Local Business Recommendation on Yelp Dataset

Chao Huang, Lin Jiang, Shuo Yang, Han Xu
Data Science Institute, Columbia University, New York, 10025, US
UNI&Email: ch3474, lj2493, sy2886, hx2282@columbia.edu
GitHub Handle: HenryNebula, jianglinlinlin1, northernleaf, Monicaxuhan

***Abstract*—In this project, we formulate our problem as providing local business recommendation for our customer on a crowd sourced review sharing platform. Our main target is providing potential places of interest for each user which he or she will be highly likely to visit in the future. In order to solve this problem, we implemented several personalized recommendation algorithms with the help of existing open source packages. Also, thanks to the abundant auxiliary information contained within the Yelp dataset, we also conducted extensive experiments to figure out how these side information may help create more personalized recommendations. In the end, we also provided some empirical conclusions on how to efficiently combine context based and content based information for a generic recommendation task.**

I. Introduction

This project is formulated under the scenario that a local business recommendation task was given to a team of data scientist of a crowd-sourced review forum, which mainly displays reviews for local business. As for most company running review forums online, for example, Yelp and TripAdvisor, the major revenue comes from advertising. Therefore, accurate and highly personalized recommendations can not only help strength user loyalty, but also help promote local business and encourage them to post more advertisements on our platform.

The objective of this project is focusing on improving the profit of our company by increasing the number of future interactions between our customers and local business. Since a large number of new interactions may come from the recommendations we provide for our users, more personalized and and accurate recommendations would inevitably increase the chance of creating new links between a user and a local business. Also, the recommender system should highly rely on users' review history in order to generate more meaningful suggestions. What's more, it should take the location information into consideration, since a very far business would be much less attractive than nearer counterparts with almost the same functions in most cases.

In the following sections, we will first conduct data cleaning and exploratory data analysis on our dataset in Section II. Then we will provide a brief introduction of the algorithms and implementation details of our models in Section III. By first introducing our experiment settings, we show our experiment results and detailed explanations in Section IV. Finally, we conclude this project with several insightful findings and discuss what we can do in the future in Section V.

II. Preliminaries

In order to test our ideas and assumptions, in this project, we use a public dataset from Yelp 2019 data challenge [1], which includes sampled reviews from Yelp within 10 metropolitan areas across 2 countries. The dataset not only includes millions of reviews of local business provided by the Yelp users, but also includes several useful files with auxiliary information. In this project, we mainly utilized the *business.json* and *user.json* files in the original dataset to collect demographic features for users and geographical and business-related features for business.

A. Data Cleaning and feature engineering

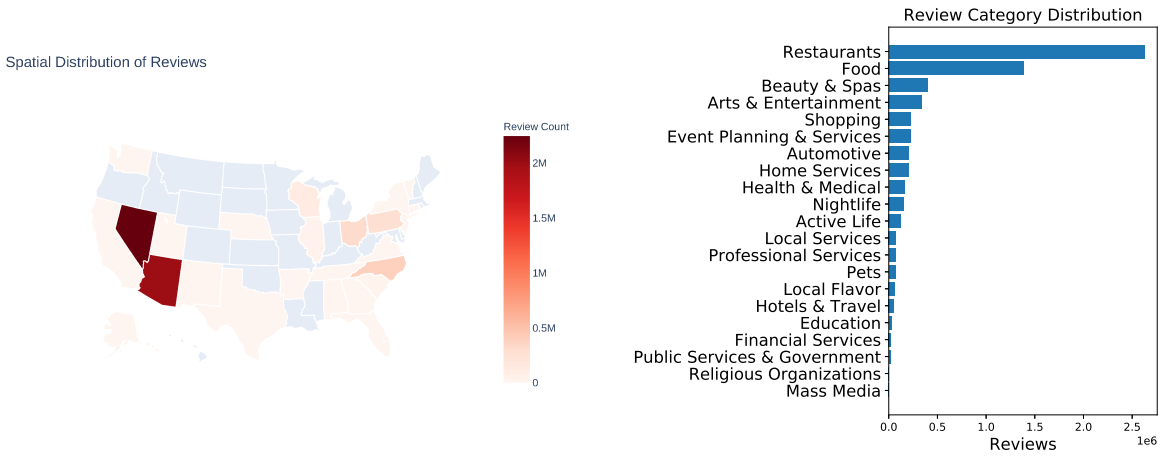
The data cleaning part can be divided into three major parts.

