Yelp Restaurant Recommender System

Si Gao
NYU CDS
sg6766
si.gao@nyu.edu

Xinli Gu
NYU CDS
xg588
xg588@nyu.edu

Xirui Fu
NYU CDS
xf620
xf620@nyu.edu

Wenxin Zhang
NYU CDS
wz2164
wz2164@nyu.edu

Abstract

The advent of crowd-sourcing review platforms makes business information readily accessible for customers. Yet, as the number of choices becomes overwhelming, there is need to filter, prioritize and personalize relevant information in order to alleviate the information overload. In this paper, we build a Yelp restaurant recommender system to provide users with personalized restaurant recommendations. We compare popular collaborative filtering (CF) and content-based (CB) methods that consider user-business interactions, restaurant attributes and text mining of user reviews. We also construct the state-of-the-art LightFM hybrid model which unites the advantages of content-based and collaborative filtering recommenders and thus improves the predictive power. Our framework to build the outperforming recommenders serves as a practical guide to help the platform provide commercial restaurant recommendation service.

1 Business Understanding and Objective

Over the recent years the digital information has experienced exponential growth, which fueled the success in many popular crowd-sourcing review platforms. Industry experts estimate that the global crowd-sourcing market is expected to gain significant lift to reach over $150 billion by 2027, growing at a CAGR of 36.5%.[1]

With the vast amount of information posted every day, customers face the challenge of information overload. Take restaurant recommendations for example. Although the restaurant business is a mega industry with projected $900 billion in sales, it is estimated that 99% of restaurants are family-owned small businesses with fewer than 50 employees.[2] A lack of concentration means that restaurants are in general very diverse in terms of food flavors, hours, categories, service and even vibes. Also they are highly localized with a high information barrier even for local customers. People still heavily rely on recommendations from their social circles, or just have to adventure new options themselves. This is at odds with other businesses on crowd-sourcing platforms such as movies or e-commerce where items are much less localized. Given the challenge in finding individualized and desirable restaurants, it is thus critical to the platform's success to provide customers with filtered, prioritized and personalized recommendations.

In this paper, we explore various methods in building an effective Yelp restaurant recommender system. Our objective is to construct a restaurant recommender that can be used to help Yelp users find potential restaurants of interest from vast ma-

[1] PR Newswire: Global Crowd-sourcing Market is Expected to Reach US$ 154.835 Mn by 2027 (link)

[2] Wiki: United States restaurant industry (link)

jority of unrated restaurants. This recommender system can greatly mitigate restaurant search costs and improve user experience, which in turn will lower customer churn and grow the customer base for the platform.

The recommender system is among the most valuable applications of machine learning in business from e-commerce, web search to social media. It became widely known during the Netflix prize challenge, where Koren et al. (2009) reported that collaborative filtering via matrix factorization is highly effective. Among the most popular is collaborative filtering (CF) which identifies the similar users and makes recommendations by what common users prefer. The CF model is known for the "cold start" issue when a new user has too few ratings. The content-based filtering (CB) that identifies similar items based on item feature is much less prone to this issue. In recent years the hybrid approach is becoming popular as it overcomes the drawbacks of each individual approach.

In this paper, we construct both classic CF and CB machine learning models, and build an ensemble model through stacking. We also consider the state-of-the-art LightFM hybrid model that unites the advantages of both CF and CB frameworks.

2 Data Understanding

2.1 Data Source

The data we use is a subset of Yelp's businesses, reviews, users, check-in and tips data that Yelp offers for academic purposes.[3] We use the business, review and user data, which contain more than 200k businesses, 8 million reviews and 2 million users respectively.

2.2 Exploratory Data Analysis

2.2.1 Business Dataset

The business dataset has the following columns: "business_id", "business_name", "address", "city", "state", "postal_code", "latitude", "longitude", "stars", "review_count", "is_open", "attributes", "categories", "hours".

The column "attributes" whose values are in dictionary type, contains rich information of the business. Their values are either boolean (e.g. "Wifi", "takeout") or ordinal ("price range"). Figure 1 shows the number of businesses in 15 cities, where Las Vegas, Toronto and Phoenix have the most businesses in the sample.

