CONTEXT-DRIVEN YELP RECOMMENDATIONS

Amogh Mishra (am5323), Siddhant Shandilya (ss5919), Swarna Bharathi (sm4776), Benjamin Livingston (bwl2114)
amogh.mishra@columbia.edu, siddhant.shandiya@columbia.edu,
sm4776@columbia.edu, benjamin.livingston@columbia.edu

1. Problem Statement

We are a mock data science team at Yelp responsible for serving superior recommendation to its customers both old and new, yielding to higher customer satisfaction and conversion rates.

2. Objective

Yelp has an incredible amount of data, but what's the best way to harness it?

Ratings, of course, are king – the most valuable and easily-processed information feedback users offer comes in the form of stars. One stars are good, five stars are bad. This we know. This data alone is enough to build a collaborative filtering algorithm that can serve restaurant recommendations to a user based on the set of restaurants they've rated highly. But there's an opportunity to do better.

There is so much more metadata that Yelp has on users and restaurants that can provide context for a recommendation. By factoring this context into recommendations, we can perhaps better understand a user's preferences, strengthen our model, and glean more insight into the motivations behind reviews.

There are countless possibilities for what we can accomplish using this contextual approach, such as:

- Recognizing that a user will only utilize expensive restaurants, so we can ensure that we don't suggest cheap ones
- Identifying that a user is only interested in restaurants with parking
- Considering a user's reputation on Yelp (including their feedback) when we factor in their preferences and serve them recommendations
- Picking up that a user enjoys Thai food, and pretty much only Thai food
- Explicitly taking into account a user's strong preference for takeout restaurants

Using collective matrix factorization and content-based recommendation approaches allows us to harness this metadata to shoot for these goals. Content-based recommendation can discover latent connections between different restaurants based on reviews, and collective matrix factorization can efficiently factor a wide swath of metadata for both users and restaurants into our model simultaneously to allow us to discover the many different elements that drive the motivation for a review, either for a user or a specific restaurant.

Ultimately, our goal will be to establish which portions of Yelp data are most useful for bolstering recommendations (at least in this fashion), which will help demonstrate which data is most vital for Yelp to collect and harness.

3. Assumptions

- Only Las Vegas city is considered for the recommendation.
- Data subset from 2014-2018
- Restaurant considered which are open and have more than 5 reviews.

- Time-decay in restaurant's profile not considered
- Reviews are from genuine customers and not faked in favour of the restaurants.
- No bias in votes to reviews.

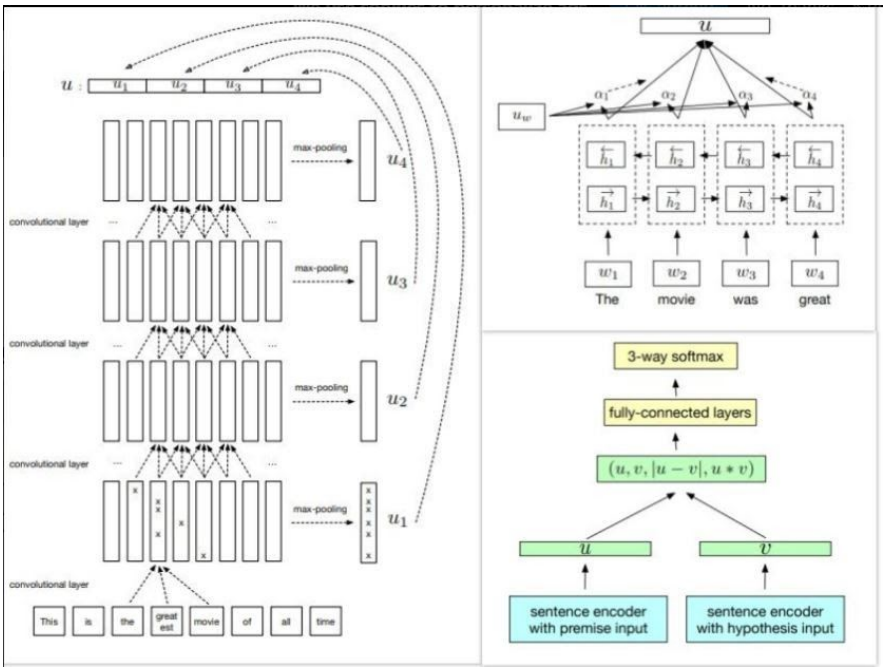
4. Data & metadata

We will use the Yelp dataset, which has a massive array of both user and restaurant metadata available (what metadata is available is detailed further at the link).

