Yelp Recommendation System

Kumari Nishu (kn2492), Mohit Chander Gulla (mcg2208), Neelam Patodia (np2723)

k.nishu@columbia.edu, mohit.gulla@columbia.edu, np2723@columbia.edu

December 2019

Table of Contents

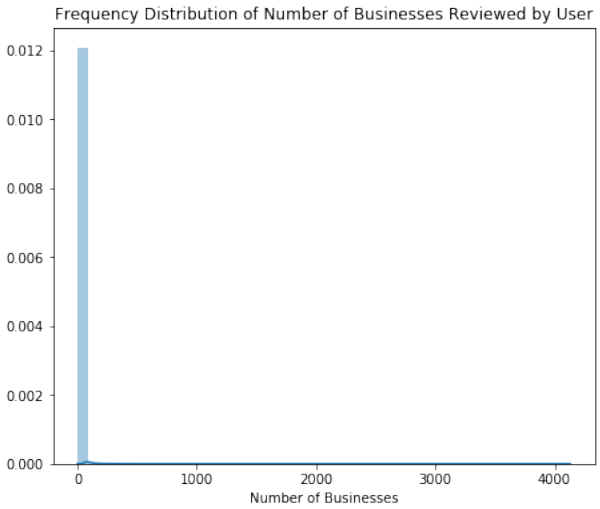
I. Project Scope

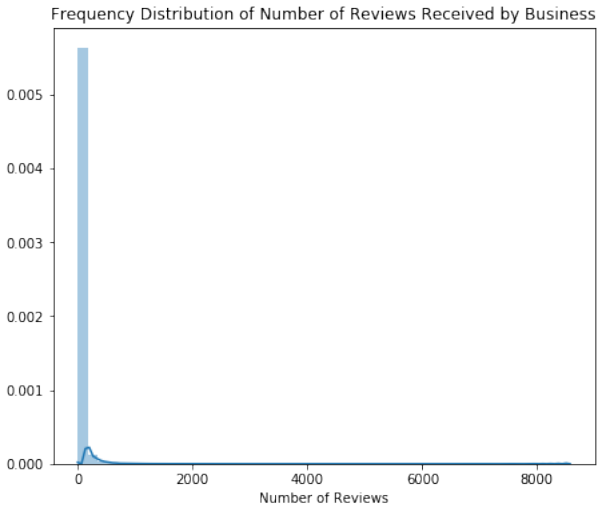
Our team's objective is to build a production-grade recommendation system using Yelp Dataset that provides curated business recommendations to users based on their explicit feedback and implicit features. This report goes in depth to define the scope of this project, to rationalize any assumptions that were taken in the process and arrive at a plan of action after analyzing results of all models that were explored.

II. Exploratory Data Analysis

In entirety, Yelp data comprises of more than 6.6M reviews for approximately 190K businesses. As we are prototyping, we will design and implement our models on a much smaller subset. This not just allows us to iterate changes faster to improve our model, but also because it is possible to learn user preferences meaningfully on a smaller subset of data.

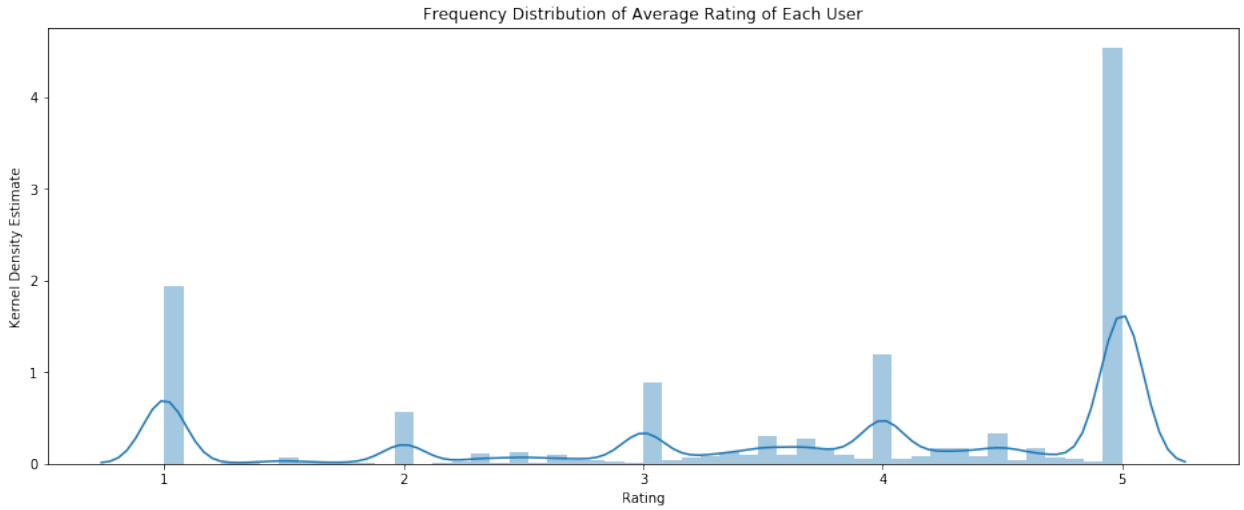
Total No. of Unique Users	**1,637,138**
Total No. of Unique Businesses	**192,606**