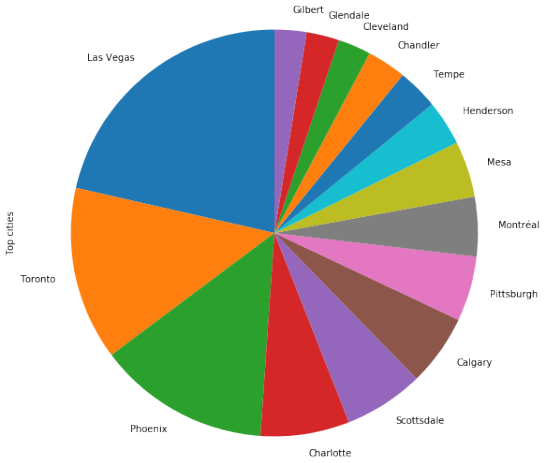

Figure 1: Top 15 cities with the most businesses

[3] Yelp open dataset: an all-purpose dataset for learning (https://www.yelp.com/dataset)

2.2.2 Review Dataset

The review dataset includes “review_id”, ”user_id”, “business_id”, “stars”, “useful”, “funny”, “cool”, “text” and ”date” columns. Figure 2 summarizes the top cities that have the most reviews. Las Vegas outnumbers other cities by a large margin, followed by Phoenix, Toronto, Scottsdale and Charlotte.

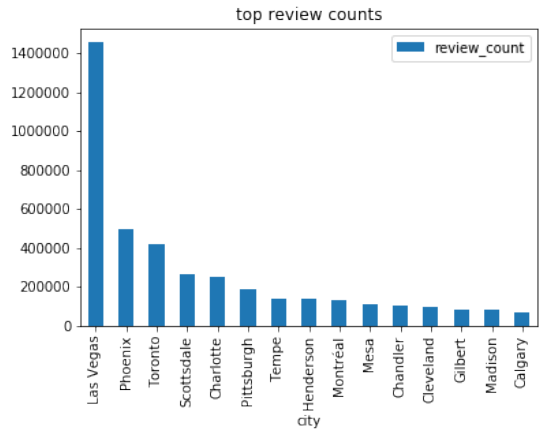

Figure 2: Top cities with the most review counts

Besides the total number of reviews, the average number of reviews per user and per restaurant should also be considered. Sparse review data, i.e. having too many inactive users or restaurants with too few reviews, cast serious challenge to the recommender system. In the appendix we plot the average number of reviews per restaurant and average reviews per user in Figure 9 and Figure 10. Las Vegas, Phoenix, Toronto and Charlotte rank highly on both lists.

2.3 Choice of Location

In order to make the recommendations relevant to local customers, we build the recommender for businesses within a city instead of the whole continent. This ensures that recommendations are accessible. Although Las Vegas is a tempting choice, we decide not to choose a tourism city where many customers are not local.

We choose to build the restaurant recommender in Phoenix. Phoenix not only has a decent sample size in businesses and reviews, but also ranks highly on average reviews per restaurant and per user. This means Phoenix could suffer relatively less from the rating sparsity issue. Our framework can be easily extended to other cities.

3 Data Preparation

3.1 Review Data Filtering and Cleaning

We merge the review and business datasets to combine user ratings with business locations and features, from which we create the samples of restaurant reviews in Phoenix. 3 percent of users have multiple reviews on the same restaurant, resulting in duplicate user-restaurant ratings. Instead of taking the simple mean, we keep the user’s latest rating on each restaurant, assuming that users may refresh their views and the newest is the most relevant.

We also find the user ratings are very sparse. The user-restaurant rating matrix has a sparsity of above 99%. To alleviate sparsity and reduce noise, we filter out inactive users who provided fewer than 5 reviews in the sample. Next, we filter out inactive restaurants whose reviews are fewer than 30.

