understand the assumptions of the models. We will discuss an important change that we believe is important in order to deal with discrete outputs.

Following the theoretical introduction and assumptions of the models, we discuss the inference process and what we can do to best predict on our data. We then discuss how the available datasets are suitable for this problem, as well as what we could extract and use from these. We finally present the experimental results, which, although not among the best on an absolute scale due to neglecting significant available information (i.e. users and items characteristics and direct relationships), confirm the developed theoretical expectations.

At each step in our process, we keep in mind that we want to compare the models from all possible directions. We consider theoretical complexity of the algorithms and associated inference problems, as well as convergence speed in terms of epoch number. Experimental results are another focus of the comparison since we are interested in both the probabilistic model and the predictive power on actual available datasets.

We conclude the paper with a discussion of the obtained results and interpretation based on the theoretical development. We also consider open questions on this topic, as well as possible future extensions of this work.

2. Background

Collaborative filtering problems have driven innovation in algorithms for learning from massive datasets. [6] and [7] respectively presented the Probabilistic Matrix Factorization (PMF) algorithm and applied the Restricted Boltzmann Machine to collaborative filtering. As [7] notes, effective methods for learning from large collaborative filtering datasets have not existed until relatively recently.

Context-unaware collaborative filtering implies the availability of (user, item) pairs with associated ratings and without any other data. As such, collaborative filtering datasets are at once immensely rich (possessing tens of millions of (user, item) pairs) and immensely sparse (possessing very few pairs associated to any specific user or item).

Due to this inherent sparseness, a context-unaware collaborative filtering problem can at first glance seem intractable. One common approach is to simply create user and item features and express (user, item) ratings as interactions of these features. Then choosing a loss on ratings allows learning on those features, in turn guaranteeing the feature interactions will extend to reasonable ratings on new (user, item) pairs. Both PMF and Neural Collaborative Filtering (NCF) fit this mold.

RBM collaborative filtering, by contrast, infers user features and assigns weights to (user, item) pairs as edges of a bipartite graph, then generating a distribution over ratings conditional on the features of each user.

3. Models

3.1. General framework

The mathematical framework we will use for our models assumes a set of M users, a set of N items, and a sparse matrix $r_{i,j}$ of ratings, where i corresponds to user i and j corresponds to item j. In all our models $r_{i,j} \in \{0, 1\}$. For each user, we want to predict, in order, the set of l items that are most likely to be rated 1. For comparison, we will use two models that were known to work well in general recommender systems settings. We present their initial version and our adaptations to meet our probabilistic needs.

As mentioned, one of the most effective methods in context-unaware collaborative filtering is probabilistic matrix factorization [6]. We will use it as a baseline for the comparison of our models.

3.2. Neural collaborative filtering(NCF)

3.2.1. Xiangan et al. model

[6] developed a new model that aims to exploit the expressiveness in the probabilistic matrix factorization model together with the power to detect non-linear patterns in the given data by a deep neural network. The algorithm is still based on a set of latent feature vectors $p_i, q_j \in \mathbb{R}^k$ and a function f. The algorithm aims to maximize likelihood assuming $p(r_{i,j}|p_i, q_j) \sim \mathcal{N}(f(p_i, q_j), \sigma)$, where f is a function learned by a neural network with $ReLU$ actiavation. Moreover, the algorithm assumes independence of the above distributions, conditional on all the p_i and q_j. This is a directed graphical model, represented in Figure 1.

To predict a list of the best items for a user i from all the ones available, one can consider the items j for which $f(p_i, q_j)$ is the highest, in order of these values.

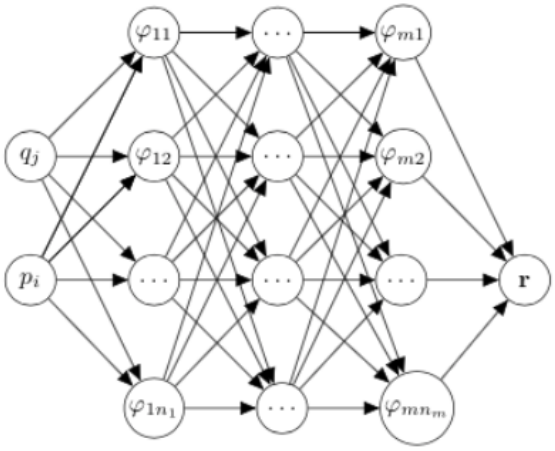

Figure 1. Graphical model of NCF

3.2.2. A discrete neural collaborative filter

There is a fundamental problem with the neural collaborative filtering model, which is that the conditional distribution of r given u and v is not discrete. For this reason, a discrete distribution would be more appropriate. The new model is trying to find the same hidden features p_i and q_j, but has a different probabilistic assumption:

$$p(r_{i,j} = 1) \sim Bern(p),$$

where $p = f(p_i, q_j)$.