First, we deal with the review files. Since all review records in *review.json* file also includes a paragraph of review in natural language, it takes up huge amount of disk space. Therefore, we first eliminate all these paragraphs as none of our models utilize them currently. There are also several interesting features correlated with each review, for example, the number of "cool" or "fun" that other users comment on a specific review. Although currently we don't utilized them explicitly, we still preserve them in the tidy version of review files, since they may serve as a confidence score for each review records. How to effectively incorporate this weight into the models would be one of future directions for this project.

Second, we selected useful fields in the user profiles. We mainly modified and chose three fields for every records. First, we deleted the friends of each user from the file since we don't consider social network effect at this point. Then we calculated the number of years that a user was an elite user of Yelp using the field of "elite", and calculated the number of days for which he or she will be an user of Yelp till the end of 2019, using the field of "yelping_since". Last, we aggregated all the compliments that a user received into a count feature by summing up the counts of all fields that starts with word "compliment".

Third, we filtered the business profiles. Currently, we discard the attributes of each business. The main reason is that

[1] https://www.yelp.com/dataset/challenge



(a) spatial distribution of reviews

(b) category distribution of reviews

Fig. 1. Spatial distribution and category distribution of all reviews

this field contains numbers of sub-fields, for example, whether this business accepts bit-coin or whether it's good for kids. As the attributes vary a lot between different categories of business, encoding all of them will lead to a huge dictionary and sparse one-hot encoding features, which may make models hard to converge. Therefore, we left these attributes as future work. For the geographical information, we only preserved the state information of each business and discarded city, address and longitude/latitude in order to reduce feature dimensions. For the field of "categories", originally it may include both the parent category and children category. In order to transform them into a unified representation, we first found out all top-level categories within the "categories.json" file and classified every category up to the top-level category.

When feeding side information to the model that accepts it, we introduced dummy variables for categorical variables and maintained the original value of all continuous variables as dense features.

B. Exploratory Data Analysis

As initial data exploratory, we plot the spatial distribution of reviews and the distribution of top categories of all reviews. We only display the spatial distribution in United States and the same pattern applies on Canada. In Fig. 1(a), it's apparent that in this dataset, some states have much more reviews of its local business than others, for example, Nevada and Arizona. And from Fig. 1(b), one can observe that business correlated with Restaurants and Food has the most number of reviews. Therefore, when we are evaluating a ranking metric, it may not be a good idea to rank over a series of random sampled business, otherwise the majority of business will be situated in these two states or belong to these two categories. In order to make the offline evaluation more similar to online scenarios, where we were asked to provide recommendations among business within similar categories and states, we would generate test candidates with same category and state as the ground truth test sample. This part will be introduced in a more detailed way in Section IV-A2.

We also implemented an interactive shiny app[2] showing the spatial distribution of reviews given by users with more than 500 reviews in the dataset. Though these users are quite active given the large number of reviews they have posted, most of their reviews are still for business within only one or two states. Therefore, it is reasonable to provide "local" business recommendations to users in most cases.

III. Our Proposed Method

In order to solve our business problem, we implemented several popular recommendation algorithms both in academia and industry, including vanilla Matrix Factorization (also known as SVD in literatures), Collective Matrix Factorization (CMF for short), Factorization Machine (FM for short) and Bayesian Personalized Ranking based Matrix Factorization (BPR_MF for short). In this section, we will explain how we implemented these algorithms using existing packages.

A. SVD

For this vanilla Matrix Factorization algorithm, we used SVD model from the Surprise package [3]. Since we want to tested the RMSE, we trained the model on user-business pair and their explicit ratings. We first transformed the training dataset to be surprise ready and then used its pre-defined predict function to predict user-business ratings on the test set. The RMSE we obtained based on the default parameters were about 1.3, which was acceptable in terms of fitting correct pattern of the underlying data. Then, we tuned the model for the main purpose of increasing our NDCG score as we wanted

[2]https://ysyang.shinyapps.io/yelp_review_map/

[3]https://github.com/NicolasHug/Surprise

the final rated business in the test set be ranked higher in our recommended list.