3.2 Text Data Cleaning

In addition to ratings, users also provide review comments on restaurants. In our sample there are

nearly 200,000 reviews for Phoenix restaurants. Figure 11 in the appendix shows the word cloud of reviews in Phoenix. We preprocess the text data by first tokenizing the review documents. Then we perform several steps of processing such as stopwords removal, stemming and lemmatization in order to get cleaned text data for our NLP modeling.

3.3 Business Feature Cleaning

In the "attributes" column, each business has its own attribute dictionary which is composed of specific restaurant features, for example, "RestaurantsGoodForGroups", "GoodForKids" and "Alcohol". Most of these attributes are True/False binary values. We convert the dict-type "attributes" into 39 new dummy variables.

We also convert "hours" into dummy variables to indicate whether restaurants are open for each day of week. Variables with too many missing values (e.g. "DietaryRestrictions", "AgesAllowed") are dropped. We end up with 44 features.

4 Modeling and Evaluation

4.1 Evaluation Metrics

Typically, the recommender system problem falls into two categories, with either explicit or implicit feedback. With explicit feedback, we are able to predict the ratings on each item in the given scale and the problem becomes a regression task. Common evaluation metrics for this type of problems include MSE, MAE and RMSE. On the other hand, implicit feedback means no exact ratings are presented for targeted items. The output of this system would be an ordered list of items for each user, which can be measured by mean average precision @k (MAP@k). MAP@k measures the average of the fraction of known positive items in the first k positions of the ranked list of results.

For our Yelp recommender system, the explicit feedback (ratings) is available. So the objective is to predict the rating as close to the actual as possible, and we pick MSE as the primary evaluation metric. Lower MSE means the system has made less mistakes and can learn more latent features of users and restaurants. However, the hybrid model (LightFM) is a ranking algorithm which only produce a list of items as the output, and we use MAP@k as complementary metrics to compare model performances. In particular, we chose k to equal to 1 because in our train-validation-test split mechanism, only one rating is left for both validation and test sets.

4.2 Train-Validation-Test Split

We split the user-business ratings data into three sets. For each user we sort the ratings on the time stamp and apply out-of-time split. We leave out the last rating in the test set, and the last but one review in the validations set. The model evaluation process is as follows: we train the model in the training set and use validation set to tune parameters; we compare our model performance over the validation set for model selection. Finally, we combine training and validation sets to train the

model, which will help increase sample size and reduce prediction errors, and evaluate the model performance over the test set.

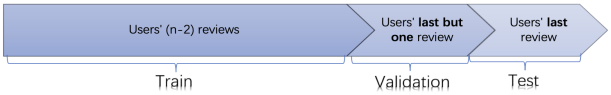

Figure 3: Train-validation-test split

4.3 Collaborative Filtering via Matrix Factorization

4.3.1 Matrix Factorization Overview

We first discuss Matrix Factorization (MF) as a class of collaborative filtering algorithms.[4]

We can turn our matrix factorization approximation of a k-attribute user into math by letting a user u take the form of a k-dimensional vector x_u. Similarly, a business i can be k-dimensional vector y_i. User u's predicted rating for business i is the dot product of the two vectors.

$$\hat{r}_{ui} = x_u^T \cdot y_i$$

We choose to minimize the square of the difference between actual and predicted ratings in the sample. This produces a loss function of the form:

$$L = \sum_{u,i \in S} (r_{ui} - \hat{r}_{ui})^2 + \lambda_x \sum_u \|x_u\|^2 + \lambda_y \sum_i \|y_i\|^2$$

Two L2 regularization terms for user and business vectors are added to prevent overfitting. Solving this minimization problem allows us to estimate the users' ratings $\hat{r}_{ui}$. Based on this framework we introduce three models in the next sections.

4.3.2 Baseline Model: Bias Only

We first lay out our baseline model with bias only:

$$\hat{r}_{ui} = \mu + b_u + b_i;$$

$$L = \sum_{u,i \in S} (r_{ui} - \hat{r}_{ui})^2 + \lambda_1 \|b_u\|^2 + \lambda_2 \|b_i\|^2,$$

where we assume that the prediction is generated from a normal distribution; μ is estimated from the training data using Maximum Likelihood Estimation; b_u and b_i act as the bias term of each user and business respectively.[5].

