Yelp Business Recommendations

Authors: Bangwei Zhou(bz2280), James Lee(jl5241), Ujjwal Peshin(up2138), Zhongling Jiang(zj2249)

A. Abstract

We compare the merits of six different recommendation algorithms side by side for making business recommendations to yelp users.

1. Biase Baseline
2. Matrix Factorization (Baseline)
3. Collective Matrix Factorization
4. Field-Aware Factorization Machines
5. Neural Networks
6. Content Based (for cold-start)

Comparing the results of the models against our baseline shows that Collective Matrix Factorization Algorithm, DeepFM Algorithm, and Wide and Deep Algorithms outperform the baseline model. The strength of deep learning approaches were showcased as we increased the size of the dataset, beating the baselines initially by .02 RMSE in 20% subsampled data but later beating them by .04 RMSE in 100% of the data. Another key advantage of the deep learning approaches were the short training time. Using 100% of the dataset, they finished training in just 800 seconds on local environment while the CMF took an equivalent amount of time for dataset subsampled down to 50% size. Additionally, the weaknesses of the baseline models were exposed when metrics that measure user experiences such as ranking, coverage, and novelty were investigated.

While our predictive algorithms didn't provide drastic improvements in accuracy, our top deep learning methods providing about 3% improvement versus the baseline, given the fast training time as well as its ability to achieve superior measures on metrics that affect positive user experiences such as coverage, ranking, and novelty metrics, there is a strong reason to believe that deep learning models can provide value to Yelp users in a meaningful manner. In this report, we discuss the merits of various algorithms we explored and finally propose a complete recommendation system, that employs an ensemble of data retrieval system based on user location, cold-start recommendations using the bias-based model as well as DeepFM for general recommendation.

B. Business Case and Objective

Our goal through this exercise is to figure out the best methodology for recommending restaurants to our users. This will be largely measured by using RMSE, which measures our forecast's error relative to the ground truth as well as ranking metrics. Additionally, we want to

measure metrics that can help us assess the sanity of our recommendation and experiences of the users. We measure these using coverage and novelty metrics to ensure that there is enough diversity in our recommendations and that our model isn't resorting to recommending the most popular restaurants as opposed to creating a truly personalized experience.

There are additional considerations such as model computation time, which affects long term overhead of maintenance of the product, and model comprehension. These are important considerations for the business as inefficient solutions will add technology debt and burden to the company's systems. Also, employing models that are well understood can help the team in getting buy-in from stakeholders. Therefore, these factors will be critically analyzed when assessing the merits of our algorithms.

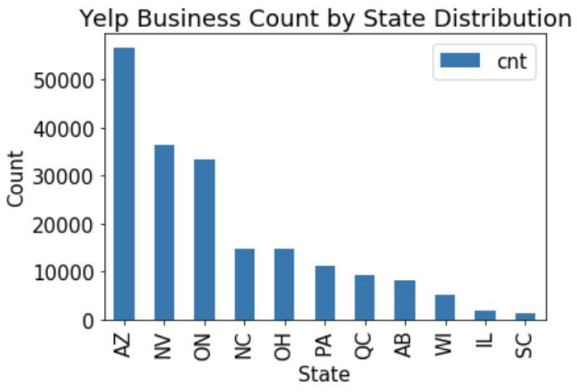
C. Data

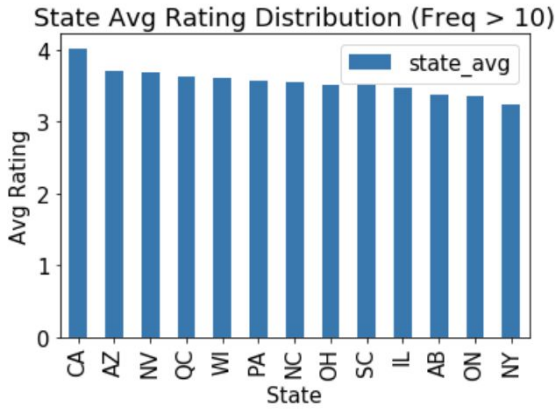
We use the publicly available yelp dataset which consists roughly of 7 million reviews from 1.6 million users across 190 thousand businesses in 10 metropolitan areas. We are additionally provided with business and user metadata as well as check-in information, tips and photos provided by the reviewers.

D. Data Exploration and Visualizations

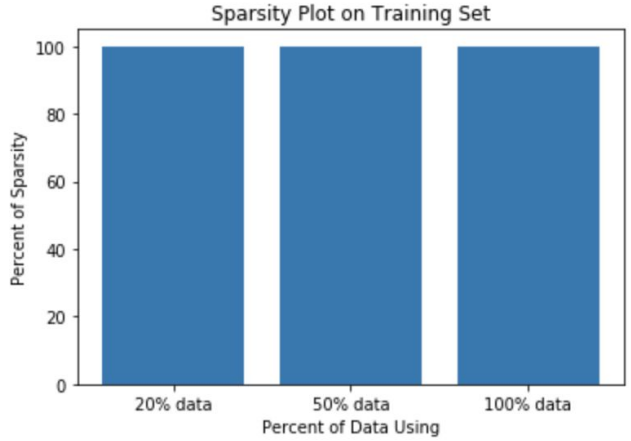
The data visualization here is mainly to help understand the state condition in our data set. We realized that some states have significantly more rating counts than others (such as AZ, NV and ON are really high on rating counts). To notice here, we filter out the states that have rating counts below 100, which is why in the graph, NY and CA are not included. This might suggest that our data set isn't very comprehensive and is heavily concentrated on those states.

With the second graph, we did some analysis on the average state rating as we will use it for one of our collective matrix factorization approaches, in which we are including CA and NY meaning we are making the threshold any state with counts above 10. Here, we do see that, though overall states are having quite similar average state ratings, some states are better (such as CA) and some states are worse (such as NY).





In addition, we also did a sparsity plot on our training sets (20%, 50% and 100%). They are all highly sparse and around 99.99%.