B. CMF

One interpretation of CMF is that it not only fits the interaction matrix as SVD does, but also fits the matrix of side information, for example, users v.s. user features. Note that by sharing user or item latent vectors during the fit of interaction matrix and side information matrix, the model can jointly learn both the context and content based features using a linear combination of loss functions of all these parts.

We utilized the cmfrec package[4] to construct a CMF model. Note that since a full CMF may utilize both the user and business side information, it may also have degenerated versions of models, for example, using only one aspect of side information or even totally ignoring side information (which in fact is the same as SVD). In order to test whether adding side information may help improve the recommendations, we run all these four versions for comparison.

C. FM

FM is similar to CMF in that it can take in side information. In addition, FM also includes interaction terms to provide better results in theory. However, in practice, like what is implied in this project, FM does not always beat the MFs.

The fastFM package[5] is used to build the FM models. Similar as CMF, we try to run all four versions (None, User, Business and Both) for comparison.

D. BPR_MF

Although the original interaction records between each user and each business are review ratings in this dataset, which is known as explicit feedback, we can still binarize the data and apply implicit models to fit the dataset. The motivation of introducing BPR model is that its objective function is specially designed for ranking tasks and the negative sampling technique introduced in the training phase may help improve the generalization ability of the model. Though BPR model can only work on implicit feedbacks and the transfer from explicit to implicit may cause the loss of information, we can still claim the special designs in BPR model help solving our problem in a better way, if the performance of BPR is much better than models using explicit feedback. Therefore, we utilized the BPR model from the Implicit package[6] to test our hypothesis that the choice of loss function may influence the final performance.

IV. Experiments

In order to check the correctness and efficacy of our implementations, we conduct experiments on the filtered Yelp 2019 datasets to answer the following questions.

[4]https://github.com/david-cortes/cmfrec

[5]https://github.com/ibayer/fastFM

[6]https://github.com/benfred/implicit

TABLE I
Dataset Statistics

Dataset	#user_id	#business_id	#stars	sparsity
Yelp-Original	1637138	192606	6685900	99.998%
Yelp-Filtered	122824	192606	3486292	99.985%

- **Q1:** How does our proposed method perform compared with the baseline models?
- **Q2:** Does side information help improve the quality of recommendations?
- **Q3:** How do hyper parameters affect our models?

In what follows, we first describe the experimental settings and then answer the above four questions.