4.3.3 ALS without Bias Correction

For ALS minimization, we hold one set of latent vectors constant (business vectors first), and take the derivative of the loss function with respect to the other set of vectors (the user vectors here). Next, we solve for the non-constant vectors (the user vectors), and then alternate by solving for the other set of latent vectors and holding the user vectors constant. We alternate back and forth and carry out this two-step dance until convergence.

Taking the derivative of the loss function defined in Section 4.3.1 with respect to x_u yields:

$$\frac{\alpha L}{\alpha x_u} = -2 \sum_i (r_{ui} - x_u^T y_i) y_i^T + 2\lambda_x x_u^T = 0$$

We denote Y $(m \times k)$ to represent all business row vectors vertically stacked on each other. The solution for x_u is: ${x_u}^T = r_u Y (Y^T Y + \lambda_x I)^{-1}$

Similarly, the solution for y_i is:

$$\frac{\alpha L}{\alpha y_i} = -2 \sum_i (r_{ui} - {y_i}^T x_u) x_u^T + 2\lambda_y y_i^T = 0$$

$$y_i^T = r_i X (X^T X + \lambda_y I)^{-1}$$

The hyper-parameters include the number of the latent factors (k); regularization for the user latent vectors; regularization for the business latent vectors; the number of training iterations. We tune the

[4]Please find Rosenthal (2016) for a detailed overview of explicit matrix factorization.

[5]See Microsoft's best practice for recommender systems. (Argyriou et al., 2020)

parameters over the validation set. Figure 4 plots latent factors against MSE with an optimal value of 25. Similarly we set regularization for user latent vectors as 0.1; regularization for business latent vectors as 0.1; the training iterations as 10.

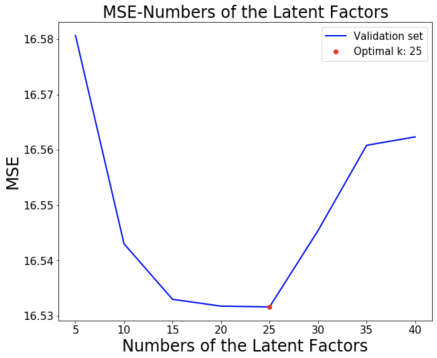

Figure 4: Latent factors in ALS model

From Figure 4 we find the lowest MSE over the validation set occurs at 16.43. This is quite high compared with the baseline model, which means that omission of bias leads to significant underperformance.

4.3.4 SGD with Bias Correction

With SGD, we solve for the unknowns in the loss functions simultaneously. The "stochastic" aspect of the algorithm involves taking the derivative and updating feature weights one individual sample at the same time.

Importantly, we recognize that each user and business have a bias term associated with them. The rationale is that some users may tend to always rate restaurants higher than others, and the same applies the business ratings as well. We also include a global bias term that applies to every rating unconditionally. The rating prediction model becomes:

$$\hat{r}_{ui} = \mu + b_u + b_i + x_u^T y_i$$

Therefore, the loss function can be written as:

$$L = \sum_{u,i}(r_{ui} - (\mu + b_u + b_i + x_u^T y_i))^2 + \lambda_{xb} \sum_u \|b_u\|^2 + \lambda_{yb} \sum_i \|b_i\|^2 + \lambda_{xf} \sum_u \|x_u\|^2 + \lambda_{yf} \sum_i \|y_i\|^2$$

We take the derivative with respect to b_u, b_i, x_u and y_u, and apply SGD to update feature weights recursively.

Similar to the previous section, we tune the hyper-parameters, including latent factors, the learning rate and regularization for user and business latent vectors and training iterations, over the validation set. The optimal parameters are set as 35, 0.01, 0.1, 0.1 and 10, respectively.