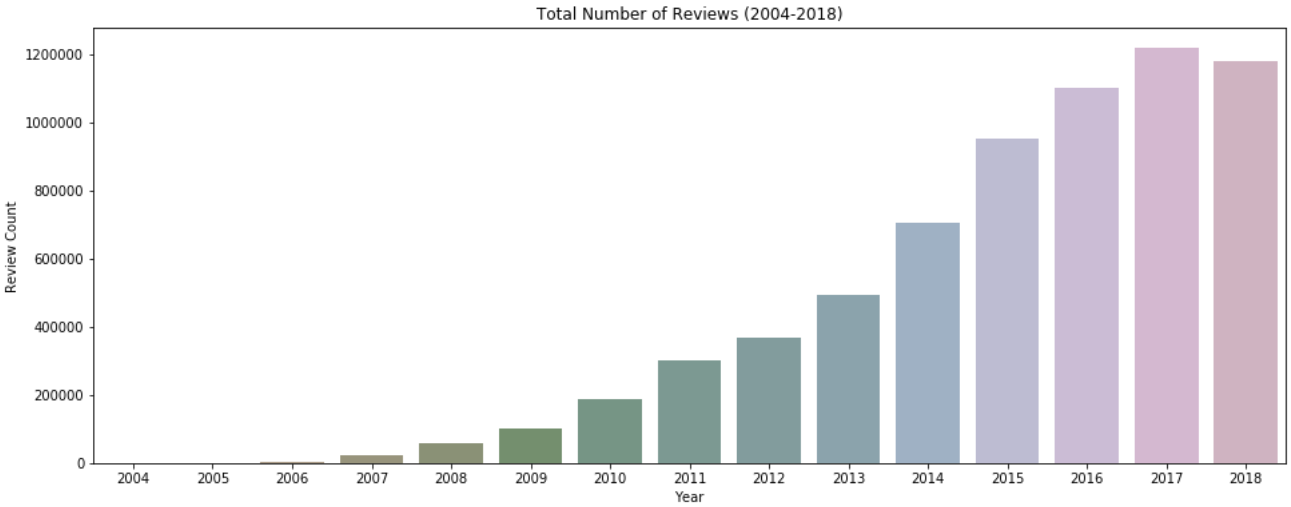
Looking at the frequency distribution of reviews, both from user and business perspective, we see that majority of users have reviewed very few businesses and majority of businesses have received very few reviews. The user-business interactions are not dense, and we would like to consider only active users and active businesses for our model in order to eliminate noise.

	Mean	Median
No. of Businesses Reviewed by User	4.08	1
No. of Reviews Received by Business	34.71	9

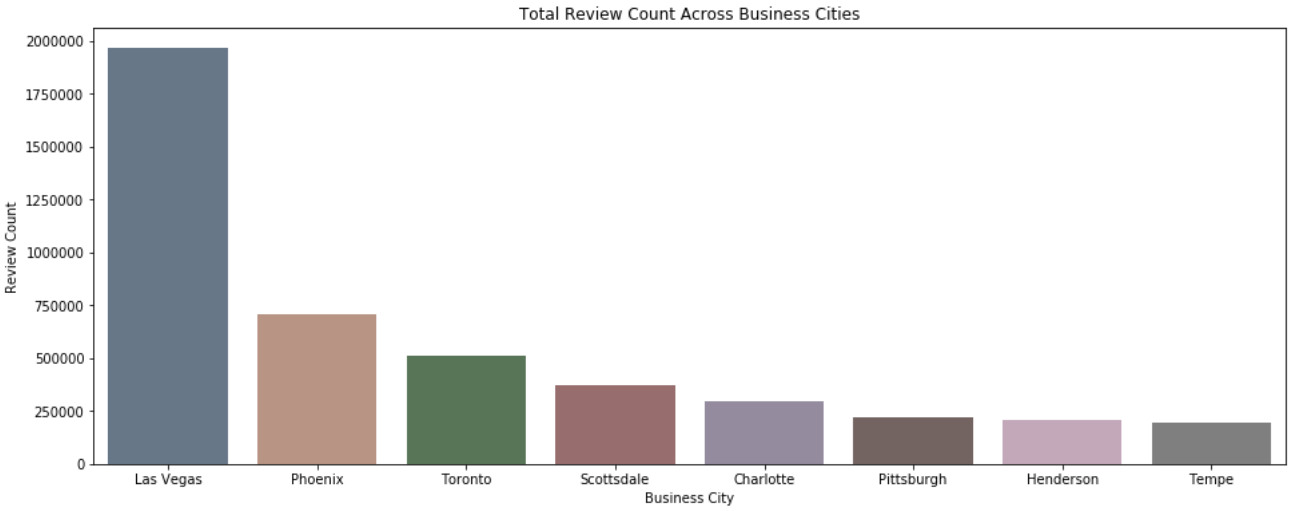


Mean of Users' Average Rating = **3.64** Median of Users' Average Rating = **4.0**

The above results were contradictory to our expectation. We assumed that users in general take the effort to provide a review mostly when they have had a negative experience. And even though there are significant number of users with their average rating across businesses equal to 1, the majority of users have their average rating greater than 4. As the perception of good and bad in terms of 1-5 scale differs from person to person, user bias is an important feature.



Over the last 14 years, the number of reviews that were posted on Yelp kept increasing before it dipped marginally in 2018. It is safe to assume that user preferences change over time and even the performance of a business can change considerably. As our end outcome is to provide users with recommendations, it should reflect the current state of a business, and align and learn from current taste of similar users. Moreover, as we are looking to train our model on a smaller subset of data, it is logical to consider recent data for analysis. We will subsequently add previous years data to analyze how well our model scales.



In terms of total review count, the businesses situated in Las Vegas have received more reviews than businesses in any other city. Las Vegas covers 30.4% of total reviews in our data. As users' spending capacity, preferences and type of businesses vary from city to city, we will subset our data to only include reviews to Las Vegas businesses. To ensure that our recommendations are relevant to users, in terms of proximity, we will consider only Las Vegas reviews data.

III. Business Rules

1. Active Users / Active Businesses – To only consider reviews for users that have at least 5 reviews and businesses that have received at least 5 reviews. This lets us ensure that our model learns on substantial data points.

2. Business Location : Las Vegas – As mentioned in the previous section, we are choosing to subset our data and consider reviews that have business location as Las Vegas. This ensures that our recommendations to users are realistic in terms of proximity.

3. Business Type : Restaurants – Due to disparity in ratings across different business types and how user bias and expectation can vary from one business type to another, we have selected only businesses that are categorized as 'Restaurants' in one of their categories.

4. Business : Open – As our end goal is to provide users with business recommendations, we are choosing to remove businesses that are now closed from our data. We learn from businesses currently open and only those show up in our recommendations for users.