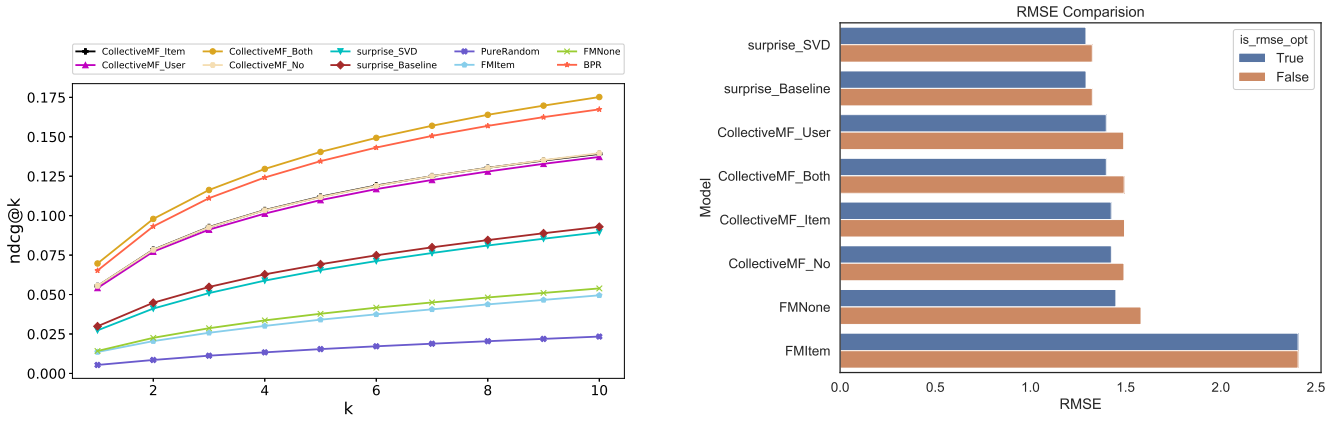
A. Experimental Settings

1) Datasets: We experimented on the filtered subset of the original yelp 2019 dataset. Since it would be quite hard to provide recommendations for users with limited number of interactions, we only keep users with at least 10 reviews in our filtered subset. However, we don't discard any business manually, so that unless it's a cold-start business or it has missing value on item profile fields, it will always be possible to include it in the final recommendations.

The statistics of the sampled dataset and the original dataset are summarized in Table I. Compared with the original dataset, the sampled dataset shares similar sparsity level with the original dataset, but has fewer than 10% users. So it might be a good trade-off to balance the scale of filtered dataset and maintaining statistical features. For those users discarded for this project, we may design a separate strategy for them in the future, considering the relatively small amount of reviews they have made before.

2) Train Test Split: As we are trying to simulate the real world scenario, which is providing recommendations for every user's next interaction, we use a leave-one-out strategy for evaluating the model performance. For all reviews provided by user u, we chose the latest one with the most recent timestamp as the test sample, and treat all the others as train samples. In this way, we can evaluate the model performance for every user. This is very similar with the online query scenario of Yelp, which also asks the user to provide the location and type of business before sending recommendations back to you.

Since in this project, we are more interested in the ranked recommendation results provided by the model, it makes sense to evaluate the ranking metrics introduced below. However, due to the large number of business in this dataset, it requires huge amount of time to generate a ranked list among all possible business. Also, it's also not very reasonable to compare two business with completely different functionalities and features, for example a bank in LA and a restaurant in NY. Instead of randomly sample business out of all available ones, which may be dominated by popular states and categories as we have seen in the exploratory data analysis part, we first clustered business



(a) ndcg@k performance

(b) rmse performance

Fig. 2. Top-N NDCG and RMSE performances of different models

to subgroups, using its top category and the state information. Then, for a test sample (u_i, b_j), we will randomly sample out at most 200 business with the same top category and state as b_j, as the candidates for generating final recommendations. Then the model only need to predict at most 200 business per user to provide the final ranked results. Note that we also guaranteed that a business that a user has already interacted before will never be chosen as a possible test candidate. In this way, we can evaluate our model offline but with a metric very similar to real world scenarios.

3) Evaluation Metric: In the evaluation stage, we evaluate our model with two groups of metrics, namely regression metric and ranking metric. For the first group, we calculate the **RMSE** (root mean squared error) between the prediction and ground truth of each rating in the test set. Although our major concern is whether the left out test samples will be included in our final recommendations and whether they can be ranked high, we still use this metric to see how well our model fit the underlying pattern and generalize to unseen data. For final model selection, we will still use one of the ranking metrics as the primal selection criterion.

For the second group, we evaluate our model using and **NDCG** (Normalized Discounted Cumulative Gain). Instead of just counting how many items among the top-N recommendations are truly relevant ones. NDCG assigns a higher weight to hits ranked higher in the recommendation list.

4) Baselines: We compared our models with two relatively weak baseline, namely a pure random model and baseline model and one relatively strong model, SVD, all without the use of side information.

The pure random model would just provide a random ranking over all test candidates when generating final recommendations. This baseline serves as the bottom line of all models.

Note that the "baseline" model we mentioned here is actually a degenerated version of SVD, which only fits one bias term for each user and item as latent vector. This model is known as "BaselineOnly" model in surprise[7].