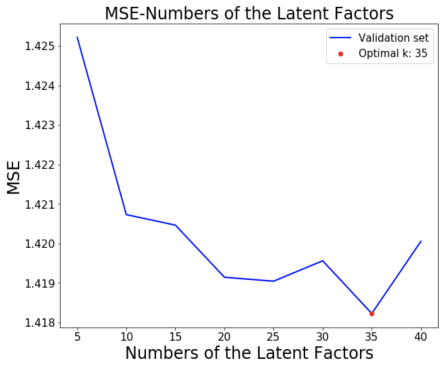

Figure 5: Latent factors in SGD model

Figure 5 plots the number of latent factors against MSEs. The best-performing model yields a MSE of 1.416 on the validation set.

4.4 Content-Based Model

Content based (CB) recommenders identify the content or attributes of items that a user prefers, and recommend similar items based on item features.[6]

[6]See Sharma (2019) for an illustrative overview.

The CB model does not need any data about other users, since the recommendations are specific to this user. This alleviates the cold start issue that many Yelp users have very few ratings, and makes the model easier to scale up. Our content-based model proceeds in three steps:

First, we build a content analyzer for restaurant data preprocessing and feature extraction. We extract the restaurants' attributes from Yelp, and convert them into vector space representation. Two sources of features are considered: the first with explicit restaurant attributes (RA), and the second with restaurant review documents (NLP). We also apply truncated singular value decomposition (truncated SVD) to reduce the feature dimension.

Second, we analyze the content of restaurants and construct a restaurant similarity matrix. Cosine similarity is applied on each pair of restaurants, x and y:

$$sim(x, y) = \frac{< x, y >}{||x|| \cdot ||y||}$$

Lastly, we build user profiles by reading the user's rating history. In the filtering step we filter the user's top k most similar restaurants to the target restaurant and make a prediction based on user's average ratings on these items.

Restaurant attribute (RA) similarity: From Yelp's business data we extract a wide range of restaurant attributes in order to depict a restaurant's complete characteristics. The features we consider include restaurants' locations, average ratings, dining policies (e.g. outdoor seating), ambience (romantic, casual, etc.), and others (e.g. parking, reservations allowed etc.). Numerical features are standardized to zero mean and unity variance, while the remaining features are converted to dummy variables. We reduce the features' dimensions by applying truncated SVD using the first 10 singular values which explain over 90% of total variations of restaurant attributes.

We tune k, the number of most similar restaurants within the user's rating profiles to the target restaurant. We estimate the user's rating predictions with the training set, and calculate MSEs over the validation set for which user ratings are known. The resulting optimal k is 15 (Figure 6).

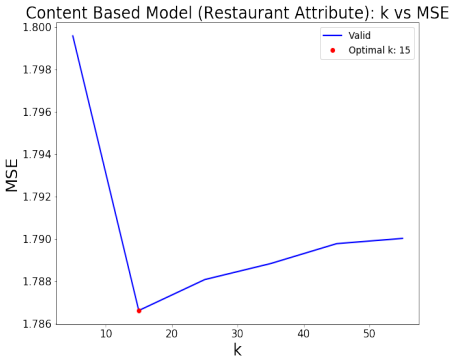

Figure 6: k neighbors in CB model (RA)

Restaurant reviews (Tf-idf) similarity: Next, we explore the restaurant reviews data using Natural Language Processing to construct restaurants' vector space representation. To avoid data leakage, only reviews in the training set are processed.

We use the bag-of-words approach to process reviews texts and perform stop words removal, stemming, lemmatization and Tf-idf vectorization. This results in a large-dimension (1750) vector space for restaurants. We then apply truncated SVD to

reduce the dimension to 20 that explains approximately 60% of variations in the reviews vector space, which is used to estimate the restaurant similarity matrix. The optimal parameter k is 45. (Figure 7)