E. Subsampling and Train-Test Split

E.1. FIltering, Undersampling and Dataset Size Considerations

The idea behind undersampling is to develop a smaller dataset that is representable, or partially representative of the entire dataset, and expedites the model development cycle. We utilize 20% dataset size generally across the board for fast development iteration and for computation intensive tasks such as hyperparameter tuning. However, it is critical to analyze model performance across dataset sizes since the models' capability to handle large dataset is an important consideration if it were to be utilized in production. Therefore we investigate two other dataset sizes as well, 50% and 100% of the data size. Whereas 20% of the dataset size is used generally, larger datasets are employed to analyze certain models' capability to handle large data as well as measure how their performance changes with larger datasets. Finally, we filter our dataset to only active users who have at least 5 reviews since there needs to be at least some data about the user for the model to perform. If a user does not have at least 5 reviews, we will be building out recommendations using cold-start methods.

E.2. Train-Test Split

Train-Test split is done in a very straightforward way. We take all the users that made through our filters as described above. Then we simply take the last two reviews of the users. The most recent review becomes our test set and the other second most recent review becomes our validation set.

F. Methods

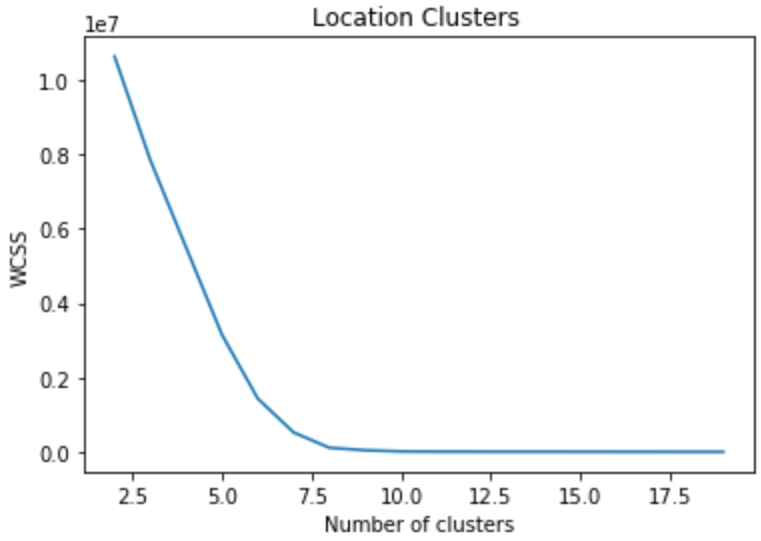
F.1. Feature Engineering

Below are some features that we engineered and were considered in select models. In particular, flexibility of Field-Aware Factorization Machines allowed for the usage of all the features discussed below.

F.1.a. Location Clusters

In addition to using state and city as features, we also used latitude and longitude to create more insightful location data. The idea is that there can be review biases related to those locations. For example, a business in New York City may face more critical food reviewers who are used to being exposed to diverse selection of delicious food whereas this may not be true for smaller areas.

In particular, we used K-means model to find clusters of business locations using latitude and longitude. K-meansmodel simply tries to minimize the euclidean distances between the clusters and data points assigned to those clusters. The number of clusters were chosen using the elbow method which looks at the loss as a function of the number of clusters. Here, we choose 10 clusters as a suitable number that reduces the loss well without overfitting on the data. Cluster assignments are fed into models as categorical features.



F.1.b. Text Analysis using Dimensionality Reduction

Additionally, we leverage the review text data to analyze review writing patterns of the users. Idea is that speaking patterns of the users may reveal attributes of the type of reviewer or person they are. Perhaps, our algorithm will be able to identify critical food reviewers based on the types of words they use to describe the businesses, and some businesses may receive positive reviews from such people as opposed to others.

In order to generate text related features, we first conduct TFIDF on the review texts. TFIDF essentially tallies the number of times a certain word shows up in a review and discounts the tallies based on how often they occur across other documents. Formally, they are defined as such,

$$tfidf(t, d, D) = tf(t, d) \cdot idf(t, D)$$

$$tf(t, d) = log(1 + freq(t, d))$$

$$idf(t, D) = log(\frac{N}{count(d \in D : t \in d)})$$

Resulting features are extremely sparse and wide set of columns, each one representing a particular word. It's infeasible to run our algorithms on these set of features. Therefore, we conduct dimensionality reduction using Truncated Singular Value Decomposition, and feed the resulting components into our models. TSVD essentially compresses the data into a smaller dimension.

F.1.c. Graph Analysis on Friend Networks

Last special feature that we are using besides the ones provided by the Yelp dataset uses connected components of friend networks. We are provided the friends of the reviewers in the user dataset. Given this, we create a graph database of all the connections. We can then identify which of the users are connected via a social network. Idea is that users that are connected will have similar preferences which can be a result of demographics. For example, international users from the same demographics may be connected and they may prefer or have more critical opinions of their native cuisine. We find these connected components using the networkx package, and then assign users a category based on which connected component they belong to. The resulting category is used as an input to the models.

F.2. Predictive Methods

F.2.a.1 Baseline: Bias-Based Model

The baseline model assumes a user has some bias towards her ratings in businesses, and hence rate businesses higher or lower in general. Also it assumes the businesses each have a higher or lower propensity to appeal to people in general. Also, the data has some underlying bias, and hence, each bias is modelled using the following equation,

$$\widehat{r}_{ui} = \mu + b_{\mathrm{u}} + b_{\mathrm{i}}$$

Here, μ is the average rating bias, and the parameters, b_{u} and b_{i} indicate the user and item bias respectively. For example, let us say that we want to find how a user *ABC*, might rate *'SHAKE SHACK BURGERS*'. Let us say that the average rating of all the restaurants in the data is 3.4,

and in general, *'SHAKE SHACK BURGERS'* is a highly rated restaurant, so it is rated 0.5 higher than the average. On the other hand, *'ABC'* is a selective user and generally rates restaurants 0.4 lower than the average. So, the baseline estimate for the user *'ABC'* for the restaurant *'SHAKE SHACK BURGERS'* would be 3.4 + 0.5 - 0.4 = 3.5. The baseline function implemented by us takes b_{u} to be the average rating for that user, and b_{i} to be the average rating for that business.