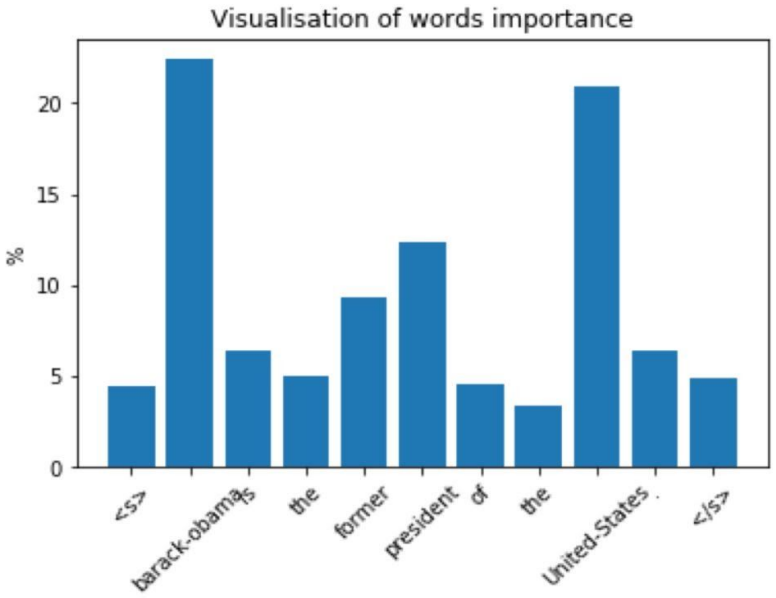
5. Models Used

- **Factorization machines (FM)**: A novel **collective factorization** approach to simultaneously factoring in both user and item metadata that uses one-hot encoding to accomplish this otherwise infeasible task, and uses creative means to consider interactions between features.

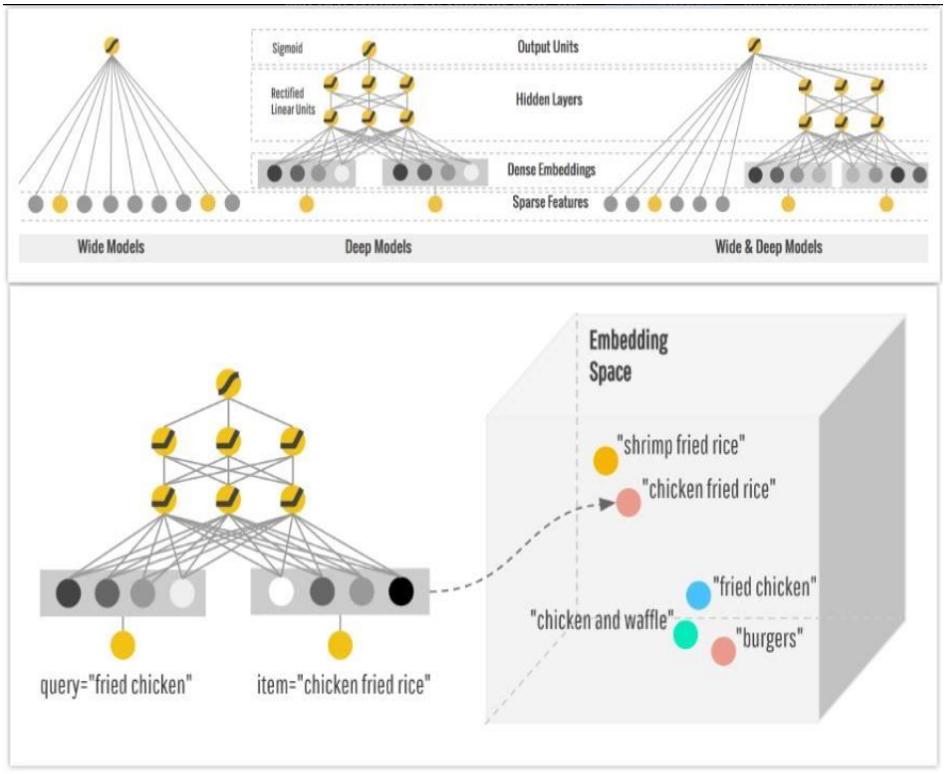
- **Infersent**: Infersent is a sentence embedding method developed by Facebook which provides semantic representation of the reviews. It is used for **content-based recommendation** on reviews for cold-start problem. Infersent is a sentence encoder trained in supervised method setting on a large Stanford Natural Language Inference (SNLI) corpus. It is a BiLSTM encoder-decoder model with max-pooling yielding embedding of 4096 dimensions. It performs much better than other unsupervised sentence encoder such as FastSent and Skip-thought and it computationally efficient. It takes less than 3 minutes to get weighted embedding of all the restaurants in Las Vegas for 2018 on 6GB NVIDIA 1060 GPU, core i7 machine. It performs superior to Universal Sentence Encoder (USE) because USE wasn't utilizing the GPU power making the embedding evaluation intractable. It also uses the representation of sentence across hidden layers to depict the important features on the sentence below. This can be very helpful in understanding pattern of features which attracts most votes on reviews.