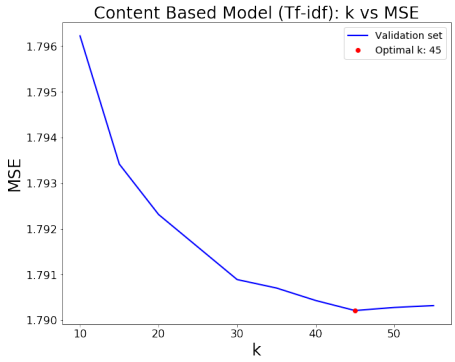

Figure 7: k neighbors in CB model (Tf-idf)

Restaurant reviews (LSI) similarity: In addition to Tf-idf vectorization, we also apply Latent Semantic Indexing (LSI) to restaurant reviews in order to estimate restaurant similarity. LSI is a count-based topic model to generate semantic embeddings by producing a set of latent topics related to the documents.

Different from the previous Tf-idf vectorization model where the similarity matrix is calculated over the entire restaurants' feature space, we perform LSI only on reviews of restaurants that the user has rated and the target restaurant. Thus, the LSI similarity is user profile-dependent that should better capture the user preference.

Figure 8 plots the MSEs against the number of latent topics for LSI model and the optimal value is 150. We find there is hardly any improvement by moving from bag-of-words to word embeddings.

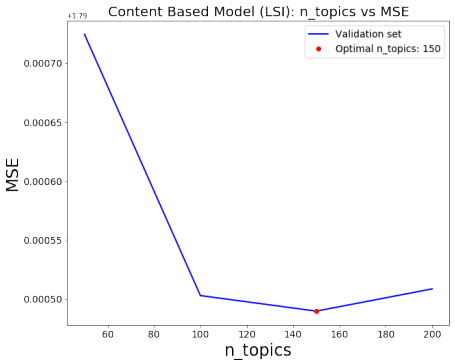

Figure 8: n topics in LSI model

4.5 Ensemble

So far we have built three matrix factorization models and three content-based models. We formulate our final prediction as a weighted average ensemble of the six models. Concretely, we set the weight of each model based on the inverse of its MSE score. By comparing the predictions with actual ratings of our validation set, we find the MSE of the ensemble model is 1.4806.

4.6 Hybrid Matrix Factorization Model

4.6.1 Model Background

Kula (2015) proposed a hybrid matrix factorization model to address the cord-start issue by combining the content-based model and the collaborative filtering approaches, which is called LightFM. The general idea is to combine the metadata features with the interaction feedback. We adopt the idea and apply the model on our Yelp restaurant recommender system.

The LightFM model operates based on binary feedback, so the restaurant ratings (scaled from 1 to 5) are normalized into two groups, S^+ and S^-. For each user and restaurant feature, their embeddings

are e_f^U and e_f^R respectively. Furthermore, each feature also has a scalar bias term (b_U^f for user and b_R^f for restaurant features). The embedding (latent representation) of user u and restaurant r are the sum of its respective features' latent vectors:

$$q_u = \sum_{j \in f_u} e_j^U; p_r = \sum_{j \in f_r} e_j^R$$

Similarly the biases for user u and restaurant i are the sum of its respective bias vectors:

$$b_u = \sum_{j \in f_u} b_j^U; b_r = \sum_{j \in f_r} b_j^R$$

The prediction for user u and restaurant r can then be modelled as:

$$\hat{rating}_{ur} = \sigma(q_u \cdot p_r + b_u + b_r),$$

where $\sigma(\cdot)$ represents the sigmoid function. We can then maximize the likelihood using SGD:

$$L = \prod_{(u,r) \in S+} \hat{rating}_{ur} \times \prod_{(u,r) \in S-} 1 - \hat{rating}_{ur}$$

Even though the LightFM can generate predictions, the raw prediction values are not interpretable and are only for ranking purpose.

4.6.2 Tuning Parameter

We tune three parameters for LightFM, learning rate, l2 regularization term and number of latent factors. We test the learning rate among [0.01,0.1,0.2,0.25,0.3,0.4]. The l2 regularization term prevents the model to overfit, and we test among [1e-7,1e-6,1e-5]. For the number of latent factors, We test among [10,20,40,60]. Using precision@k to select the model, the best score is 0.0101 when learning rate is 0.25, regularization is 1e-5 and number of latent factors is 10.