F.2.a.2 Baseline: Matrix Factorization using Surprise

Matrix factorization is a class of collaborative filtering algorithms. The general idea behind matrix factorization is that there can exist a lower dimensional latent space of features in which users and items can be represented such that the interaction between them can be obtained by simply dot producing the corresponding dense vectors in that space. In short, it decomposes a m*n user-item interaction matrix into two m*k and k*n matrices, sharing a joint latent vector space, where m represents the number of users, and n represents the number of items. In terms of its outcome, we are likely to observe that close users in terms of preferences as well as close items in terms of characteristics can have close representations in the latent space.

The mathematical overview is as follows:

Given a n*m matrix, such that $M \approx XY^T$. X is the user matrix where rows represent the n users and Y is the item matrix where rows represent the m items. We want to search for the dot product of matrices X and Y that best approximate the existing interactions; i.e., we want to find X and Y that minimize the "rating reconstruction error":

$$(X, Y) =_{X,Y\ (i,j)\in E}[(X_i)(Y_j)^T - M_{ij}]^2$$

Adding a regularization term, we can also get:

$$(X, Y) =_{X,Y\ (i,j)\in E}[(X_i)(Y_j)^T - M_{ij}]^2 + \lambda/2(\sum_{i,k}(X_{ik})^2 + \sum_{j,k}(Y_{jk})^2)$$

In general, we obtain the matrices X and Y following a gradient descent optimization process. And once the matrices are obtained, we can predict the ratings simply by multiplying the user vector by any item vector.

In this Yelp Rating Challenge, we used the python surprise package to implement MF. The MF algorithm there uses the SVD approach, which is essentially

$$P_{m*n} = U_{m*m} \sum_{m*n} V_{n*n}.$$

There, the prediction is

$$\hat{(r_{ui})} = \mu + b_u + b_i + (q_i)^T p_u,$$

and the regularized squared error that needs to be minimized is

$$\sum_{r_{ui} \in R_{train}} (r_{ui} - \hat{(r_{ui})})^2 + \lambda(b_i^2 + b_u^2 + ||q_i||^2 + ||p_u||^2).$$

As the way the package is designed, we tuned on n_epochs, lr_all and leg_all to get an optimal hyperparameter set, where n_epochs is the number of iterations of the SGD (stochastic gradient descent) procedure, lr_all is the learning rate for all parameters, and reg_all is the regularization term for all parameters.

F.2.b CMF

To best the baseline (which in our case, is Matrix Factorization), we first tried the Collective Matrix Factorization technique. This means that, in addition to the ratings, we also want to include additional features on either the items or the users, to see how the model perform. Before we discussed what additional features we included, we might say a few words on the method and its mathematical intuition.

Similar to Matrix Factorization, Collective Matrix Factorization also aims to decompose, but in this time, we are decomposing two matrices X and Y into three matrices U, V, and Z, such that $X \approx f(UV^T)$ and $Y \approx f(VZ^T)$. Here, X can be the same matrix we see in the case of Matrix Factorization, i.e., a simple user-item matrix filled by ratings. And matrix Y is the matrix of additional features. It can be on user or on item. And the features can be one-hot encoded (i.e., categorical) or numerical. For example, in our case, we both involved state location for businesses (which is a categorical feature) and the average rating (which is a numerical feature).

Collective Matrix Factorization also has a function to minimize, and we will explain the function through the lens of the python cmfrec package, which is the package that we used to implement Collective Matrix Factorization. The function to minimize is as follows:

$$argmin_{A,B,C,D,U_b,I_b} \|(X - \mu - U_b - I_b - AB^T)I_x\|^2 + \|U - AC^T\|^2 + \|I - BD^T\|^2 + \lambda(\|A\|^2 + \|B\|^2 + \|C\|^2 + \|D\|^2 + \|U_b\|^2 + \|I_b\|^2)$$

Where X is the ratings matrix, I is the item-attribute matrix, and U is the user-attribute matrix, and A,B,C,D are lower-dimensional matrices. $|X|$, $|I|$, $|U|$ are the number of non-missing entries in each matrix. And $Ubias_u$ and $Ibias_i$ are user and item biases. Thus, for such a reason, when tuning, we tune w_main and w_item when we are including additional item features and we tune w_main and w_user when we are including additional user features.

Now, with CMF, we implemented three approaches in total- state average rating, state location, and user average rating. The idea is as follows:

We first included state average rating as an item additional feature. Since our objective here is to predict the last rating for each active user, we wanted to see if location can be a useful means to bring closer the predicted ratings to the actual ones. To better utilize the location feature though, we came up with two approaches. One is to calculate the state average rating, and the other is to use state location as a categorical feature. With the state average rating, we wanted to observe if some states can have a higher average rating than others. This means that either the restaurants in that state are significantly better or the users in that state are more lenient and friendly. From EDA alone, the average rating seems to make some sense, as we observed that California has the highest state average rating (though with much fewer data observations) and New York has the lowest (same, with much fewer data observations than AZ and NV). This might suggest that, even though New York (especially manhattan area) is known as a food hub, the yelp users here can be a little picky and critical, whereas the users in CA can be more friendly. By all means though, we wanted to see if this "secondary" information that we generated for ourselves can help better predict the result.

With the second approach on the location feature, we simply fed in the one-hot encoded columns. This time, we just want to see if the location itself, as a categorical feature, can bring us any closer to the actual value.

Lastly, we also did a user average rating for the user info. This is similar to the state average rating approach, but we just wanted to see if by directly analyzing the users' behaviors, that normally how critical they are with the restaurants, can help us better understand them and have a better predicted result than other approaches. To notice here though, we didn't use the given average rating, since that would include the last rating that we were trying to predict; thus, we had to hand-calculated the average rating again on the training set. We also wanted to include the yelp spending time, that is the time they been using yelp and see if this can help us better predict the result. But due to the time constraint, we are not able to implement.

F.2.c. Field-Aware Factorization Machines

Model that is commonly used in prediction problem is logistic regression with cross-features. When all the cross-features are added, we have what we called a Poly2 Model.

$$\phi(\mathbf{w}, \mathbf{x}) = \sum_{j_1, j_2 \in C_2} w_{j_1, j_2} x_{j_1} x_{j_2}, ^4$$

Where you have a weight corresponding to each features you are using as well as each pairs of features including the second degree polynomial features. FFM is similar to such a model, but instead of having a single notion of a particular variable, you learn separate variables of the

same variable for each different context. For example, you would have a variable user_1 that is interacting with a particular business_1. Additionally, FFM helps in working with sparse datasets by learning these relationships in latent space.