Now, to make predictions, remark that $p(r_{i,j} = 1|U, V) = f(p_i, q_j)$, so for an user i, the algorithm will choose the r's with the highest value for v_i, in order. Moreover, we require this new neural network to have a softmax layer at the end to get a probability distribution instead of a score prediction.

The main reasoning behind using a model like 3.2.1 was its ability to differentiate through the loss between a rating that is slightly inaccurate from one that has a higher error, given that there is a natural ordering of the ratings. While this is generally true for this type of problems, in the problem of two classes this advantage is not useful, so we prefer to use a better suited Bernoulli model.

Remark that the set of parameters that needs training is the set of user latent features p, the set of item latent features q, and the weights w of the neural network. In total, we get $K(M + N) + nn$ parameters to train for, where nn is the number of parameters of the neural network.

3.3. Restricted Boltzmann machines(RBM)

Unlike the NCF model, an RBM is an undirected graphical model, represented by Figure 2. It is, moreover, a generative model. The model still assumes the existence of

user latent features, without assuming the existence of item latent features. The model creates a unit for every user, where latent features of every user (which we will denote by $p_1, ..., p_K$) are connected to the items the user has rated. In the following, we will present the model from [7]. We let m be the number of items rated by a particular user. We number them from 1 to m for ease of notation. The model assumes the following makes assumptions:

$$p(r_i = k|h) = \sigma\left(b_i + \sum_{l=1}^{m}\sum_{j=1}^{K} W_{lj}p_j\right),$$

$$p(h_j = 1|V) = \sigma\left(b^{(j)} + \sum_{i=1}^{m}\sum_{j=1}^{K} W_{ij}r_i\right),$$

where σ is the sigmoid function, and W_{lj}^k are shared weights between latent features and items.

One can see that the above formulation is equivalent with the joint model:

$$p(V, h) \sim \exp\left(\sum_{i=1}^{m}\sum_{j=1}^{K} W_{ij}h_j v_i + \sum_{i=1}^{m} b_i v_i + \sum_{j=1}^{K} h_j b^{(j)}\right)$$

For ease of notation we will let:

$$E(v, h) = \sum_{i=1}^{m}\sum_{j=1}^{K} W_{ij}h_j v_i + \sum_{i=1}^{m} b_i v_i + \sum_{j=1}^{K} h_j b^{(j)}.$$

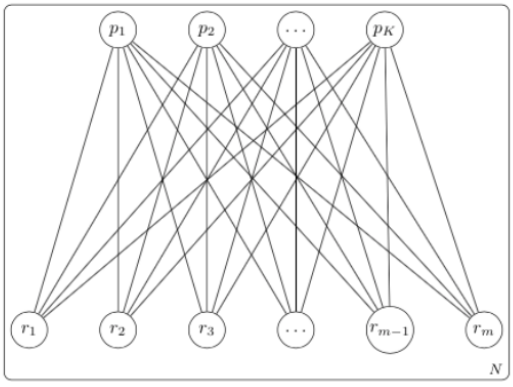

Figure 2. Graphical model of RBM

3.4. Comparison of NCF and RBM

From a theoretical point of view, the two models exhibit slightly different assumptions. Both models assume conditional independence of the ratings given the latent features. From d-separation, and the graphical models shown in Figure 1, we can infer that in NCF the latent parameters are not independent given the ratings, although the p_i's are independent conditional on the ratings and the item latent features. On the other hand, in a Restricted Boltzmann machine, the hidden features p are independent given the ratings and the weights of the model.

As mentioned in the background section, one of the main issues in recommender systems problems is that the matrices are very sparse, and there is very little knowledge of users' preferences. Given that the NCF models the probability $p(r|p, q)$, we expect it to be less able to make accurate predictions on users for whom the system has very little information, but more accurate prediction on users that rated a lot of items. On the other hand, the RBM model gives an interpretation of the joint distribution $p(r, p)$, and so the model will overfit less in training because of the lack of information of certain users.