Infersent model architecture



- [Wide and Deep Learning for Recommendation system]: This method uses memorization from Generalized Linear models and generalization from Neural Network models to predict the rating for a user. Memorization of feature inputs helps in finding patterns in prediction using cross product feature interactions and it is considered to perform well. However, when It comes to generalization, Deep neural networks can generalize better to unseen feature combinations through low-dimensional dense embeddings learned for the sparse features. Being prone to overfitting when it comes to prediction, Deep models alone tend to overfit the underlying model. Thus we try to use a combination of both GLM and deep Neural Network side by side at the same time to obtain a lift in the prediction accuracy. Following is its architecture:



- **KGAT: Knowledge Graph Attention Network for Recommendation:** A latest State-of-the-Art paper published in KDD 2019. It proposes that a superior recommendation requires side information rather than just user-item collaboration. With such an objective, factor machines fail to work as they assume each interaction as an independent instance and collaborative filtering doesn't encode side information. This paper constructs knowledge graphs and leverages Graph Convolution Network for encoding information using TransR algorithm. It also uses attention mechanism on top of it to filter important neighbours. This architecture provides superior quality of recommendation since all the attributes of the restaurants are linked together such category, location etc. Unfortunately, the computation on our machine and Colab ran into exhaustion of memory and we weren't able to replicate the results for 2019 data.

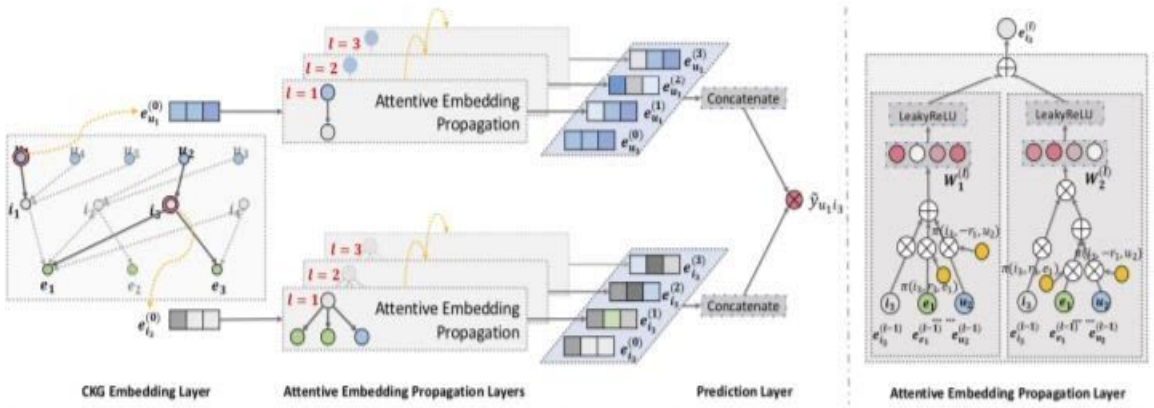


Figure 2: Illustration of the proposed KGAT model. The left subfigure shows model framework of KGAT, and the right subfigure presents the attentive embedding propagation layer of KGAT.

We also used two simple baseline models that we will outline later.