FFM predictions can be written as,

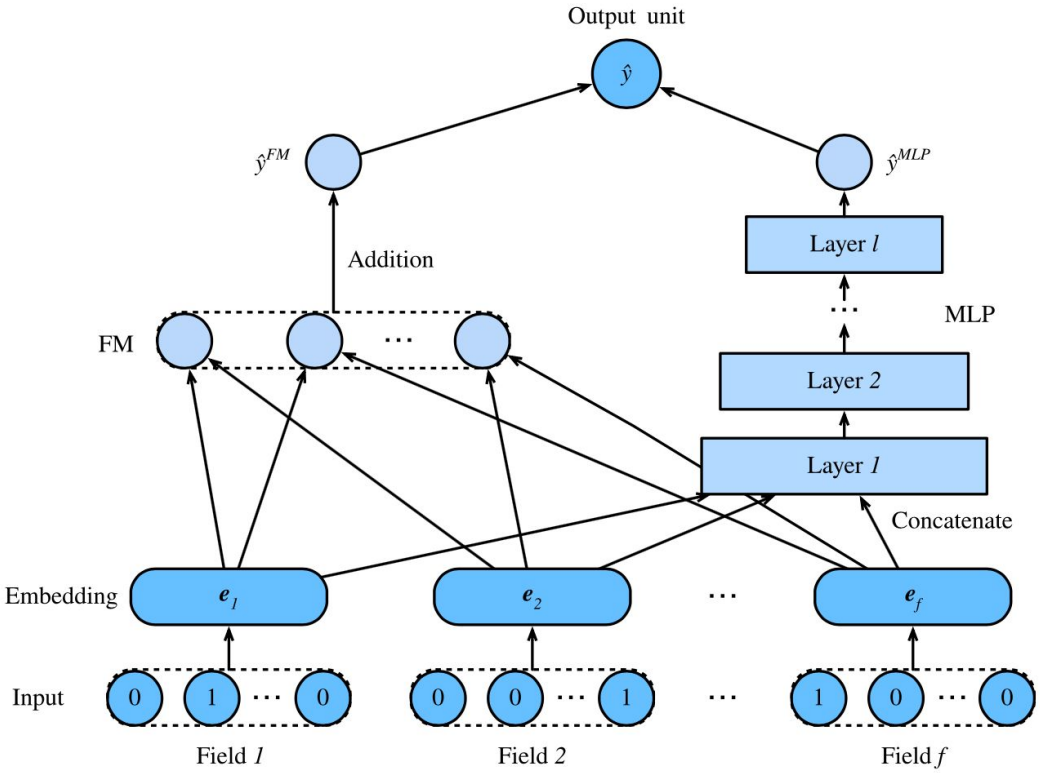
$$\sum_{f_1=1}^{F} \sum_{f_2=f_1+1}^{F} \mathbf{w}_{i_1} \cdot \mathbf{w}_{i_2},$$
$$\text{where } \; i_1 = \Phi(v_{f_1}, f_1, f_2), \; i_2 = \Phi(v_{f_2}, f_2, f_1),$$

Where F is the number of features, w is the weight matrix and w_i denotes the embedding of the -th entry. The mapping ϕ maps a value v of feature f_1 in the context of feature f_2 to an index from 1 to d.

These are important considerations because FFM is useful in learning feature interactions by converting them into a space where similar features are embedded near one another.

Additionally, given that we can practically input any meaningful categorical features into the model, FFM is a very flexible model that can leverage different types of features that many other algorithms cannot. This is especially helpful with our problem setting where we are given lots of meaningful and diverse data. In particular we found that including the location clusters discussed above in feature engineering section helps improve the accuracy of the FFM model.

F.2.d. DeepFM



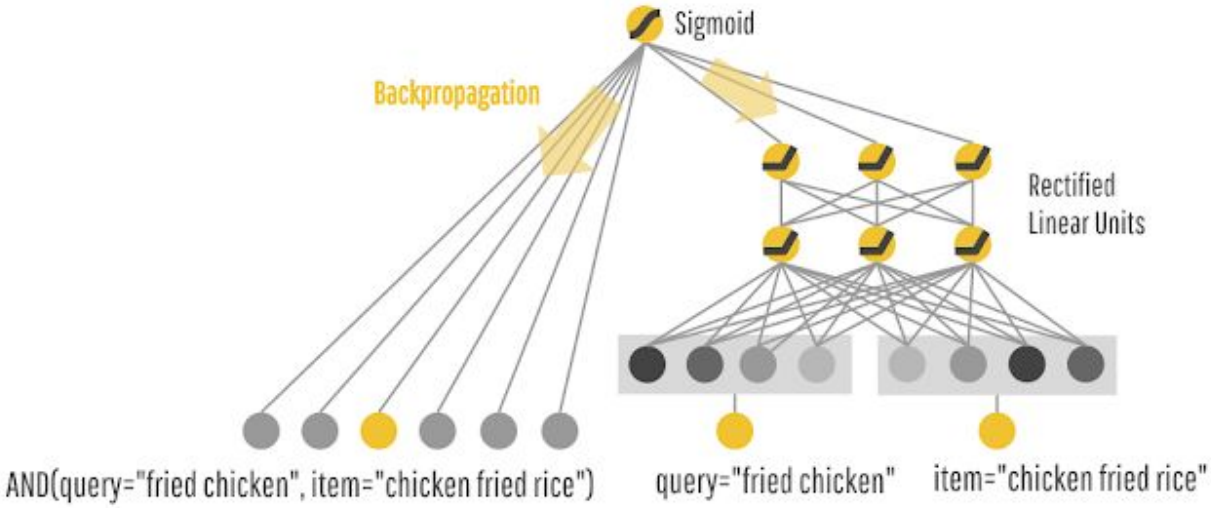
DeepFM Model Architecture

It is a model which integrates the feature representation learning of a neural network with factorization machines.

Adding nonlinear transformation layers to factorization machines gives it the capability to model both low-order feature combinations and high-order feature combinations. Moreover, non-linear inherent structures from inputs can also be captured with neural networks.

We have run a model which includes Business City, State as the field.

F.2.e. Wide and Deep Learning



The Wide part of the model tries to capture the co-occurrence of a query-item feature pair correlates with the target label. The Deep model generalizes the query-item interactions.