One of the principal advantages of RBM over NCF is that RBM does not assume the independence of the ratings of a certain user on a set of movies the user hasn't rated. As a consequence, the RBM might be better suited as a model for predicting a list of the best l ratings, since a generative model is better able to model the interaction between different movies on the predicted list of best l movies. As such, being able to predict the list $i_1, ..., i_l$ such that $p(v_{i_1} = ... = v_{i_l} = 1|V)$ seems a better modeling assumption, since it doesn't assume independence. This turns out to not be a great advantage, since we used an approximation method which will be further explained in section 4.

On the other hand, the NCF model has an advantage as a discriminative model, given that it directly predicts $p(r|p_i, q_j)$, while the RBM fits the model for $p(r, p)$. Since the MAP is a conditional metric, the NCF models have a slight advantage since they are trained for the particular task.

4. Inference (or Training)

4.1. Neural collaborative filtering

4.1.1. Xiangan et al

The standard neural collaborative filtering would try to maximize the maximum likelihood estimation given its probabilistic assumptions. Thus, this is equivalent to minimizing

$$\sum_{e \in T}(r_{e_i,e_j} - f(p_{e_i}, q_{e_j}))^2,$$

where T is the training set, and e_i and e_j are the user and item labels of data-point e respectively.

Thus this model minimizes the mean squared error loss of the predictions. To find the optimal parameters p_i, q_j, as

well as the weights between layers in the neural net that gives f, stochastic gradient descent can be used to find the optimal parameters.

4.1.2. The discrete version

This will follow the same approach as above. However, maximizing the MLE is not equivalent to minimizing the function

$$\sum_{e \in T} r_{e_i,e_j} \log p + (1 - r_{e_i,e_j}) \log(1-p),$$

where as before $p = f(p_{e_i}, q_{e_j})$.

Remark that this loss function corresponds to the cross-entropy loss.

The different functional forms strengthen our ideas from section 3, since the cross-entropy loss is more suitable for classification problems than mean squared error loss. Gradient updates are analogous to the ones in subsubsection 4.2.1.

The gradients of the weights in the neural network can be found using back-propagation, while the gradient updates for the latent feature vectors p_i and q_j, after seeing one example (i, j) will be

$$p_i \leftarrow p_i - \eta \cdot \frac{\partial f}{\partial p_i} \left(\frac{r_{i,j}}{f(p_i, q_j)} - \frac{1 - r_{i,j}}{1 - f(p_i, q_j)} \right)$$

$$q_j \leftarrow q_j - \eta \cdot \frac{\partial f}{\partial q_j} \left(\frac{r_{i,j}}{f(p_i, q_j)} - \frac{1 - r_{i,j}}{1 - f(p_i, q_j)} \right)$$

Learning is summarized in algorithm 1.

Algorithm 1 NCF training

Initialize the weights and hidden latent features
for number of epochs **do**
  Perform gradient updates for latent features and neural network weights
end for
Make predictions based on the neural net trained function f, taking for each user i the js with the highest $p(p_i, q_j)$

4.2. Restricted Boltzmann machines

4.2.1. Parameter learning

In our implementation, we follow [7], making simplifications where possible. We would like to find the parameters W_{ij} that maximize the likelihood $p(V)$. Now, remark that

$$p(V) = \frac{\sum_h e^{E(v,h)}}{\sum_{V',h} e^{E(V',h)}}$$

Thus, the gradient updates are

$$W_{i,j} \leftarrow W_{i,j} - \eta \left(\mathbb{E}_{data}(v_i h_j) - \mathbb{E}_{model}(v_i h_j) \right)$$

The second term is expensive to compute, since it requires additional time and thus we will use approximate inference. Instead of maximizing the likelihood, we will maximize the constrastive divergence, a method developed by Hinton in [4].

The new gradient updates are:

$$W_{i,j} \leftarrow W_{i,j} - \eta \left(\mathbb{E}_{data}(v_i h_j) - \mathbb{E}_{model_T}(v_i h_j) \right),$$

where $\mathbb{E}_{model_T}$ simply means the approximate expected value of $v_i h_j$ obtained from Gibbs sampling over T epochs. We initialize r at their observed values, and the subsequently sample p and v.