The LightFM-CF model with only user-restaurant interactions returns the precision@k score of 0.00041. The hybrid model with user/ restaurant features is seen significant improvement with a precision@k score of 0.0101.

5 Discussion

5.1 Result Summary and Interpretation

Table 1 shows the final results of our models on the validation set.

Model	MSE	MAE	Precision@1
MF-BiasOnly	1.4369	0.9585	0.000462
MF-ALS	16.4328	3.8370	0.004308
MF-SGD	1.4178	0.9478	0.000308
CB-RA	1.7866	1.0084	0.000718
CB-TFIDF	1.7903	1.0100	0.000564
CB-LSI	1.7904	1.0100	-
Ensemble	1.4806	0.9478	-
LightFM-CF Only	-	-	0.00041
LightFM-with Feature	-	-	0.010103

Table 1: Final Result Summary

Based on MSE and MAE, MF-SGD outperforms the baseline model MF-BiasOnly, since it captures both the user/item biases and the latent preference features in the rating matrix. MF-ALS significantly underperforms other models as it does not correct for bias, which highlights the importance of controlling for rating biases.

Content-based models underperform the baseline. Although they are more adaptive to the cold start issue, this similarity-based approach fails to deliver MSEs comparable to model-based methods (MF). The ensemble model performs just in line with the baseline MF-BiasOnly, but it strikes a

good balance between performance and mitigating the cold-start issue, unlike the CF methods.

However, using only MSEs to evaluate recommendation models is limited. Predicting ratings is not a direct way for recommendations. Whether the model can correctly predict users' ranking orders of preference can be more relevant. Therefore, precision is introduced to evaluate the ranking performance. As for precision@1, a correct order of items is more important than precisely predicting ratings. We can see that most of our MF and CB models have higher precision@1 scores than the baseline with the exception of MF-SGD.[7]

While other models generate explicit rating predictions out of which the ranking order is estimated, the direct goal of LightFM is ranking and its loss function WARP aims to maximize the rank of positive examples. In addition, the categories of restaurants are incorporated to improve the recommendation. LightFM-withFeature performs the best among all candidate models in terms of precision@1, and is chosen to be the final model.

5.2 Model Evaluation

We train the LightFM-withFeature model using combined train and validation sets. The resulting model achieves the precision 0.008769 on the test set. LightFM-withFeature incorporates the advantages of collaborative filtering and content-based models. In the cold start scenario, the model performs well due to its hybrid nature. Also, LightFM-withFeature is very efficient to train.

5.3 Deployment

There are several issues to consider in deployment of the Yelp restaurant recommender system.

The first is the cold start issue. The traditional collaborative filtering approach only applies to a warm start since it relies on enough user-restaurant interactions. The content-based models are explicitly designed for handling the cold-start problem. Our final model, the LightFM approach is also robust as it combines both collaborative filtering and content-based methods. We recommend using the hybrid model for the actual deployment. More advanced hybrid approaches often include deep learning models such as DeepFM and Wide and Deep Learning (Guo et al., 2018).

Secondly, we would like the recommender system to go online and perform real-time recommendations. That is, whenever a user has left feedback, explicit or implicit, the system updates the corresponding user latent features and restaurant features respectively and prompts updated recommendations.[8] To perform real-time predictions timely, we prefer less resource-consuming models and use distributed computations. For example, we can deploy the Spark cluster that can perform parallel computation for matrix factorization. In general, the ALS and SGD based models can scale well because the computation can run in parallel. Content-based models can also scale up well since

[7] For CB-LSI model we skip the full user-restaurant ratings prediction due to computing power limits, and leave the precision@1 estimation for future studies.

[8] Implicit feedback like click-through may be more ubiquitous that can be also used for recommenders. We didn't include the implicit feedback since it is not available on Yelp.

they don't require user-item interactions.