F.2.f Content Based Model

Approach 1

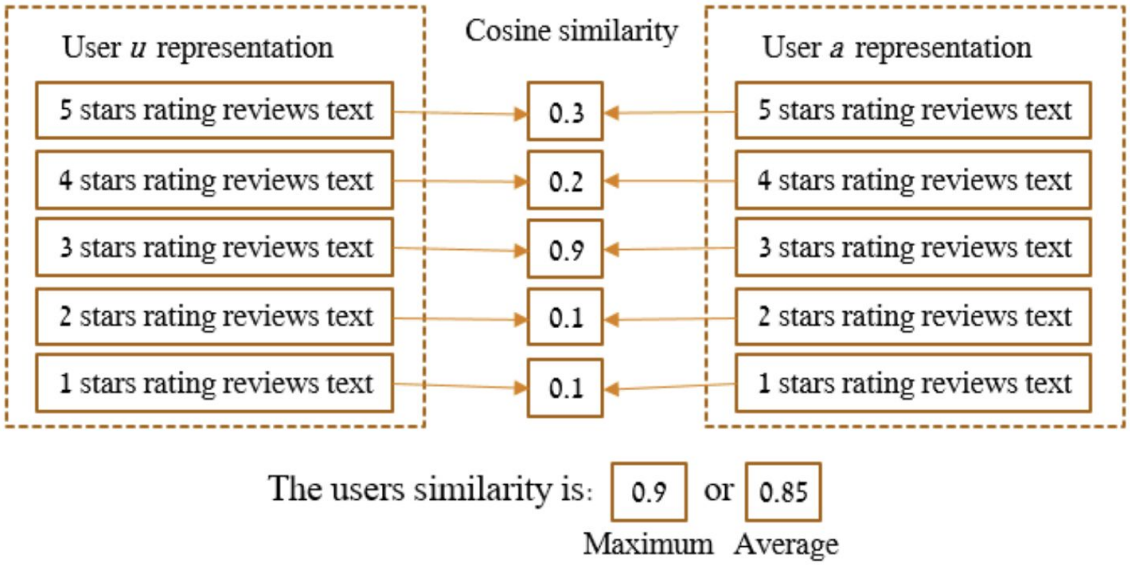
Implemented according to **CF-MCM (User-Item based) approach** in this following paper: https://arxiv.org/pdf/1607.00024.pdf

The idea of this model is to predict the rating of a review given by user U to restaurant R by comparing the review to the ones written by other users who have also reviewed restaurant _R_. The predicted rating is calculated by taking a weighted average of other users' ratings, where the weight is calculated based on similarity of reviews. Here are the details:

Algorithm 3 User-User Rating Prediction Process

procedure PREDICT($userID, itemID$)
  $usersReviews \leftarrow$ Dictionary of Dictionary {user, {rating, concatReviews}}
  for each review $RV_u \in usersReviews$ **do**
    $RV_u \leftarrow$ normalized text of RV_u
  end for
  for each user $u_i \in usersRatings$[itemID] **do**
    for each review $RV_u \in usersRevies$[userID] **do**
      for each possible rating value $r \in$ {1,2,3,4,5} **do**
        $userReviews_r \leftarrow$reviews whose rated r in $usersReviews[userID]$
        $u_iReviews_r \leftarrow$reviews whose rated r in $usersReviews[u_i]$
        $similarity_r \leftarrow$similarity value between $userReviews_r$ and $u_iReviews_r$
      end for
      $w_{u_i,u} = \max_r(similarity_r)$
    end for
  end for
  return $\bar{r_u} + \frac{\sum w_{u_i,u} * (r_{u_i,i} - \bar{r_{u_i}})}{\sum w_{u_i,u}}$
end procedure

Two users ratings are compared in the following manner: first aggregate all reviews user _u_ and user _a_ have written; then bucket them according to ratings given (from r = 1.0 to 5.0); finally find the max of all 5 pairs of similarity.



Approach 2

G. Evaluation Metrics

Before we discuss the metrics, we want to point out that given the nature of our problem, a lot of the metrics particularly around ranking, novelty, and coverage were improvised. For example, when building out the recommendation, we only consider the businesses that are in the same city as the user's last visited business. This is relevant because we don't want our users to get recommendations that are too far, and also, that is how we intend to serve users our model's recommendations. As such, we used our intuition to build out custom metrics that we think would do a good job in assessing the sanity of our recommendation system. These caveats are explained and pointed out along the way of this discussion section.

G.1. Regression Metrics

We present three different regression metrics that are measured but we primarily use the RMSE.

Root Mean Square Error (RMSE) calculates square rooted sum of square residuals of predictions. It measures numerical difference between all ground-truth ratings and actual ratings in test set.

Mean Absolute Error (MAE) calculates sum of absolute residuals. It measures numerical difference between all ground-truth ratings and actual ratings in test set, and more robust to outliers.

R-squared measures what percentage of variance in target that is explained by predictions. The higher the value, the better the predictions.

G.2. Ranking Metrics

We first make top 10 recommendations of restaurants for each user and see how relevant the rankings were. First we discuss the metrics then discuss a few caveats to the approach we took to measure them.

Inclusion of Last Review in Top 10 Recommendations: We train our model on the training set and make top 10 recommendations to the users. We then look at the test set and see if the user's latest review was included in that top 10 recommendations. We calculate the proportion of users who received such recommendations. In other words we have,

$$\frac{1}{n} * \sum_{j}^{n} \sum_{i \, \varepsilon \, j's \; top \; 10}^{10} Rel(i)$$

where a recommendation is relevant if it is the latest visited restaurant of the user and n is the total number of users measured.

Average Ranking of Latest Restaurant: We train our model on the training set and make a prediction on every single business and user combination. We then measure the average rankings on that prediction of the latest business that the user visited. In other words we have,

$$\frac{1}{n} * \sum_{j}^{n} \sum_{i}^{m} Rank(i)(1_{i=latest\ visited\ business\ of\ j})$$

Where the indicator function is one if the restaurant is the last visited restaurant of user j, n is the total number of users and m is the total number of businesses.