5. Review Date : 2014-2018 – As user taste and business performance can change notably over years, we will consider reviews data for the last 5 years as the base set. We will train our model on this data and check scalability by subsequently adding previous years data.

IV. Sampling Strategy

After applying the business rules mentioned in the previous section, the size of our data reduces both in terms of unique users and unique businesses it covers. Working on a smaller subset of data ensures that we can explore our model better and quicker, but we have to ensure that the sample is indicative of the population otherwise the model would not scale well in accuracy.

Total No. of Unique Users	**31,710**
Total No. of Unique Businesses	**4,221**
Total Review Count	**362,907**

Using this subset of review data, we will split it further into three sets – train, validation and test. The test dataset is created by taking the latest review, by date, of each user. Naturally, this data set will consist of 31,710 reviews as one review for each unique user is held out. The validation dataset is created by taking any two reviews for each user, which is why it will have exactly twice as many records as test dataset. The remaining reviews comprise the training set on which we will train our model.

Training Set Review Count	**31,710**
Validation Set Review Count	**63,420**
Test Set Review Count	**267,777**

V. Feature Engineering

Using implicit features to learn user preferences can play a crucial role in improving the overall quality of the model. Yelp captures various attributes for each user and business in their dataset. We will leverage the existing set of features provided to us and selectively create new features that can intuitively help us predict the rating for the holdout review. The metadata of important features to our model are listed below:

User Features

1. Review Count – total reviews provided by each user
2. Average Stars – average rating of all reviews for each
3. Fans – number of fans of each user (normalized by review count)
4. Friends – number of friends of each user (normalized by review count)
5. Elite – total number of years the user held elite status
6. Lifetime – difference in days between last review date and yelping since date
7. Compliment Count – total number of compliments for all reviews of each user
8. Compliment Score – sum of each compliment count (normalized by review count)

Business Features

1. Review Count – total reviews received by each business
2. Average Stars – average rating of all reviews received by each business
3. Total Hours – total hours of operation in a week for each business
4. Total Checkins – total checkins by all users to each business
5. Age of Business – difference in days between latest review and first review
6. Business Type – one-hot encoding of top 10 business type for each business. Along with 'Restaurant' type weather a business belongs to any of the following types is recorded:
   Food, Fast Food, Nightlife, American, Bars, Mexican, Sandwiches, Pizza, Burgers

VI. Factorization Machine using LightFM

Factorization Machines (FM) is a supervised learning model. It is usually trained by Alternating Least Squares (ALS), Stochastic Gradient Descent (SGD) or Markov Chain Monte Carlo (MCMC). We have used the implementation of FM from LightFM library which is based on SGD.

FM models the interactions of variables by mapping them to a low dimensional space. FM is a generalization of a few factorization models, including matrix factorization and SVD++. FM is computationally efficient on large sparse datasets. This property is one of the best aspects of FM which makes it popular for recommendation. We explored FM as a classification model to predict the rating on holdout set.

Model Training

The ratings data is acquisitioned in a tabular form as a pandas DataFrame in python. Similarly, we have user features and item features in the same form. However, we needed a sparse matrix representation (also known as interaction matrix in FM) for ratings data, user features and item features to implement the model from LightFM library. Hence, we first transformed our data to the required sparse matrix format for all three datasets.

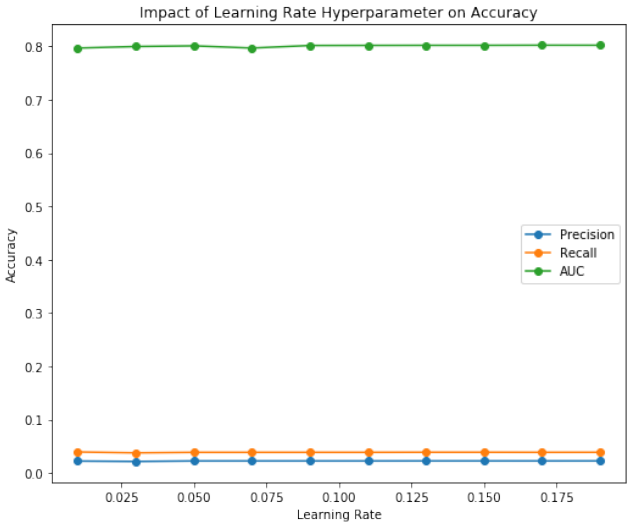
LightFM makes it possible to incorporate both item and user metadata into the traditional matrix factorization algorithms. It represents each user and item as the sum of latent representations of their features, thus allowing recommendations to generalize to new items (via item features) and to new users (via user features). Additionally, we have also explored the FM model without using explicit features to see how the model performs.

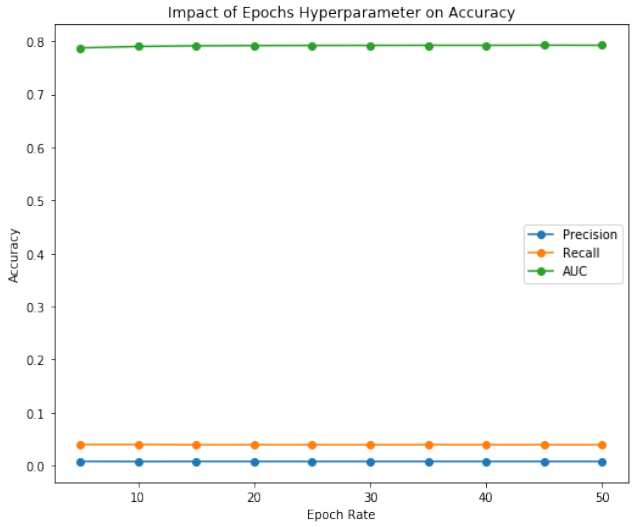
Hyperparameter Tuning

We used the validation dataset to tune the hyperparameters of FM. We tuned two parameters – learning rate and epoch. We tuned the learning rate ranging from 0.01 to 0.19 over ten different

values. We tuned the number of epochs from 5 to 50 over ten different epoch values. We found the best learning rate and the best epochs using the validation dataset.