Lastly, it is advisable to perform the A/B test in a real-world setting before launching. We would randomly select a group of users to show the Yelp website with the updated model and the rest of the population with the original website, with other factors properly controlled. One can then evaluate the statistical significance of the evaluation metrics such as difference in click through rates between different groups. The A/B test can demonstrate the impact of the new recommender engine on user experience and quantify the commercial values.

5.4 Ethical Considerations

Milano et al. (2020) proposes privacy risks may arise when data include private information. However, the datasets we use are encrypted and only contain users' explicit behaviors like ratings and reviews. Implicit behaviors like click-through data, and personal information like location and age, are not included. In practice, one should always exercise extra caution in using any user-sensitive information.

Since recommendation can increase public exposure of the business, some restaurants may pay people to write positive reviews and give high ratings. Then other small restaurants who cannot afford such payment are less likely to be recommended. Also, users will be influenced to create a feedback loop. Thus, an important next step of building a robust recommendation system is to filter out those fake reviews, and aim to provide fair opportunities to all businesses.

References

Andreas Argyriou, Miguel González-Fierro, and Le Zhang. 2020. Microsoft recommenders: Best practices for production-ready recommendation systems. In *Companion Proceedings of the Web Conference 2020*, WWW '20, page 50–51, New York, NY, USA. Association for Computing Machinery.

Huifeng Guo, Ruiming Tang, Yunming Ye, Zhenguo Li, Xiuqiang He, and Zhenhua Dong. 2018. Deepfm: An end-to-end wide & deep learning framework for CTR prediction. *CoRR*, abs/1804.04950.

Yehuda Koren, Robert Bell, and Chris Volinsky. 2009. Matrix factorization techniques for recommender systems.

Maciej Kula. 2015. Metadata embeddings for user and item cold-start recommendations.

Silvia Milano, Mariarosaria Taddeo, and Luciano Floridi. 2020. Recommender systems and their ethical challenges. *AI & SOCIETY*, pages 1–11.

Ethan Rosenthal. 2016. Explicit matrix factorization: Als, sgd, and all that jazz.

Nikita Sharma. 2019. Recommender systems with python: Content-based filtering.

A Appendix: Code Repository

The code for this project can be found on Github: (https://github.com/Yelp-Recommender-System/FancyYelpers.git)

B Appendix: Figures

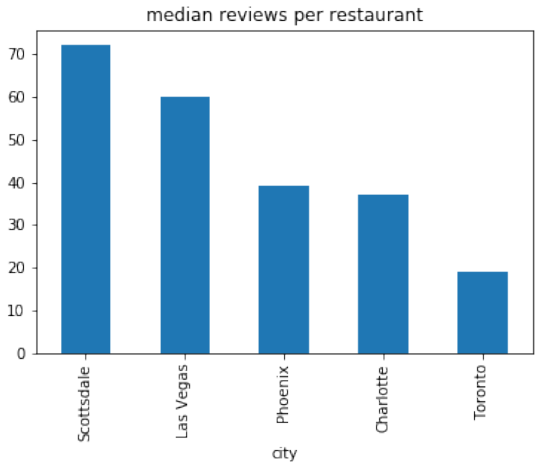

Figure 9: Median reviews per restaurant

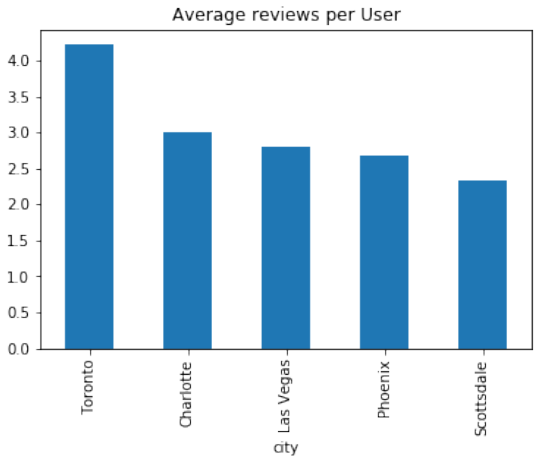

Figure 10: Average reviews per user

Figure 11: Reviews word cloud