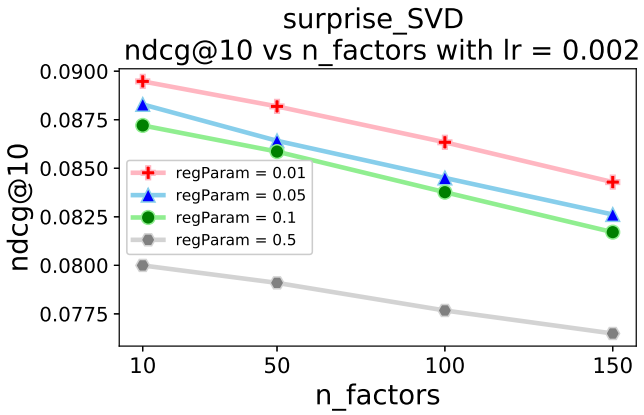
5) Parameter Settings: In order to fine tune our model, we search for the optimal combination of hyper-parameters through grid search.

For most models, we tune regularization term and number of latent factors through cross-validations. The grid search range for the optimal regularization term is from 1e-4 to 1e-2 for most models, and from 0.1 to 1 for FM model since it's more likely to overfit. For the number of latent factors, we tried $[2, 4, 8, 16]$ for most models. For models using SGD as the optimizer, we also fine tuned the learning rate within the range from 1e-3 to 1e-4.

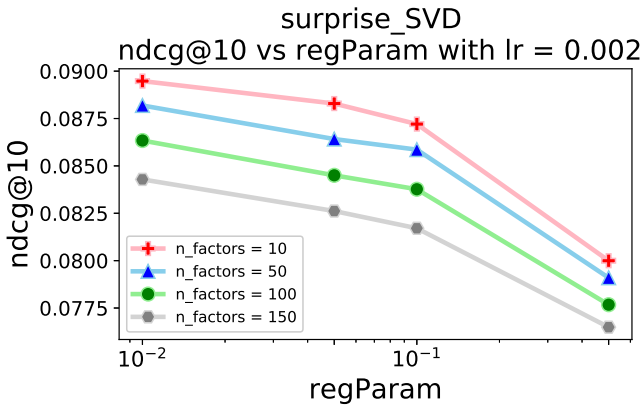
For CMF models, since the total loss is a linear combination of three losses corresponding to interaction, user side information and item side information respectively, the weights vector has two degree of freedom in total (since they will always add up to one after normalization). However, the weights are continuous values and it would be quite hard to tune every possible combination. Therefore, after trying several initial values, we use the following empirical rule for tuning parameters. The weight for the interaction part will always be a free parameter and the weight will be 0.3 and 0.2 for user and item side information respectively. If only use side information or item side information is utilized, then it will have the weight of 0.5 by itself. In this way, we only need to tune one weight, and it helps control to what extent our loss relies on context and content information relatively.

Also, we use the same regularization parameter for the first order latent vector and second order latent vector to reduce the number of hyper-parameters of FM models, since they take quite a long time to converge.