This approximation does not try to maximize an upper bound on the likelihood, but rather make gradient steps towards the minimum in the correct direction with a potentially different size.

4.2.2. Predictions

In general, we would like to compute the posterior from the generative model $p(v_{i_1} = 1, v_{i_2} = 1, ..., v_{i_y} = 1)$, for an arbitrary number of unobserved examples y. This, in general, is intractable, since it requires 2^y number of computations for each example. Thus, we decided, for predictions, to approximate $p(v_{i_1} = 1, v_{i_2} = 1, ..., v_{i_y} = 1) \approx p(v_{i_1} = 1) \cdot ... \cdot p(v_{i_y} = 1)$. This corresponds to the mean field inference from [8], chapter 21.3. Thus, we will choose the most likely movies by their marginal probabilities, i.e, $p(v_q = 1|V)$ will be equal to

$$\frac{e^{b_q} \prod_{j=1}^{m} \left(1 + \exp\left(\sum_{l=1}^{m} v_l W_{lj} + v_l W_{ij} + b^{(j)} \right) \right)}{1 + e^{b_q} \prod_{j=1}^{K} \left(1 + \exp\left(\sum_{l=1}^{m} v_l W_{lj} + v_l W_{ij} + b^{(j)} \right) \right)}$$

Learning is summarized in algorithm 2.

4.3. Comparison of NCF and RBM

Learning for NCF is usually done through stochastic gradient descent, which is what we did above. The method suffers from the inability to find global maxima, and the instability of the learning. On the other hand, the training for the RBM model is done through a method very similar to the Expectation maximization model and thus is guaranteed to converge.

While this is an important theoretical fact, the necessity to compute marginal probabilities makes exact inference hard

Algorithm 2 RBM training

Initialize the weights W_{ij} from users to features randomly
for number of epochs **do**
  for gibbs sampler iterations **do**
    Initialize the v_i at the data, sampling hidden features from the v_i via our distribution in subsection 3.3, and then compute the expectation $\mathbb{E}_{\text{model}}(v_i h_j)$ over our sampled distribution
  end for
  Compute $\mathbb{E}_{\text{data}}$ as a sum over the observed user ratings and directly inferred hidden features.
  Update W_ij as in subsubsection 4.2.1
end for
Make predictions based on the formula in subsubsection 4.2.2

for RBM, so NCF is the only model that is able to stick to its probabilistic assumptions.

Another important fact to recognize is that predictions take less time to compute for the NCF compared to RBM after training, so NCF might still be better suited for applications that require fast predictions once the model is trained.

Another trade-off we had to make to reduce computational time when training RBM was the mean-field approach to approximate the joint distribution. This reduces the efficency of RBMs since we are not able to do exact inference, and thus we slightly deviate from the probabilistic assumptions during training.

5. Methods

5.1. Datasets

We used two *kaggle* datasets to validate our models. Both of these have discrete binary "ratings" given by users to the datasets' respective items: hotel clusters (Expedia datasets) and events (Event Recommendation dataset). Although they contain a lot more information about both users and items, we are only interested in the users and items' IDs, and the corresponding rating r_{ij} for user i and item j.

This allowed us to construct the input for our algorithms as required by these methods, constructing a sparse array of ratings, having dimension $N \times M$, where N and M are as earlier described. For the Expedia dataset, the ratings array has approximately $200M$ entries, since there are $2M$ users and 100 hotel clusters. The Events ratings array has approximately $20M$ entries, but it is different in structure since there are approximately $2k$ users and $10k$ events.

Kaggle competitions provided these datasets with winning prizes from obvious sponsoring companies. However, for privacy and security reasons, there is a disclaimer in both competitions' pages that the data points are not representative of real users behaviors. We can only take this for granted and make the best predictions we can.

5.2. Features

Since we decided to compare context-unaware collaborative filtering models, the only models we assumed we had were the matrix of ratings that users gave items. In both cases, it was a sparse matrix of zeros and ones.

5.3. Inference

We first cut down the massive *kaggle* datasets Expedia and Events Challenge to preserve only the (user, item) pairs (items were respectively booking clusters and event ids).

Then we split our (user, item) pairs into 80% training and 20% validation. On the training data in NCF, we learn user features and item features with the algorithm 1, while our RBM learned weights W_{ij} from the training data with the algorithm 2 and user hidden features. We validated the NCF by computing the MAP with respect to user labels in validation against the strongest (user, item) interactions. To validate the RBM, we held the weights constant from training and inferred hidden features for the users, and then calculated MAP against the user labels in validation.