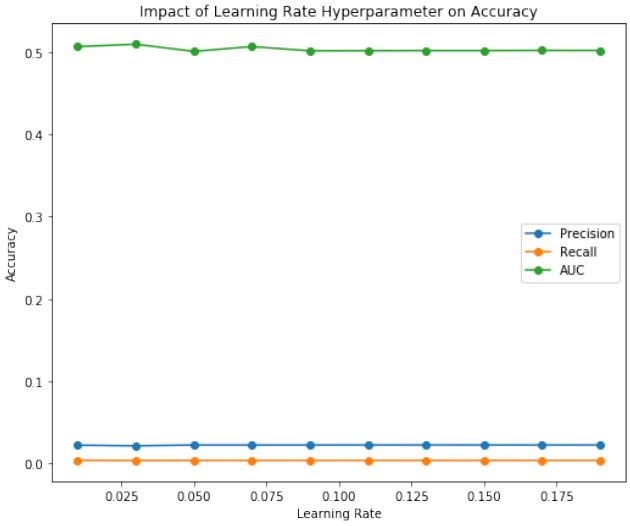
Factorization Machine with Implicit Features

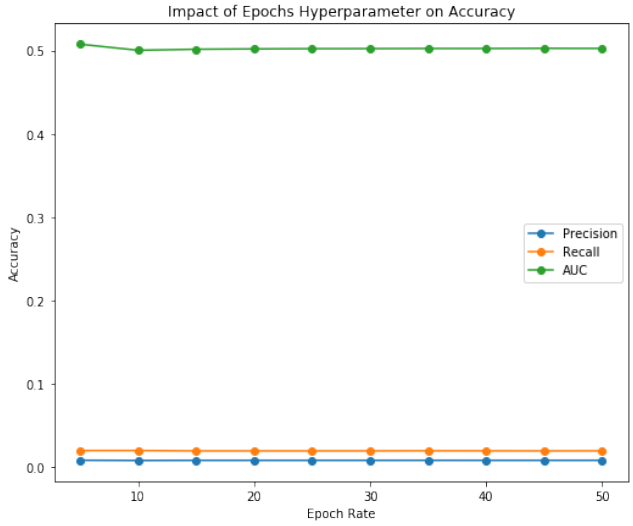




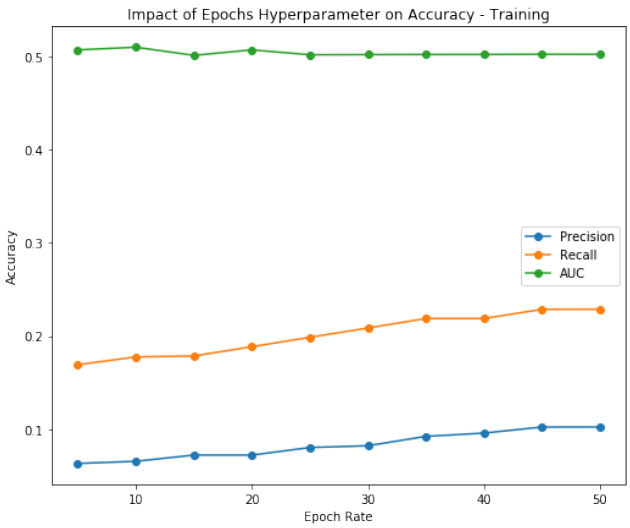
The recall and precision of this model is very low even with tuning its hyperparameters. We will now check the same results with removing implicit features.

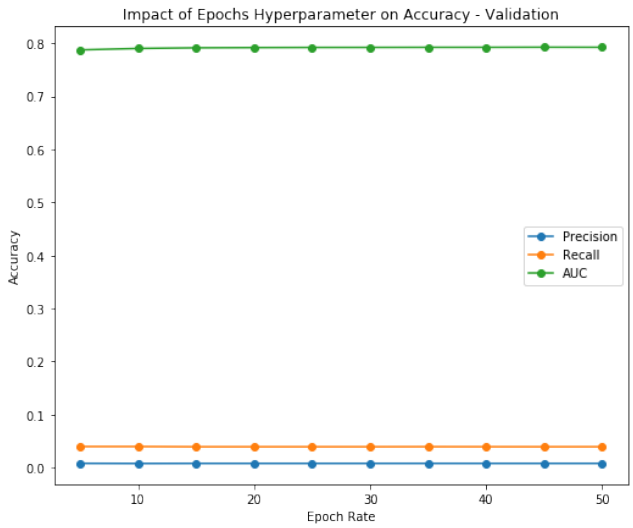
Factorization Machine without Implicit Features





There is no improvement in any of the accuracy metrics and in fact AUC dipped significantly as soon as we removed features from our model. Although on validation data, removing features has no impact, the accuracy metrics improved significantly on training data with no features.





This indicates that the model is able to improve on training dataset but does not generalize well and accurately predict ratings on validation set. The improvement in accuracy was only possible after removing the user features. Therefore, we will train our final model without features using the best set of hyperparameters. But we are not very confident of getting good results on test.

Model Evaluation

We have used three evaluation metrics to measure the performance of our FM model - precision, recall, and AUC score. We have evaluated the performance on both validation and test dataset.

Model	Recall	Precision	AUC
Validation Set	0.0231	0.039	0.8011
Test Set	0.0044	0.036	0.7491

The model achieves very low accuracy both in terms of precision and recall. The improvement in accuracy that we observed on training set, did not translate into the test set. Overall the model that we have does not perform well and would have a hard time in outperforming even baseline.

Please refer to the Python Notebook for in detail analysis of model exploration and results.