[7] surprise.prediction_algorithms.baseline_only.BaselineOnly

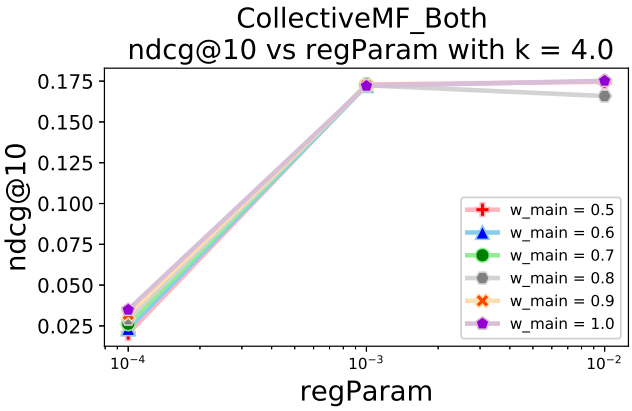


(a) SVD: ndcg@10 vs n_factors

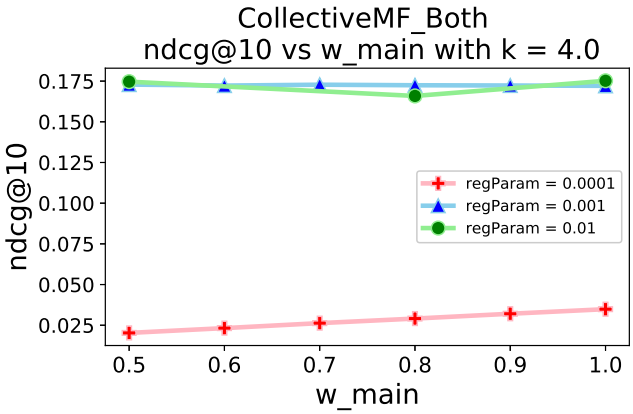


(b) SVD: ndcg@10 vs regParam

Fig. 3. The effect of hyper-parameters on the SVD model



(a) CMF: ndcg@10 vs regParam



(b) CMF: ndcg@10 vs w_main

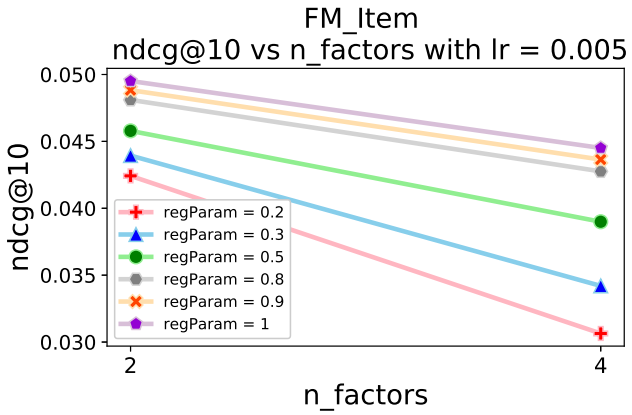
Fig. 4. The effect of hyper-parameters on the CMF model

B. Model Performance (Q1)

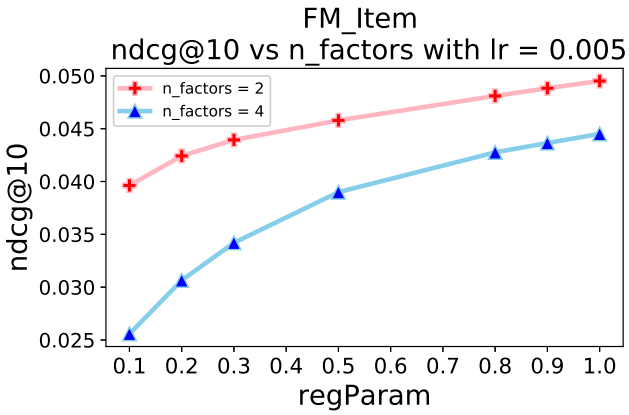
As shown in Fig. 2(b), when using RMSE as the criteria for choosing models, the SVD baseline seems to be the best one, although all other models except the FM family achieve relatively similar performances. However, if a ranking metric is utilized, as shown in Fig. 2(a), all models except the FM family achieve better results than the SVD baseline. Note that since the FM model doesn't converge with every set of parameters after 100 epochs when user side information is utilized, we don't present the results of FM models with user side info or with both side information. And since models specially designed for ranking metrics like BPR and PureRandom baseline, we obviously don't have their RMSE results.

From these results, we can have several interesting observations,

- All models seem to learn something from the dataset since they all beat the pure random baseline when NDCG@10 is used as the evaluation metric. In order to find out how ranking quality changes with different recommendation sizes, we explored NDCG values from when $K = 1$ to $K = 10$ as shown in Fig. 2(a), where K is the number of recommended items. We observed that CMF performed the best when utilizing both user and business side information. This makes senses and it helped confirm that adding side information could help improve our recommendation quality in this case of Yelp dataset. NDCG on all models improved as we increased the recommendation size and the increase of K could give a larger boost in CMF and BPR than for other models. Note that the SVD and Baseline model has simiar NDCG@K performance and the Baseline is slightly better. This may come from the fact that this dataset is extremely sparse and the SVD model has far more parameters than a simple Baseline mode and would be easier to overfit on the training set.
- The RMSE metric may not be consistent with the ranking metric, and a small difference between the regression