In NCF, the number of parameters was $NK+MK+N+K$ in the PMF portion corresponding to user hidden features, item features and biases. There were $5 \cdot 2K + 5 \cdot 5 + 5 \cdot 1$ parameters in the overlaid neural network corresponding to two hidden layers of 5 neurons each, an input layer accepting the concatenation of user and item features and an output probability.

The RBM is fully characterized by its weight parameter $W \in \mathbb{R}^{F \times M \times K}$ that maps hidden features to the output space of M items by $K = 2$ possible ratings, and its individual feature and rating biases $b^{(j)}$ and b_i of order F and MK respectively.

5.4. Metric

The metric we used was mean average precision. If, for a certain user u, we let the accuracy at l be $P_u(l) = \frac{p}{l}$, for $l \leq k$, where p is the number of the first p users from our list that match the number of users from the true list, then the mean average precision is defined as:

$$MAP_k = \frac{\sum_{u \in \text{users}} \frac{1}{k} \sum_{l=1}^{k} P_u(l)}{\text{number of users}}$$

For the expedia data-set, we used MAP_5 for the expedia dataset and MAP_{200} for the events dataset, in accordance

with the evaluation metrics of the two kaggle competitions.

This metric makes sense from an intuitive point of view. We want to predict how well our model does in predicting the best users, and reward the model for accurate predictions on users that are first in the list compared to the ones that are last in the list.

6. Results

We report results of the PMF model in addition to the ones previously described for benchmarking purposes. For each one of the three models, we plot the validation error against the dimensionality of the latent vectors. The best performing instance of each model on the validation data is declared winner and we then report testing errors of these models, in Table 1.

Model		Expedia (Test map)	Events (Test map)
PMF		0.024	0.010
NCF	2 ReLU	**0.050**	0.121
	2 Tanh	0.032	0.044
	ReLUs&Tanh	0.009	0.050
RBM		0.040	**0.131**

Table 1. Test errors of each model on each dataset

The PMF model was largely developed in $T2$. We implemented the NCF model and considered the $ReLU$ architecture described in [2]. Additionally, we tested non-linear architecture using $Tanh$ activation function. This is because we were expecting some significant non-linear behavior of the latent variables. In keeping with both ideas, we have also combined these into a mixed architecture using both $ReLU$ and $Tanh$ layers. There are many other parameters to be varied in a neural architecture, such as number of neurons at each layer, but we sticked with 5 of these per layer. We have implemented the [7]'s RBM algorithm as well, and additionally to varying the dimensionality of the latent features, we have also varied the number of iterations of the Gibbs sampler. One can easily note that introducing the neural predictive function instead of naively using the standard dot product indeed produces better results, for most architectures. The RBM also performs significantly better than the benchmark, which confirms the predictive power of the generative model. Figure 3 shows the performance of all these architectures and algorithms as a function of the dimensionality of the latent vectors. The respective PMF performance is also marked for reference, with dotted thin black line.

Based on Figure 3, we have chosen the best performing latent dimensionality value for each architecture, and this trained model was used to produce the tests results of Table 1.

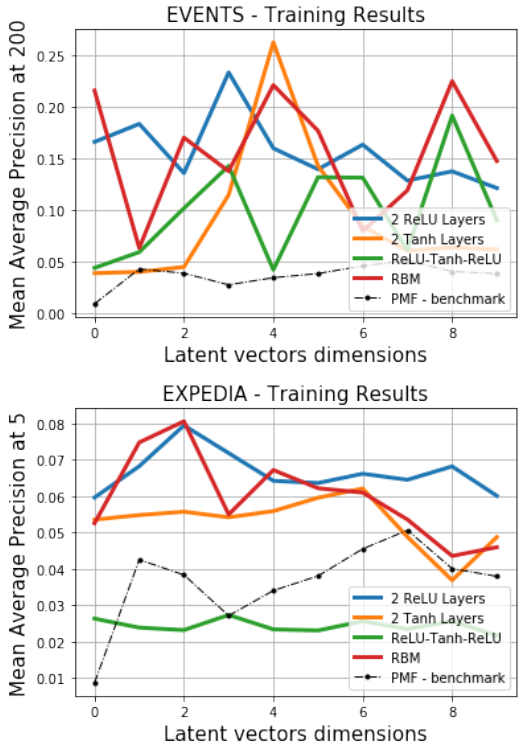

Figure 3. NCF training error in terms of hidden dimensions

Figure 4 shows the performance of the NCF model as a function of training epoch, for both data sets. These values have been normalized by their respective maximum value so the can be meaningfully plotted together. The important behavior is the convergence one, which shows that the inference step from the data behaves properly.