6. Metrics Used for performance evaluation (both quality and scalability)

a) **Root Mean Square Error (RMSE)**: RMSE is the standard deviation of the residuals (prediction errors). Residuals are a measure of how far from the actual ratings, the predicted ratings are. It is a measure of how spread out these residuals are. It gives more weight to outliers.

Formula: $$\sqrt{\frac{\sum_{i=1}^{N} (Predicted_i - Actual_i)^2}{N}}$$

b) **Mean Absolute Error (MAE):** MAE is the average vertical distance between each actual rating and the predicted rating. Formula: $\frac{1}{n}\sum_{i=1}^{n} \left|Predicted_i - Actual_i\right|$

c) **Coverage**

1. **User coverage**: First we define what a good recommendation is (eg, a predicted rating >3.5). Then, user coverage is the proportion of users for which at least k movies can be recommended well. First, we define what a good recommendation is and fix a value of k to calculate the coverage value.

2. **Catalogue coverage:** First we define what a good recommendation is (eg, a predicted rating >3.5). Then, Catalogue coverage is the proportion of movies for covered in the recommendation for all the users. In other words, the fraction of items that are in the top-k for at least 1 user.

d) **Model size**: Sample size from the original data to train the model including the learning of hyperparameters.

7. Sampling Methodology

We will subset our data spatially, temporally, and categorically, using only restaurant data from the city of Las Vegas from January 2014 to November 2018, only considering reviews by users that had more than five reviews. We will test our data by holding out the final review by each user and considering the root mean square error of our predictions for that final review based on our model.

8. Baseline Models

For any recommendation system we build or any algorithm we use, It is important for us to test if it is performing well. Since we are short of a relative metric most of the time, the simplest way to measure the performance is to test our algorithm against a baseline model. A baseline model is one which is the simplest model and we don't put in extra effort to make any prediction for each user-item pair. We have incorporated two baseline models in our analysis:

a) **Bias baseline model**: In this, we are predicting the baseline estimate for given user i and restaurant j by using:
$\hat{r}_{ij} = \mu + b_i + bj$ .Here, $\hat{r}_{ij}$ is the estimated rating of the ith user and jth restaurant. b_i is the bias of the ith user and b_j is the bias of the jth restaurant from the ALS model.

b) **ALS:** Here, we have used the ALS method from pyspark. Spark Machine Learning currently supports model-based collaborative filtering, in which users and products are described by a small set of latent factors that can be used to predict missing entries. spark.ml uses the alternating least squares (ALS) algorithm to learn these latent factors. Matrix factorization is a technique in which we create latent factors to build a recommendation system where we try to jointly factor the users and items in the same latent space. This significantly reduces the size of the matrices which gives an enormous computational advantage over the behaviour based models like user-user/item-item collaborative filtering. Apart from scalability, it also reduces the possibility of overfitting by regularizing the model so that it can be generalized over the data set.

For Baseline models, the final results on the test set are:

Bias Baseline

Metric	RMSE	MAE
Bias Baseline	1.24	1.00

ALS Baseline

Metric	RMSE	MAE
Bias Baseline	1.65	1.38

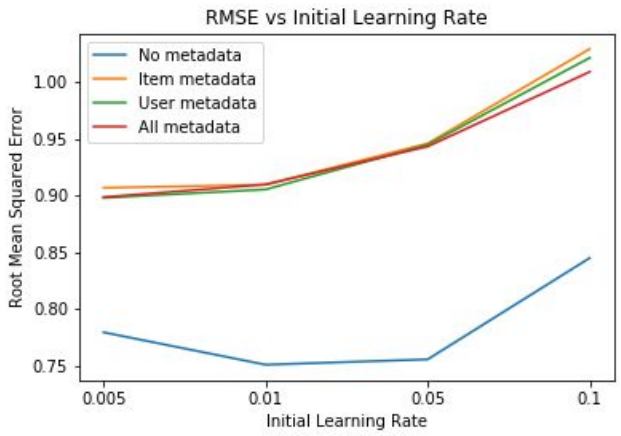
Coverage using ALS

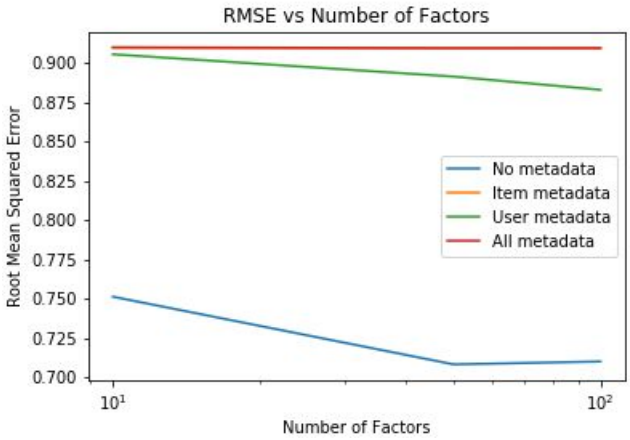
Metric	User Coverage	Catalogue Coverage
ALS	85%	53%

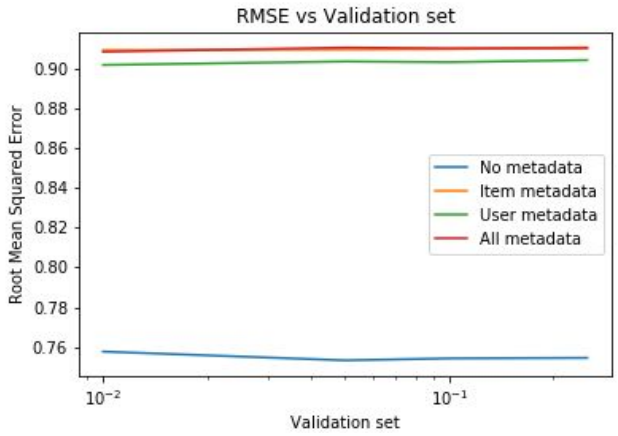
9. Factorization Machines

We implement factorization machines using pyFM, a Python implementation of FM designed by Google's Corey Lynch that uses stochastic gradient descent with adaptive regularization. We tried tuning three hyperparameters separately: the size of the validation set (with 10 latent dimensions and initial learning rate of 0.01), the number of latent dimensions (with a validation set that was 1% the size of the training set and number of latent dimensions as 10), and the number of latent dimensions (with validation set that was 1% the size of training set and initial learning rate of 0.01). We experimented with different iterations of the model where we included various mixes of data and metadata: we tried including all the metadata, only the user metadata, only the restaurant metadata, and neither.

Our questions were simple: would adding user and restaurant metadata improve the model, and was one type of data more valuable than the other?







Number of Factors: # of latent dimensions considered.

Note (graphs): The orange line representing only *Item metadata* is overlapped by the red *all metadata* line in the case of RMSE vs Validation set and RMSE vs Number of Factors graphs.

RMSE on Test Dataset

Varying Validation set size (0.01 initial learning rate, 10 latent dimensions)				
Validation Set (proportion)	No Metadata	Item Metadata	User Metadata	All Metadata
0.01	**1.220**	1.370	1.360	1.369
0.05	**1.227**	1.371	1.405	1.375
0.1	**1.221**	1.384	1.400	1.369
0.25	**1.230**	1.370	1.374	1.368

Varying Initial Learning rate (1% validation set, 10 latent dimensions)				
Initial Learning Rate	No Metadata	Item Metadata	User Metadata	All Metadata
0.005	**1.235**	1.377	1.350	1.364
0.01	**1.222**	1.382	1.357	1.380
0.05	**1.247**	1.504	1.391	1.507
0.1	**1.302**	1.831	1.399	1.768

Varying Latent Dimensions (1% validation set, 0.01 Initial Learning Rate)				
Number of Latent Dimensions	No Metadata	Item Metadata	User Metadata	All Metadata
10	**1.222**	1.382	1.357	1.380
50	**1.234**	1.382	1.330	1.393
100	**1.238**	1.370	1.337	1.370

Generally, we found that the addition of the metadata did not improve the model, and likely led to overfitting. Now, this is not to say that adding some of the metadata might have made the model better, but it's clear that haphazardly adding large swaths of it hurt more than helped.

Examining our results further, it seems that restaurant metadata is very slightly less useful than user metadata. This could be because the context of who the person making the review matters more than the context of the restaurant they're reviewing, but it could also be that there were more continuous variables in the user metadata, and more one-hot encoded data in the restaurant metadata. It's also reasonable to argue that the difference between the two isn't significant enough to discern any real difference in their utility.