Model Recommendations

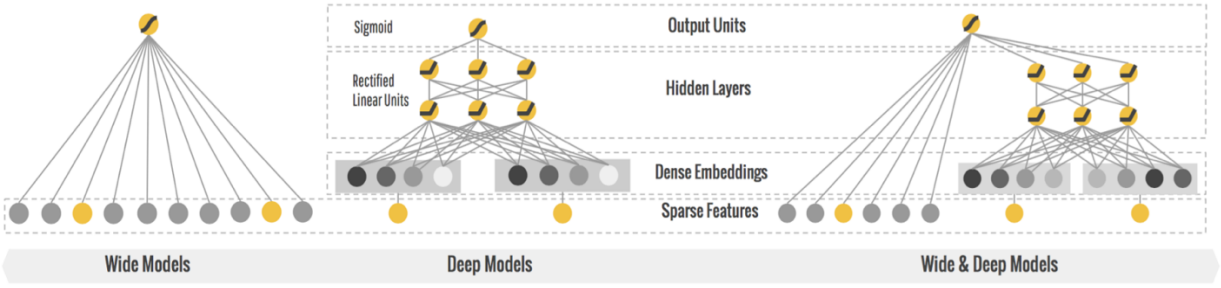
We have constructed an exhaustive list of user-business pairs – businesses that a user has not reviewed before. We find the top-k recommendations using the predicted rating. Essentially, we predict the ratings for all the unvisited businesses by a user and select the top k businesses with highest ratings. We return these businesses as the top-k recommendations for that user.

Conclusion

In this recommendation model, we implemented Factorization Machines by using both explicit user-item features and without features. Surprisingly, the simple model gave better performance compared to the one with implicit user and item features. This behavior can be attributed to the noisy behavior of the features and the lack of learnable patterns from those features. In all the training and prediction process, we have run our LightFM model on a single thread which can be improved by running on a cluster with multiple thread, and thereby reducing the training and prediction time.

VII. Wide & Deep Learning Model

Google's Wide & Deep Learning combines the power of both memorization – by training a wide linear model and generalization – by training a deep neural network. It is useful for classification and regression problems with sparse inputs such as recommender systems, search and ranking problems. For our use case, we define wide categorical and deep continuous columns and learn user preferences through these implicit and explicit features. It allows us to predict the rating for a business by a user. Based on the predicted ratings, we can provide top-k recommendations for each user.



Model Training

To learn user preferences, we build tensors for average rating for each user and average rating for each business, compliment score and count, number of friends and fans, lifetime of user and business, total check-ins, hours of operation, and presence of business category. The model is trained as a classifier on these wide and deep tensors to predict rating on a scale of 1-5.

Hyperparameter Tuning

The wide and deep learning model can be customized by adding more hidden layers and nodes in the architecture to learn user taste better from the embedding vectors. We will iteratively add more layers to our model and measure how the accuracy changes as our model becomes more

complex. We will learn number of hidden layers and nodes on the validation set and use the best model to make predictions and recommendations.

Hidden Layers & Nodes	Recall	Precision	AUC	Loss
[100, 50]	0.4912	0.4415	0.6509	1.4101
[500, 250, 50]	0.4466	0.4466	0.6541	1.4023
[1000, 500, 100, 50]	0.5579	0.4414	0.6508	1.4061

We can observe that as we increase the complexity of our deep learning model by adding more layers and more nodes within each layer, the overall accuracy on validation dataset increases.

However, it is also important to note that the increase in accuracy is marginal (except for recall) and we should consider selecting a sub-optimal set of hyperparameters in certain situations as training a deeper model can increase execution time significantly without any tangible benefit.

Model Evaluation

To evaluate the quality of our model, we will focus on accuracy metrics such as precision, recall, and auc. We will choose the best set of hyperparameters based on accuracy on validation set. The model is explored by increasing the depth of the model in terms of layers and nodes. In our use case, we tested our model for following configurations: {100, 50}, {500, 250, 50}, and {1000, 500, 100, 50}. The model with most depth gave the best results, although by a very slight margin.

On test data, the model performed well and in accordance with validation results.

Model	Recall	Precision	AUC
Wide & Deep	0.6797	0.4714	0.6703

Model Recommendations

Recommendations are made using the same exhaustive list of user-business pairs as discussed in the previous section. Based on the predicted rating of each un-reviewed business for a user, we will provide the top-k highest rated businesses as recommendations.

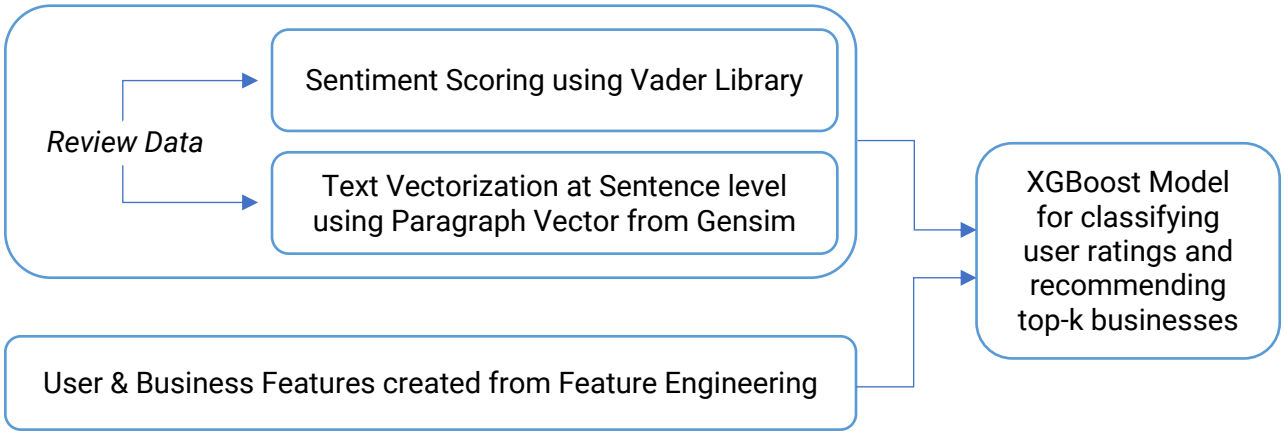
Conclusion

As the wide and deep learning model is not just capable of learning features but can generalize very well, we saw much better accuracy from this implementation. Furthermore, the features we created in the feature engineering played an important role in improving the quality of the model.