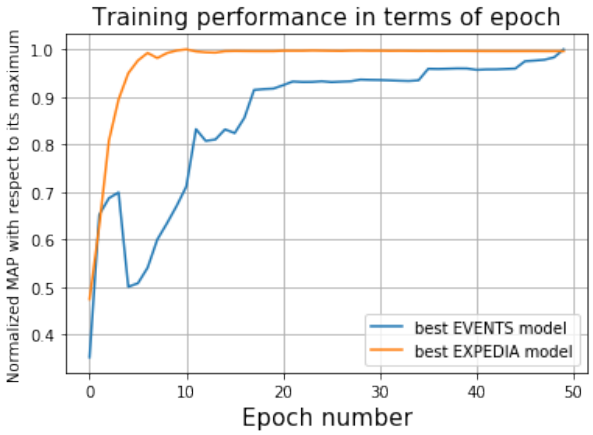

Figure 4. NCF training error in terms of epoch

To compare the results between NCF and RBM models, it is important to separate between the users for whom we have considerable amount of training data, and those for

whom we only have the bare minimum. We look at the separate predictions on "sparse users" versus "non-sparse users" of both models. We define the sparse users by counting how many items a user has interacted with, and let these be the bottom 50% in terms of this count. The non-sparse users are the top 50%. Table 2 shows these results on the respective subsets of users from each dataset.

		NCF	*RBM*
Expedia	Sparse	0.031	0.030
	Non Sparse	0.070	0.049
Events	Sparse	0.074	0.068
	Non Sparse	0.169	0.183

Table 2. Test errors on sparse and non-sparse users

7. Discussion

7.1. Difference in performance

- One thing that needs to be noted is that both models beat the PMF baseline, and thus lived up to expectations.

- One of the initial goals of the paper was testing the power of the structure of the ReLU activation neural network, that seemed rather arbitrary. We can see from our experiments that ReLU activations indeed prove to have the best results our of all neural network configurations.

- Coming into the project, we had a prior that NCF should have better results on a large dataset, while RBM should have better results on small datasets, based on [5]. Proposition 1 in [5] claims that for a discriminative-generative pair of models $h_{generative}$ and $h_{discriminative}$, then $h_{generative}$ has lower asymptotic error. Moreover, they find out that the discriminative model converges to its assymptotic error faster. There is still some value in this prior, since it can be seen that the NCF model performs slightly better than the RBM model on the events dataset, while, on the other hand, the NCF model performs significantly better than RBM on the richer Expedia dataset.

- Consequently, although the two models are not exactly a corresponding generative discriminative pair, we expected RBM to perform better on the large Expedia data-set and worse on the Events dataset, given their similarities, and their main difference as a discriminative versus a generative model.

- From the results, we see that our expectations were confirmed by the results, and that NCF performs better in both models. This is most likely true because the NCF model is able to exploit its high generality in terms of functional forms and thus find the best model for its function f through training. On the other hand, RBM does not have this power and thus performs worse than NCF on the large dataset, but has the ability to better avoid overfitting on the small dataset and is thus a more accurate model on the events dataset.

7.2. Difference in performance by user type

We can see that in both datasets, the performance of the model on sparse and non-sparse data-sets has a clear pattern. More specifically, the RBM model is better able to predict users that have rated very few items, while the NCF model is better able to make accurate prediction on users that have rated many items. The reason RBM does better on "sparse" users is a feature of the generative model that RBM uses. On the other hand, NCF does better on non-sparse users because it is better able to find complicated patterns that these users fall into using its neural network structure. We expected similar results, as we explain in section 3.

7.3. Training

Differences in training easiness were enormous. Training the RBM model proved harder than expected, because of the long time that is necessary for sampling in the gradient descent step.

The NCF training, on the other hand, was much faster per epoch, despite the neural network architecture, most likely because of the fast pytorch automatic gradient library.