It is quite possible that *some* less aggressive infusion of metadata would yield better results, and future further experimentation is warranted. But as with many machine learning models, we can confidently say that there's a limit: at a certain point, adding more metadata hurts the model.

However, our best-performing model here does beat our baseline runs of augmented least squares and our baseline bias model, so there is value to an FM model.

Overall, if our null hypothesis is that metadata doesn't help the FM model, we can't *definitively* say we fail to reject it: but we certainly cannot reject it based on what we've seen.

Less Prolific Users and Less Popular Businesses

We also explored how the model would perform when we test it over less prolific users and less popular businesses (restaurants).
Less prolific user - A user who rated less than or equal to 5 times
Less popular business - A business which has less than or equal to 100 ratings

	Less Prolific Users	Less Popular Businesses
RMSE on test	1.238	1.332

The RMSE value related to less prolific users is close to the other RMSE values of the model and so, we can say that this model predicts equally well for the less prolific users compared to the other users. However, the RMSE value of

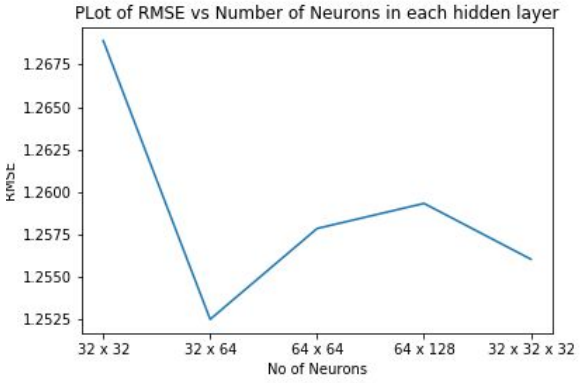
less popular businesses seems to be higher than the overall RMSE values of no metadata case. This gives an idea of the importance of number of ratings that each business has.

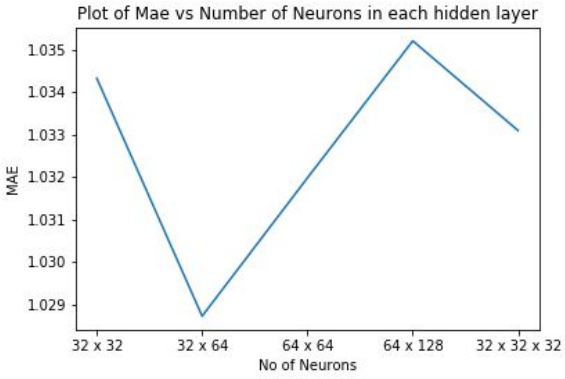
10. Wide & Deep

To try a deep learning approach for recommending restaurants to users, we used pytorch - widedeep which is based on Google's Wide and Deep algorithm. To give an overview of what wide and deep algorithm we define the wide component as a generalized linear model of the form $\boldsymbol{y = w^T x + b}$ where y is the prediction, x is a vector of **d** features, **w** are the model parameters and b is the bias. The deep component is defined as a feed forward neural network which takes as inputs as either one hot encoded input features or sparse categorical features which have high dimensions which are first converted into dense low dimensional embeddings. The wide components and deep components are combined using a weighted sum of their output log odds as the prediction, which is then fed to one common logistic loss function for joint training.

We have used user and business metadata as the input to both the models. In particular, the one hot encoded features (attributes and categories) are a part of the generalized linear model whereas the continuous features from user metadata are used in the deep model. We have also used embeddings of the city which can be effective if there are a lot of feature categories.

The results are:





Varying No of Neurons in each hidden layer		
No of neurons	**RMSE**	**MAE**
32 x 32	1.268	1.034
32 x 64	**1.252**	**1.028**
64 x 64	1.257	1.031
64 x 128	1.259	1.035
32 x 32 x 32	1.256	1.033

	Less Prolific Users	Less Popular Businesses
RMSE on test	1.251	1.37

We also attempted to calculate user and catalogue coverage using the predictions from the wide and deep model. Since we have approximately 32,000 users and 5500 restaurants, predicting rating for each restaurant for each user was demanding resources. So we took 1000 users and 500 restaurants to calculate the coverage and generalized the method so that given the resources, we can run it on the entire dataset. Coverage results are:

	User Coverage	Catalogue Coverage
ALS	97%	12%

We have also generalized the code for giving top 10 restaurants for each user which he or she has not visited. For a sample user, the recommendation looks like:

Restaurants already visited

user_id	Restaurant
1	Capriotti's Sandwich Shop
1	Wicked Spoon
1	Taco Bell
1	Lotus of Siam
1	Delhi Indian Cuisine
1	Pho Vegas
1	The Buffet
1	Mr Sandwich

Top 10 restaurants recommended by the model

user_id	Restaurant
1	Fountains of Bellagio
1	Blue Ribbon Brasserie - Las Vegas
1	Eatt Gourmet Bistro

1	The Corndog Company LV
1	The Venetian Las Vegas
1	Azuza Hookah Lounge & Cafe
1	Tina's Gourmet Sausage House
1	Pinball Hall Of Fame
1	The Steakhouse at Treasures
1	Estiatorio Milos

11. Content-Based Recommendation For Cold Starts Using Natural Language Processing

Reviews are an important source of information depicting the 'true' nature of the restaurant. Not only can it improve the recommendation results but can also help in the cold start problem. To leverage them, we select the top 20 reviews used for each restaurant in Las Vegas for 2018. Each review is weighted by its 'useful' and 'cool' votes. All the reviews are fed to our language model, Infersent, which generates low perplexed embedding of reviews. It understands the important features of the reviews using BiLSTM architecture. How we calculate a representation of a restaurant is as follows:

$$Restaurant_j = w_j * embedding_j / \sum w_k * embedding_k$$

where k runs from 1 to total restaurants and weight (w) = # of useful + # of cool votes.

This weighted review embedding is inserted into an annoy index to improve throughput of the recommendations performance. Below are a test case of content-based recommendation results:

***Input**: This was the best pizza place ever*
***Recommendation**: "Pizzaria", "Hungry Howie's Pizza", "Can't Dutch This", 'Sababa Mediterranean Grill', "Joey's Pizza", "LaBella's Pizzeria", 'Marias Taco Shop', 'Pizza Hut', 'Rigos Taco', "Noble Roman's Pizza Slots A Fun"*

***Recommendation**: Worst ambience ever. Never go to this place*
***Output**: "Subway", "Cadillac To Go", "17 South Booze & Bites", "Hazel M Wilson Dining Commons", "Pizza Hut", "Simon & Joe's Eatery", "Sofia's Pizza", "Subway", "McDonald's"*

12. Summary of Results

Model	RMSE
Bias Baseline	1.24
ALS	1.65

FM	**1.22**
Wide and Deep	1.252

The Factorization Model that we developed has performed better without any metadata compared to that with user/item/all metadata. Also, it performs better than both the Baseline and ALS but differs marginally from Wide and Deep

The best RMSE for test set obtained is 1.22 in the case of model with no metadata. The parameter settings for the corresponding model are 0.01 validation set proportion, 0.01 initial learning rate, and 10 latent dimensions.

13. Further Questions/Concerns

I. *Potential Watchouts:*

1. Scalability: Would we perform batch-processing for training? If not, how can we approach a near real-time recommendation.
2. Sparsity: How to make the computation efficient with increasing data sparsity?
3. Grey - Sheep User Problem: How to provide recommendation to unpredictable user's activity?
4. Synonymy - How to prevent not recommendation the semantically same restaurant?
5. Shilling Attack - How to make recommendation robust against attacks to throw off the model?
6. Adaptability - How to construct a model which quickly adapts to new user's taste?
7. Discover - How to discover the latent user's taste and provide a new/not-so-popular items yet highly recommendable item.
8. Social Sensing - How to incorporate the signals from the user's social media activities to improve the robustness of the model?
9. Location-aware recommendation - How to adapt user's taste in a new location from the past location's taste profile.
10. Feature engineering - Improving subsets of metadata to test with FM model.
11. Hybrid recommendation - Connecting Content-based for rating classification along with Collaborative models.

II. *Need For Business Collaboration:*

1. Work with PMs to undertake better feature engineering.
2. A/B testing.
3. Budgeting to scale up the infrastructure to test current state-of-the-art models.
4. Designing policy framework for enhanced diverse recommendations, targeted advertising etc.