VIII. Content-Based Model using Doc2Vec and XGBoost

In this approach, we intended to utilize the textual features available in Yelp dataset in the form of reviews given by users to businesses in conjunction with the features discussed earlier.

We followed the below approach:



Sentiment scores help in assessing what a user feels at a broader level for a business it reviews. It acts as a pseudo indicator of the rating given by the user. Vader Library is used for this purpose as it has been pre-trained on social data and efficiently identifies the various ways users tend to emphasize about their liking in social setting.

Paragraph Vector, Doc2Vec is an unsupervised neural network model, which has been used to generate a continuous vector representation for reviews. The commonly used techniques such as count vectorization, n-grams and tf-idf cannot be used in our case as it loses the ordering of the words, as a result sentences having the same words are represented in an identical manner. Using n-grams (preferably bigrams/trigrams) we could only select top k most frequent phrases. In this case, for a highly rated restaurant we could miss nuances pertaining to negative feedback quite easily by assuming that the frequency of positive phrases would be more than negative comments. We thus used Doc2Vec as it allows us to capture the content of the entire review into a fixed dimension, user-defined vector size ($N \; X \; M$, where N is number of reviews and M is the fixed vector size). It is an extension of word vector model and it generates sensible context aware word embedding for input sentences of variable length.

We utilized these new set of features, namely – *sentiment scoring* and *reviews* along with those obtained through feature engineering and passed it through a classification model, XGBoost for ratings classification. XGBoost is chosen over other classification algorithms such as Random Forest and SVM as it reduces error by reducing bias and is known to be deployed at production level in organizations to handle large datasets with 100s of features computationally faster.

Model Training

Sentiment Scoring – The pre-built Vader library allows us to determine the sentiment scores for review data without the need of any further training. It measures intensity along 3 parameters – positive, negative and neutral – and combines them to give a compound score which has been utilized for the purpose of our analysis.

Paragraph Vector – Unlike the word vector model which only considers words as input nodes, in the paragraph vector model each paragraph id (in our case review id) also acts as input node and is mapped to a unique vector. We have assigned a unique id for each review. The paragraph vector (D) along with word vector (W) is considered as a member of the context set. The context is sampled from a sliding window of fixed length over the review. The context window considers words to the left and right of the target word. While the paragraph vector is shared only across the context from the same paragraph, the word vectors are shared across all paragraphs. The concatenation of the vectors (W and D) is then used to predict the next word in context. Using this vector, we have a thorough representation of both the user and business features, enabling us to better our recommendations.

XGBoost – The combined user and business feature set inclusive of the review set is then fed into a classification model. XGBoost uses ensemble techniques which combines the estimates from weaker models. It is an iterative training process such that new trees are added that predict the residuals of prior trees and combines it with the previous trees to make the final prediction.

Hyperparameter Tuning

We used the validation dataset to tune the hyper-parameters of Paragraph Vector and XGBoost

Paragraph Vector

Since paragraph vector is unsupervised, in order to arrive at its optimal set of hyperparameters, we base the selection on the performance evaluation of the basic XGBoost model.

1. The first task in building the vector space is to choose between the two base architectures – Distributed Memory (PV-DM) and Distributed Bag of Words (PV-DBOW). Given window of words $(W1, W2, W3, W4, W5)$ the PV-DM model predicts $W3$ given the rest, while the PV-DBOW model predicts $W1, W2, W4, W5$ given $W3$. For us, we found PV-DBOW architecture to perform better.

Base Architecture	Recall	Precision	AUC
PV-DM	0.3634	0.4684	0.7111
PV-DBOW	0.3661	0.4687	0.7134

2. Dimensionality of Feature Vectors: We set the value of this hyperparameter to 100 as used in most cases. Also, since the length of our reviews were mostly around 1-2 sentences, increasing the dimensions would result in adding noise.

3. The window size which contributed towards the contextual element was tuned over three sets: 5, 8 and 10. Though all are giving nearly same results, we chose a window size of 5 as the length of review sentences are not long either.

Window Size	Recall	Precision	AUC
5	0.3814	0.4827	0.7218
8	0.3749	0.4745	0.7192
10	0.3661	0.4687	0.7134

4. For training the selected model, we choose Negative Sampling over hierarchical softmax as the former is similar to stochastic gradient descent. Instead of changing the weights each time by accounting for thousands of observations, it uses a sample of them which is defined by K. We thus tune this value of K over 5, 10 and 15. K=15 was the most suited parameter for our data.

Negative Sampling	Recall	Precision	AUC
K = 5	0.3655	0.4629	0.7145
K = 10	0.3661	0.4697	0.7134
K = 15	0.3831	0.4745	0.7222

5. We set the sample threshold to be $10e^{-6}$ so that high frequency words are down sampled. Due to this feature we did not have to explicitly remove stop words in the text preprocessing step as it is taken care of by the model.

XGBoost

Post tuning the hyperparameters of the paragraph vector model, we tune the XGBoost model on three most significant hyperparameters.

1. The number of estimators is tuned to 100 post assessing its efficacy for 50, 100, and 150. We have chosen 100 instead of 150 as by adding 50 more estimators the performance increases very marginally while the computational time increases substantially.