Training for NCF had, on the other hand, some other problems we expected. Changing the learning rate and optimization was crucial, as many times the gradient descent method would get stuck in a local minimum of the non-convex function that was optimized. This hypothesis is supported by the fact that we obtained a 10-fold increase in MAP from PMF to NCF through changing the functional form f from the model.

7.4. Open questions

- A counter-intuitive fact is that for the NCF, the number of features that maximized the validation error was smaller for the large data-set $K = 2$ than for the small data-set $K = 3$. While this is counter-intuitive, it suggests the fact that the events data did not fit the very simple functional forms of the PMF or RBM model, and a function that's less natural, obtained through a neural network is better able to explain the data.

- We thought about the possibility of combining the two models to give rise to a better model. One such way would be to train both an RBM and an NCF, and use the RBM for sparse users and the NCF for rich users. While this model seems appealing, and it would definitely provide better results that both RBM and NCF, we are unsure about a probabilistic approach that would explain such a model. Without a probabilistic model that clearly pinpoints the assumption of the model, we are not currently satisfied with pure experimental that show the power of such a model.

- Another counter-intuitive fact is shown in Figure 4. The training of NCF on the events dataset seems very noisy, especially during the first 10 epochs. We think this behavior is happening because training is done with respect to the cross-entropy loss, which in theory should be decreasing throughout loss, but the training does not guarantee increase in MAP. It would be interesting to find a model that guarantees the increase of the MAP throughout training, through a smooth approximation of this non-differentiable function.

- It is also counter-intuitive that the graphs of MAP by K looks very noisy in every graph. It would be interesting to understand theoretically why this effect exists in the results.

8. Conclusion

The scope of our project was limited in some sense due to our datasets being paradoxically sparse and massive, so we often had to subsample to train effectively. Additionally, since we implemented the collaborative RBM and NCF models from scratch, and the former was impossible to implement in PyTorch due to the customized gradient updates of [7] that included the stochastic element of a Gibbs sampler at each epoch. As such, while our final implementation of the collaborative RBM was respectably fast, it was necessary to heavily vectorize the updates and equations in [7] in Numpy, which was quite challenging.

Nevertheless, we feel that our investigation specifically has arrived at some important conclusions in section 7 as a result of this project, concerning the differing power of RBMs and NCFs to explain sparse users and sparse datasets in subsection 7.2.

If we had more time to investigate the collaborative filtering problem, we would be interested to explore how these methods, while exceedingly inventive in creating new user and item features, could be used in tandem with other user and item features (i.e context-aware collaborative filtering), as in [3] wherein the authors fruitfully incorporate side information into a vanilla PMF algorithm.

9. Acknowledgments

We would like to thank Professor Rush for referring us to the work of Russ Salakhutdinov which proved invaluable for this project. We would also like to thank Nancy and the CS281 course staff for their feedback which helped clarify the project and its goals.

References

[1] Nam, J. et al., 2013. Large-scale Multi-label Text Classification - Revisiting Neural Networks.

[2] He, Xiangnan, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu, and Tat-Seng Chua. "Neural Collaborative Filtering." 2017.

[3] Adams, Ryan Prescott, George E. Dahl, and Iain Murray. "Incorporating side information in probabilistic matrix factorization with gaussian processes." arXiv preprint arXiv:1003.4944 (2010).

[4] Hinton, G. (2002). Training Products of Experts by Minimizing Contrastive Divergence. Neural Computation, 14(8), 1771-1800.

[5] Ng, Andrew Y., and Michael I. Jordan. "On discriminative vs. generative classifiers: A comparison of logistic regression and naive bayes." Advances in neural information processing systems. 2002.

[6] Mnih, Andriy, and Ruslan Salakhutdinov. "Probabilistic matrix factorization". In *Advances in Neural Information Processing Systems*, pp. 1257-1264. 2007.

[7] Salakhutdinov, R., Mnih, A. & Hinton, G., 2007. Restricted Boltzmann machines for collaborative filtering. Proceedings of the 24th international conference on machine learning, pp.791798.

[8] Murphy, Kevin P. Machine Learning : *A Probabilistic Perspective.* Adaptive Computation and Machine Learning. Cambridge, Mass.: MIT Press, 2012.

[9] https://www.kaggle.com/c/event-recommendation-engine-challenge

[10] https://www.kaggle.com/c/expedia-hotel-recommendations