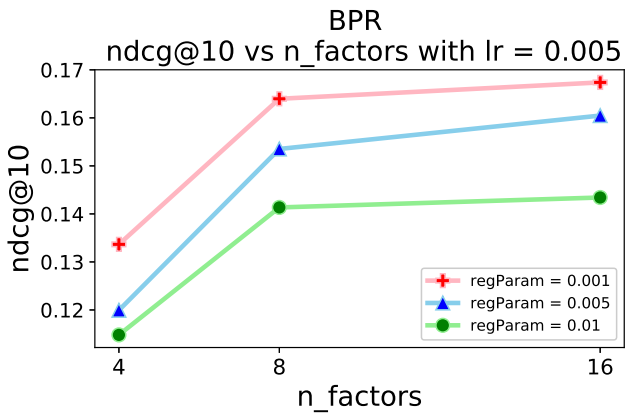


(a) FM_Item: ndcg@10 vs n_factors

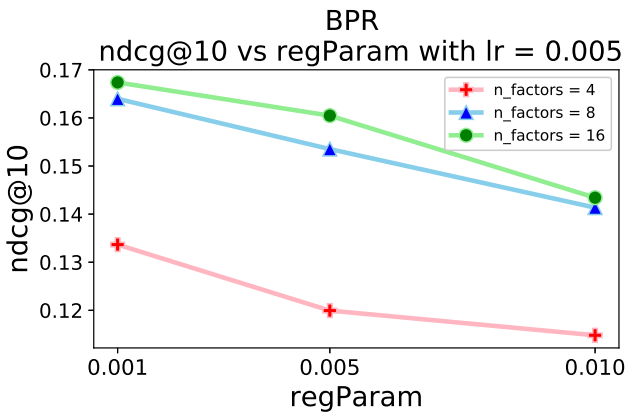


(b) FM_Item: ndcg@10 vs regParam

Fig. 5. The effect of hyper-parameters on the FM_Item model



(a) BPR: ndcg@10 vs n_factors



(b) BPR: ndcg@10 vs regParam

Fig. 6. The effect of hyper-parameters on the BPR model

metrics can still lead to a huge gap between the quality of final recommendation list. As shown in Fig. 2(b), the blue bars stands for the performance of optimal models selected using RMSE, and orange bars shows the RMSE of the optimal model with highest NDCG@10 scores. We can observe that almost no model can have the lowest RMSE and highest NDCG@10 at the same time. Therefore, we need to choose a metric that better fits our problem definition and our targets. For the problem of local business recommendation, obviously providing a high-quality recommendation list is more important than predicting the rating of a review. Therefore, we will focus on ranking metric in the followin part.

- The choice of optimizer and optimization strategy may lead to different results even under the same model formulation. The CMF model which doesn't use any side information should be almost the same as SVD since they are both vanilla matrix factorization models. However, we can observe a small difference in RMSE and a larger gap in NDCG@K between the degenerated CMF model and the SVD baseline. After checking the implementations, we observe that the CMF is optimized using L-BFGS as the optimizer with the jacobian matrix computed by Tensorflow, however, the surprise SVD method is optimized using SGD without second order differential information. Therefore, the differences in the performance may come from the choice of optimization algorithm.
- The factorization machine seems to perform as the worst models on this dataset, even the one without any side information. One possible reason is that the factorization machine will fit more latent factors than traditional matrix factorization models. However, since our dataset is quite sparse, the more parameters a model has, the more likely that it will overfit the train dataset. Therefore, the factorization machine may be preferred for a denser interaction matrix.
- Note that for BPR model, although there may exist an

information loss during the transformation from explicit feedback to implicit feedback, it performs actually quite well on NDCG@K compared with the baseline. Therefore, it may indicate that a rank loss may be more appropriate than the point-wise loss when the final target and evaluation metric is about providing a good ranking result.

C. Effect of Side Information (Q2)

In this section, we mainly focus on models that can take both the context information and content information, namely the CMF family and FM family and use the ranking metric for evaluation.