Some caveats of our approach is that instead of making a prediction on the entire business universe, we make predictions on the businesses that are in the same city as the business of the user's last review. This makes sense since it would be futile to recommend a restaurant in Los Angeles to a user who is in New York City. This also allows us to reduce the computation time of our recommendation, which is another key advantage we want to have when we serve our model to the users. Additionally, we measure the above two metrics on a subsample of users as opposed to all the users to save computation time, and we also only take subsample of ratings that were positive ratings since we want to measure how good our recommendations are.

G.3. Coverage

We measure the coverage of our recommendations by looking at the proportion of our recommendations that are distinct. In other words, we measure,

$$\frac{number\ of\ distinct\ businesses\ in\ all\ recommendations\ to\ the\ subsample}{number\ of\ all\ recommendations\ to\ the\ subsample}$$

High value from this metrics help us find assurance in that our model isn't recommending the same businesses to everyone.

G.4. Novelty

We measure the novelty of our recommendations by simply taking the proportion of businesses in our top ten recommendations that the user hasn't been to. This is crucial since we don't want our recommendations to be filled with restaurants that the user has already been to. We measure novelty as simply.

$$\frac{1}{n} * \sum_{j}^{n} \sum_{i}^{m} (1_{i=user\ j\ has\ not\ been\ to\ recommendation\ i})$$

G.5. Runtime

We measure the runtime of the model's training as well as its prediction time on validation set using Python's time library.

H. Hyperparameter Tuning

H.1. Matrix Factorization (Baseline)

Parameters (n_epochs, lr_all, reg_all)	**RMSE**
5, 0.002, 0.4	1.4159
5, 0.002, 0.5	1.4173
5, 0.002, 0.6	1.4173
5, 0.003, 0.4	1.4011
5, 0.003, 0.5	1.4016
5, 0.003, 0.6	1.4031
5, 0.005, 0.4	1.3798
5, 0.005, 0.5	1.3806
5, 0.005, 0.6	1.3821
7, 0.002, 0.4	1.4042
7, 0.002, 0.5	1.4047
7, 0.002, 0.6	1.4057
7, 0.003, 0.4	1.3872
7, 0.003, 0.5	1.3885
7, 0.003, 0.6	1.3900
7, 0.005, 0.4	1.3649
7, 0.005, 0.5	1.3665
7, 0.005, 0.6	1.3671
10, 0.002, 0.4	1.3894

10, 0.002, 0.5	1.3903
10, 0.002, 0.6	1.3917
10, 0.003, 0.4	1.3714
10, 0.003, 0.5	1.3734
10, 0.003, 0.6	1.3746
10, 0.005, 0.4	**1.3497**
10, 0.005, 0.5	1.3507
10, 0.005, 0.6	1.3527

H.2.a Collective Matrix Factorization (state average rating)

Parameters (w_main, w_item)	**RMSE**
0.5, 10.0	1.4091
0.5, 5.0	1.4093
0.5, 0.5	1.4096
5.0, 10.0	1.3255
5.0, 5.0	**1.3253**
5.0, 0.5	1.3255
10.0, 10.0	1.3287
10.0, 5.0	1.3251
10.0, 0.5	1.3288

H.2.b. Collective Matrix Factorization (state location)

Parameters (w_main, w_item)	**RMSE**

0.5, 10.0	1.4093
0.5, 5.0	1.4093
0.5, 0.5	1.4093
5.0, 10.0	**1.3243**
5.0, 5.0	1.3248
5.0, 0.5	1.3249
10.0, 10.0	1.3290
10.0, 5.0	1.3281
10.0, 0.5	1.3278

H.2.c. Collective Matrix Factorization (user average rating)

Parameters (w_main, w_user)	**RMSE**
0.5, 10.0	1.4091
0.5, 5.0	1.4093
0.5, 0.5	1.4096
5.0, 10.0	1.3255
5.0, 5.0	**1.3253**
5.0, 0.5	1.3255
10.0, 10.0	1.3287
10.0, 5.0	1.3285
10.0, 0.5	1.3288

H.3. DeepFM (City as Field)

Parameters (dropout, hidden units, embedding dims)	**RMSE**
0.1, (128, 128), 8	**1.3217**
0.1, (128, 128), 16	1.3445
0.1, (128, 128), 32	1.4504
0.1, (256, 256), 8	1.3218
0.1, (256, 256), 16	1.3455
0.1, (256, 256), 32	1.4599
0.1, (256, 128), 8	1.3218
0.1, (256, 128), 16	1.3469
0.1, (256, 128), 32	1.4702
0.3, (128, 128), 8	1.3223
0.3, (128, 128), 16	1.3488
0.3, (128, 128), 32	1.4510
0.3, (256, 256), 8	1.3225
0.3, (256, 256), 16	1.3496
0.3, (256, 256), 32	1.4610
0.3, (256, 128), 8	1.3223
0.3, (256, 128), 16	1.3501
0.3, (256, 128), 32	1.4710
0.5, (128, 128), 8	1.3231
0.5, (128, 128), 16	1.3501
0.5, (128, 128), 32	1.4657
0.5, (256, 256), 8	1.3243
0.5, (256, 256), 16	1.3518
0.5, (256, 256), 32	1.4676

0.5, (256, 128), 8	1.3260
0.5, (256, 128), 16	1.3610
0.5, (256, 128), 32	1.4799

H.4. Wide and Deep Learning (WDL)

Parameters (dropout, hidden units, embedding dims)	**RMSE**
0.1, (128, 128), 8	1.3299
0.1, (128, 128), 16	1.3473
0.1, (128, 128), 32	1.4681
0.1, (256, 256), 8	1.3285
0.1, (256, 256), 16	1.3452
0.1, (256, 256), 32	1.4667
0.1, (256, 128), 8	1.3243
0.1, (256, 128), 16	1.3441
0.1, (256, 128), 32	1.4604
0.3, (128, 128), 8	1.3222
0.3, (128, 128), 16	1.3415
0.3, (128, 128), 32	1.4586
0.3, (256, 256), 8	**1.3219**
0.3, (256, 256), 16	1.3358
0.3, (256, 256), 32	1.4514
0.3, (256, 128), 8	1.3256
0.3, (256, 128), 16	1.3394
0.3, (256, 128), 32	1.4573