Estimators	Recall	Precision	AUC
50	0.3835	0.4674	0.7208
100	0.3832	0.4751	0.7222
150	0.3844	0.4745	0.7234

2. The max depth of tree is tuned for 5, 7 and 9 values. A depth of 9 is most optimal for our data

Max Depth	Recall	Precision	AUC
5	0.3832	0.4751	0.7222
7	0.3894	0.4780	0.7248
9	0.3969	0.4863	0.7270

3. The sub-sampling is tuned for values of 0.6, 0.7 and 0.8. Optimality was achieved for 0.6 fraction

Sub Sample	Recall	Precision	AUC
0.6	0.3983	0.4903	0.7280
0.7	0.3945	0.4841	0.7264
0.8	0.3969	0.4863	0.7270

Model Evaluation

Using the tuned hyperparameters from above, we will evaluate the model on validation and test dataset. The accuracy metrics for the best trained model are as below:

Best Model	Recall	Precision	AUC
Train Set	0.8052	0.8496	0.8858
Validation Set	0.3983	0.4903	0.7280
Test Set	0.4023	0.4877	0.7457

Model Recommendation

Since text reviews and sentiment scores would not be available for new user-business pairs, we resorted to the set of master user and business features to derive these features for a new pair. For the 100D review features, average of all reviews which this user has given in the past were considered. Since reviews highlight user preferences/likes-dislikes, for new recommendations we believe the average of all user reviews is a suitable parameter. Similarly, for the sentiment score, we believe it is more associated to whether a business has a higher positive/negative sentiment. Therefore, for a new user-business pair, the average of all sentiment scores received by this business is considered.

Conclusion

In this model, we have thus tried to incorporate the additional information available in the form of text reviews to recommend businesses to users. The use of sentiment score and text vectors

in conjunction with other user-business features and passing it through an XGBoost model for classification provides promising results.

IX. Baseline Model

To understand how effective our model is in predicting the rating for latest review of each user, we will use Surprise package's baseline estimator which internally implements Alternating Least Squares (ALS). The algorithm tries to minimize the following regularized squared error:

$$\sum_{r_{ui} \in R_{train}} \left(r_{ui} - (\mu + b_u + b_i)\right)^2 + \lambda\left(b_u^2 + b_i^2\right)$$

where r_{ui} is the baseline estimate for a given user and item, μ is global mean of all ratings, b_u is user bias, b_i is item bias, λ is the regularization parameter.

For our implementation, we have set the regularization parameter of user bias and item bias to its default values and minimizing the total loss in 20 epochs. The trained model is evaluated on validation set and the results are as stated below:

Dataset	Recall	Precision	AUC
Validation Set	0.2749	0.2749	0.5468
Test Set	0.2582	0.2582	0.5364

We are now ready to evaluate this baseline model against the three models we implemented in the earlier section of the report. The comparison between all models is discussed in detail in the next section.

X. Model Evaluation

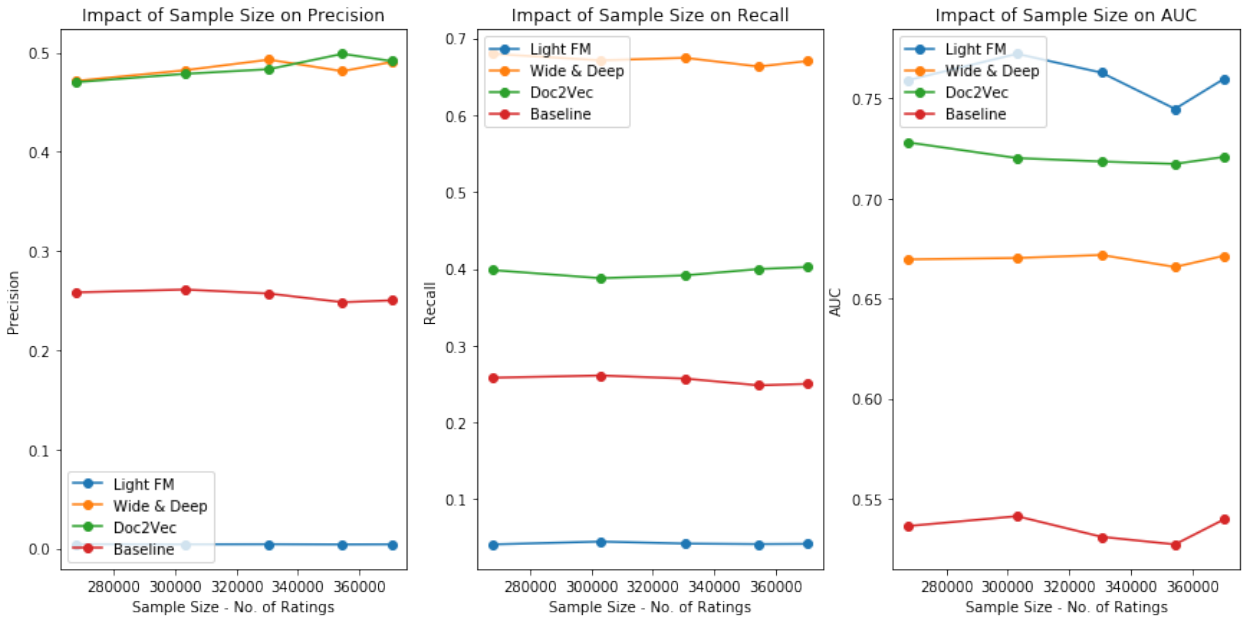
Including the baseline model, we have four models in total that predict the rating on holdout set. We will consider the implementation of factorization machines without features for comparison as we observed that our model outperforms when user and business features are removed. We can see that our Wide & Deep and Content-Based model outperform baseline, whereas FM fails.