For CMF family, using the optimal hyper parameters for each model, we may observe that by adding a single aspect of side information, the model doesn't perform better than the one without any side information. However, when both side information are utilized, the model has a sharp increase in the performance. This phenomenon can be explained as follows. Due to the mechanism we used in the tuning phase, if both side of information is utilized, each side information will have a smaller weight compared with the cases when only one side information is used. Therefore, the model will mainly depend on the context part. Since the side information contains less important information compared with the interaction records, depending on the side information heavily may decrease the accuracy of prediction. From another aspect, it may serve as a regularization part which helps prevent the model from overfitting the context information. Therefore, we can observe an increase in performance when both side information is used. However, it also reflects that CMF model may be quite sensitive to hyper-parameter settings. Therefore, more experiments or a auto-tuning framework may be needed to validate this observation.

For FM family, by adding item side information, the performance of RMSE is slightly better than that without any side information, but the performance of NDCG worse than one without side information. Although the fastFM package doesn't converge on dense user side information and we don't have further observations for the model with both sides of information, from the experiments we have found that FM model will be easier to converge when sparse features are utilized, since most user side features are dense features.

D. Impact of Hyper-parameters (Q3)

Due to limited space, here we mainly focus on four specific models and study how their performances would change with regard to different hyper parameter settings. These models are SVD, CMF_Both, FM_Item and BPR. And we also unified the name of number of latent factors as "n_factors" and regularization parameter as "regParam". For CMF model, the "w_main" parameter stands for the weight of the loss of context part.

For SVD model, as shown in Fig. 3(a) and Fig. 3(b) when the learning rate is fixed as 2e-3 (the optimal value found by grid searching), the model seems to perform better with a smaller regularization term and a smaller number of latent factors. The model seems to be saturated with a small number of model parameters, which may due to the sparse characteristic of interaction matrix.

For CMF_Both, as shown in Fig. 4(a) and Fig. 4(b), it seems that the combined model is not very sensitive to the choice of linear combination weights, but quite sensitive to the regularization term, since we can observe a sharp increase of NDCG when the regularization term increases from 1e-4 to 1e-3 under all settings of linear combination weight. Also, this observation is consistent with the tuning instructions given by the package author who suggests the choice of regularization term may affect the convergence of this model. Therefore, when multiple data sources are fused into one model, the regularization term always plays an important role in balancing the fit over different domains.

For FM_Item model, the performance is better for n_factors is 2 than 4 as shown in Fig. 5(b), with the same regularization parameters. When n_factors is fixed as shown in Fig. 5(b), the model performance is improved as regularization gets larger and the improvement is more sensitive in the low range of regularization. It seems that the FM_Item model prefers less latent factors and heavier regularization. This is consistent with the fact that FM is a complex model and our dataset is quite sparse.

For BPR model, as shown in Fig. 6(a) and Fig. 6(b), when the learning rate is fixed as 5e-3, the model seems to fit the model better with a smaller regularization term and larger number of latent factors, in other words, the model selection process tells us that a more complicated model may be preferred for a better fit. One of the possible explanations is that with the help of negative sampling in BPR scheme, the model are less sensitive to overfit problem. And since every positive training sample may be coupled with different negative samples within different epochs, it helps increase the ability of generalization.

V. Conclusion and Future Work

In this project, we tried both pure context based models and models along with side information. Through extensive experiments, we study the effect of hyper-parameters and data size and the pros and cons of each model. We find that Collective Matrix Factorization has the best effect of merging side information with original interaction information.

In the future, in order to make a more productive model, we think there are several things we can try to improve the performance and robustness of our model,

- Adapt two phase recommendation models to filter out irrelevant movies before applying a ranking model to generate recommendations. Currently, the test candidates are

sampled randomly from the group sharing same category and state information. However, if we can apply a more complicated model, for example, a recall model based on business similarities, then the offline testing framework would be more similar to the real world scenario.

- Add multi-thread support and batch learning framework. So far, most model needs to load all the data into memory and some can only perform fitting on only one core (e.g. fastFM). One way to solve this problem would be using deep learning framework like Pytorch or Tensorflow, which already has auto-grad module by itself and supports batch learning. Therefore, almost any loss function can be optimized in a SGD manner without manually calculating the gradients. For multi-thread support, it may be useful to use Open-BLAS library or rewrite core code using Cython or Numba.
- For sparse side information, another interesting experiment would be testing whether deep neural models like "wide and deep" may help with this big dataset.