0.5, (128, 128), 8	1.3271
0.5, (128, 128), 16	1.3428
0.5, (128, 128), 32	1.4597
0.5, (256, 256), 8	1.3269
0.5, (256, 256), 16	1.3440
0.5, (256, 256), 32	1.4604
0.5, (256, 128), 8	1.3280
0.5, (256, 128), 16	1.3472
0.5, (256, 128), 32	1.4627

I. Results

*** For graders and instructors: for all CMF methods, we ran the 100% data on google cloud, which then was successful. But for Bias Baseline and Matrix Factorization (which we didn't have time to run on google cloud), was not able to run on local and resulted in N/A memory overflow.**

I.1. Overall Comparison on 20% Subsampled Data

Regression Metrics

Model Name	**RMSE**	**R^2**	**MAE**
Bias Baseline	1.3545	0.1980	1.1270
Baseline (Matrix Factorization)	1.39	0.1449	1.1845
CMF (state average)	1.3495	.1867	1.1202
CMF (state)	1.3496	0.1865	1.1201

CMF (user average)	1.3493	1.869	1.1194
FFM	1.83	.17	1.10
DeepFM	1.337	0.202	1.090
WDL	1.335	0.205	1.094
Content Based(CF-MCM)	1.449	0.021	1.131

Ranking Metrics

Model Name	**Inclusion of Last Review in Top 10 Recommendations**	**Average Ranking of Last Positive Review**
Bias Baseline	12.8%	511
Baseline(Matrix Factorization)	9.3%	528
CMF (state average)	13%	484
CMF (state)	13%	473
CMF (user average)	13%	470
FFM	15%	493
DeepFM	15.66%	491
WDL	15.90%	464
Content Based(CF-MCM)	N/A(Computation Problem)	N/A(Computation Problem)

Novelty and Coverage Metrics

Model Name	**Coverage(% of unique recommendations)**	**Novelty(% of new restaurants in top 10 recommendations)**
Bias Baseline	20%	97.5%
Baseline(Matrix Factorization)	50%	97.1%
CMF (state average)	26%	96.9%

CMF (state)	26%	96.9%
CMF(user average)	26%	96.9%
FFM	25%	96.5%
DeepFM	24.80%	96.87%
WDL	23.69%	96.72%
Content Based(CF-MCM)	N/A(Computation Problem)	N/A(Computation Problem)

I.2. Comparison of Results on 50% Subsampled Data

Model Name	**RMSE**	**R^2**	**MAE**
Bias Baseline	1.3607	0.198	1.133
Baseline(Matrix Factorization)	1.379	0.175	1.163
CMF (state average)	1.372	0.178	1.152
CMF (state)	1.371	0.178	1.15
CMF (user average)	1.372	0.178	1.15
FFM	1.69	.23	1.04
DeepFM	1.343	0.213	1.097
WDL	1.345	0.211	1.116

Ranking Metrics

Model Name	**Inclusion of Last Review in Top 10 Recommendations**	**Average Ranking of Last Positive Review**
Bias Baseline	18%	440
Baseline(Matrix Factorization)	12.7%	422

CMF (state average)	18%	448
CMF (state)	18%	444
CMF (user average)	18%	440
FFM	14.9%	415
DeepFM	18.53%	379
WDL	19.82%	391

Novelty and Coverage Metrics

Model Name	**Coverage(% of unique recommendations)**	**Novelty(% of new restaurants in top 10 recommendations)**
Bias Baseline	20.5%	97.9%
Baseline(Matrix Factorization)	20%	98%
CMF (state average)	20.4%	97.9%
CMF (state)	20.5%	97.9%
CMF (user average)	20.5%	97.94%
FFM	27%	97.5%
DeepFM	22.61%	96.83%
WDL	22.40%	96.53%

I.3. Comparison of Results on 100% Subsampled Data (LIMITED RESULTS DUE TO COMPUTATION LIMITATIONS)

Regression Metrics

Model Name	**RMSE**	**R^2**	**MAE**
Bias Baseline	1.40	0.1507	1.1736
Baseline(Matrix	1.40	0.140	1.189

Factorization)			
CMF (state average)	1.3908	0.1612	1.1727
CMF (state)	1.3905	0.1615	1.1724
CMF (user average)	1.3907	0.1612	1.1726
FFM	1.65	.25	1.02
DeepFM	1.362	0.195	1.134
WDL	1.362	0.195	1.115

Ranking Metrics

Model Name	**Inclusion of Last Review in Top 10 Recommendations**	**Average Ranking of Last Positive Review**
Bias Baseline	1.4%	667
Baseline(Matrix Factorization)	N/A (Memory Overflow)	N/A (Memory Overflow)
CMF (state average)	19%	422
CMF (state)	19.2%	421
CMF (user average)	19.2%	421
FFM	11.2%	504
DeepFM	19.51%	432
WDL	19.67%	453

Novelty and Coverage Metrics

Model Name	**Coverage(% of unique recommendations)**	**Novelty(% of new restaurants in top 10 recommendations)**
Bias Baseline	N/A (Memory Overflow)	N/A (Memory Overflow)
Baseline(Matrix Factorization)	N/A (Memory Overflow)	N/A (Memory Overflow)
CMF (state average)	20%	97.6%

CMF (state)	20%	97.6%
CMF (user average)	20%	97.5%
FFM	26.5%	98%
DeepFM	20.65%	96.11%
WDL	20.33%	96.34%

I.4. Comparison of Runtime by Differing Data Size

Training Runtime:

Model Name	**20%**	**50%**	**100%**
Bias Baseline	2 seconds	9 seconds	14 seconds
Baseline(Matrix Factorization)	23 seconds	62 seconds	112 seconds
CMF (state average)	824 seconds	1569 seconds	1237 seconds (google cloud, can't run on local; thus, better compute)
CMF (state)	858 seconds	890 seconds	399 seconds (google cloud, can't run local; thus, better compute)
CMF (user average)	825 seconds	1636 seconds	1810 seconds (google cloud, can't run local)
FFM	80 seconds	419 seconds	784 seconds
DeepFM (on Tesla P100 GPU)	72 seconds	284 seconds (overfitting, so ran only for one epoch)	829 seconds
WDL (on Tesla P100 GPU)	71 seconds	285 seconds	834 seconds

We tried to solve the overfitting issue in both the models, however,even after regularization, dropout and simplifying the model, it did not get solved, so we ran with a model which runs for only one epoch.