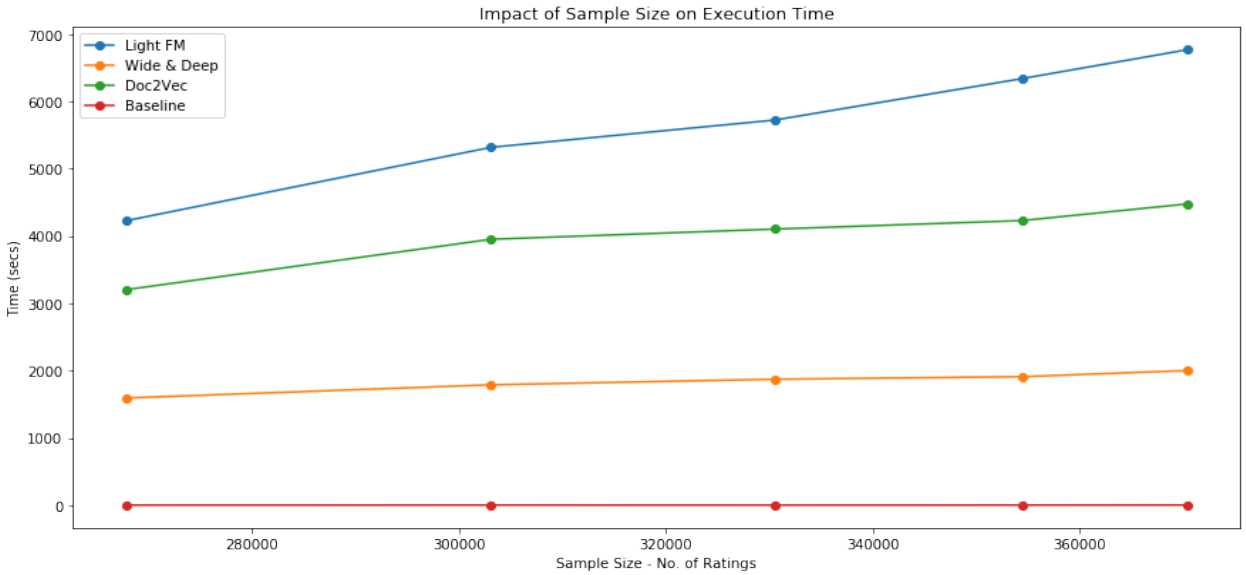
Model	Recall	Precision	AUC
FM	0.0413	0.0045	0.7592
Wide & Deep	0.6797	0.4714	0.6703
Doc2Vec + XGBoost	0.3983	0.4701	0.7280
Baseline	0.2582	0.2582	0.5364

XI. Model Scalability

Given that the end outcome of this project is to evaluate various recommendation systems and productionalize the one that performs best, it is critical to evaluate how well each model scales. As we discussed earlier, we choose a relatively small subset of our data to explore each model and generate recommendations. In order for any model to be effective, it needs to sustain the load of a production system where the prediction task may involve all users and businesses. We need to test the scalability of our model in terms of both accuracy and execution time.

Impact of Sample Size on Accuracy & Execution Time





Initially, we considered only last 5 years of reviews and to test scalability we will iteratively add an extra year of data going back. It naturally increases the total review count and we would like to observe If the accuracy or execution time gets severely impacted.

As we can see in the charts above, the accuracy of all model – in terms of precision, recall, and AUC – is fairly consistent. This indicates that our sampled dataset is a true reflection of the total population. However, in terms of execution time we see that LightFM is the slowest, followed by Doc2Vec + XGBoost content-based model, followed by Wide and Deep Learning model. Baseline is the fastest among all models and scales very well.

However, the decision of which model is most suited to be productionalized is a combination of the two graphs. We need a model that not just scales well but gives good accuracy while it does. In that regard, we can discard LightFM as it has the worst accuracy and is the slowest (although it has a multi-thread implementation which can be explored). Even baseline model is not a good option as it only scales well and does not provide good accuracy.

XII. Coverage

Up until now, our focus was on accuracy metrics which was reliant on how well our model would classify the rating on test set, i.e. one held out review for each user. However, we need to now leverage our classification model to generate business recommendations for each user and we achieved this by essentially finding the rating of all user-business pairs that have no interaction and returning top-k highest rated businesses as predictions.

This is equivalent to filling the user-business interaction matrix with rating values wherever it is missing. As the original interaction matrix is very sparse, we have a massive task at our hand if we want to predict rating across all businesses for each user. We generated this exhaustive list and it required 116M predictions of user-business pairs. This is computationally not feasible on our system. In future, when a model is productionalized, such prediction jobs can be scheduled at weekly cadence on a server with large compute.

However, in order to evaluate the coverage of our recommendations, we will attempt to calculate approximate catalog coverage. We randomly subset the interactions matrix for set of users and businesses in the same ratio as original data. After reducing the dimension of our interactions matrix diagonally we were able to compute catalog coverage for each model.

Catalog coverage is defined as the count of the union of all top-k recommendations for all users, divided by the count of all items that could have been recommended.

Catalog Coverage	**FM**	**Wide & Deep**	**Doc2Vec + XGBoost**
	0.2302	0.4737	0.5318

XIII. Conclusion

After building several prototype recommendation systems, we can make an informed decision as to which model and which approach works best for Yelp data. Among our options, we would recommend an ensemble of the Wide & Deep Learning model and Doc2Vec + XGBoost content-based model. Wide and Deep model offers impressive generalizability, gives an option to encode and learn from specific feature interactions, and is part of the TensorFlow library that is based on Theano – a Python library that optimizes matrix calculations. From the content-based model, we get the advantage of leveraging our reviews data. With help of natural language processing, we were able to extract meaningful features that when fed into an XGBoost classifier gave very good results. To productionalize a recommendation system, we need to put emphasis on how well our model scales as the amount of data increases. Our analysis in section XI of this report, give us confidence to move forward with a recommendation system of this setup.

XIV. Future Work

1. Image Data – We should explore and implement a model that can learn user preferences through the images that are tagged to a review. There is a good chance that we may be able to extract some insights from this data.

2. Social Network Analysis – In our current project, we have only considered how large is a user's friend circle is as it indicates his/her social media presence. It would be interesting to form a social network graph of users and learn if taste and preferences are similar in strongly connected components of this graph.

3. Ensemble – We should explore various approaches of implementing an ensemble that aggregates qualities of independent models. It is essential if we want to combine a deep learning neural network, with models that specifically tackle text and image data.

4. Business Type – As we have only considered 'Restaurant' businesses in our analysis, we should incorporate all business functions and make our model more generic.

XV. References

1. Wide & Deep Learning: Better Together with TensorFlow, Google
2. How to build a LightFM Recommendation System in Python, Towards Data Science
3. Doc2Vec Paragraph Embeddings – Documentation & Tutorial, Gensim
4. Wide & Deep Learning for Recommender System, Research Paper
5. Factorization Machines with LibFM, Research Paper