Prediction Runtime:

All models under all data sets have prediction runtime under 1 second.

J. Result Discussion

J.1. Baselines

Baseline models perform very well in terms of accuracy, allowing only 3% RMSE improvement to our top deep learning models. In addition, its training times are lower than all of our other models, taking just 14 seconds to calculate some simple biases for 100% dataset size. However, we believe that there are key weaknesses to these approaches that merit the usage of some of our other methods suggested.

First, the bias baseline tremendously underperforms in terms of ranking metrics. Only 1.4% of its recommendations in top 10 recommendations contained our users' latest businesses or businesses in the test set for 100% dataset size. This was a drastic decrease compared to 20% where it scored 12.8%, which was also a relatively bad measure. This makes us believe that bias baseline drastically underperforms as the dataset size gets bigger, likely providing same recommendations to its users and therefore not providing a truly personalized experience to our users.

Matrix Factorization using surprise also largely underperformed in ranking metrics where only 12% of its top 10 recommendations contained the users' latest businesses while our other algorithms measured around 18% for 50% dataset size. So given these factors, we believe that the baseline models, while performing well in accuracies, come with major downfalls that need to be critically considered.

J.1. CMF

With Collective Matrix Factorization, and its three approaches, we observed no significant differences from them. Perhaps, this stems from two reasons. First, we might be able to do a better job at tuning. Due to the time constraint, we were not able to conduct a sophisticated and thorough tuning on all of the parameters that can be helpful. Instead, we chose some of the most important ones to save time. This might be one of the areas where the three approaches can differ from each other and meaning differences can arise. However, it is also possible that by tuning alone won't change a lot. This leads to our second hypothesis that potentially, we might need to include more critical and meaningful additional attributes of items or users, such as business categories or user years spent on yelp. However, we do see that, most cases (other than the 50% data, that CMF approaches failed to beat both baselines), our CMF

methods were able to beat the baseline RMSE scores. Thus, including an additional feature on items or users does seem to help better predict the ratings of the users. But there also seems to be limitations to this approach, and the run time is the most challenging one as we can observe from the run time analysis table, that CMF approaches require most of the times across all models. For this reason, CMF might not be the best choice, considering the not-so-impressive improvement on the RMSE scores as compared to the haunting increase on the run time. In other words, CMF really proposes this unbalanced and skewed accuracy-runtime trade-off. In general, with CMF, we would conclude to say that though it could be interesting to test the model with different additional attributes, it seems best to find other means that can perform similar tasks.

J.2. FFM

FFM model surprisingly underperformed across all dataset sizes for regression metrics. However, the fast training time was promising. It was the fastest algorithm for 100% dataset size. Additionally, it was suggesting a variety of businesses to users, achieving a highest coverage of 26.5% but consequently yielding an inferior ranking metric. There is a reason to believe that FFM is able to suggest a more personalized recommendations in exchange for relevance. Additionally, it is able to incorporate virtually any categorical variables at ease, making it a great option if it were to be paired with novel features. However, we believe that overall, it is difficult to make a case for FFM because of its relative poor performance at least for our problem at hand as we weren't able to find the right parameters and model set-ups that can demonstrate a strong performance.

J.3. Deep Learning Methods

The deep learning models outperformed all other models by a significant amount. The best RMSE is achieved by a deep learning model (RMSE = 1.362, 100% dataset). However, it does not seem to outperform other models when it comes to metrics that lead to a great user experience, like novelty and coverage. FFM does really well in those metrics. If user experience is a focus for the use case of the recommendation system, Deep learning models would not be our suggestion. There is another facet to training a deep learning model, which is the high cost that is incurred to maintain that model and to train that model, along with the sensitive hyperparameter tuning that needs to be carried out such that the model does not overfit. In our case, we did overfit, right off the first epoch, and we tried out several approaches to solve the overfitting, like increasing dropout, decreasing the complexity of the model, increasing regularization, but all of them did not work and the model would overfit after the first epoch, so we ran hyperparameter tuning on the first epoch itself. However, in production, the cost incurred for running a deep learning model might not make sense.

J.4. Content Based Methods

We investigated using CF-MCM model to potentially solve cold-start or limited information problems. However, the model suffered tremendously in terms of computation time for 20%

subsample data. In addition, it was not able to outperform the baseline model. Given its difficulty, we decided to forgo this approach, and stick with the bias-based model to serve the cold start problem.

K. Building a Complete Recommendation System

Below, we lay out how we would implement a complete recommendation system given our understanding of the problem statement, the models we built and the features we explored. In summary, we suggest the following approach.

1. Upon receiving a query from a user, narrow down the list of businesses to a short list using location cluster feature discussed in section F.1.a.
    a. Consider only these small set of businesses when providing a recommendation unless asked otherwise by the user
2. If a user has less than 5 reviews, serve a cold-start based recommendations using the bias-based model.
3. If a user has more than 5 reviews, make predictions on the small set of businesses found in step 1 using DeepFM. Rank the businesses using the ratings and provide the recommendations.

L. Conclusion

Overall, there is a strong reason to believe that DeepFM is a strong algorithm. First, it is able to outperform in RMSE across all data sizes versus the baselines. Additionally, we can see from its performance improvement and run-time that it is robust across dataset sizes, pulling further away from other methods as dataset size gets bigger.

However, we shouldn't be quick to discredit the baseline methods as our best models were able to outperform them only by 3% in RMSE. However, as demonstrated by their coverage, ranking and novelty metrics, it is hard to believe that they can provide a truly personalized experience to our users. Therefore, we believe that there is a place for our baseline models for cold-start problems.

However, we must note that while DeepFM is the superior algorithm, it will be difficult to make a case for lots of smaller organizations which can suffer from the following problems.

1. Lack of rich and abundant data
2. Lack of talent for model construction and maintenance
3. Lack of data-driven culture within the company that can get behind the idea.

For organizations in such a situation, a strong performance of our baseline and Matrix Factorization models suggests that those can be great alternatives. On the other hand, for organizations that do not suffer from those problems, we recommend using our complete recommendation system which uses a business retrieval system based on location, bias-based baseline for cold-start recommendations, and DeepFM model for general recommendation.