PERSONALIZATION FINAL PROJECT

Priyanka Vijay Lahoti, Tanvi Gautam Pareek, Arusha Kelkar
pvl2111@columbia.edu, tgp2108@columbia.edu ,ak4432@columbia.edu

Problem Statement

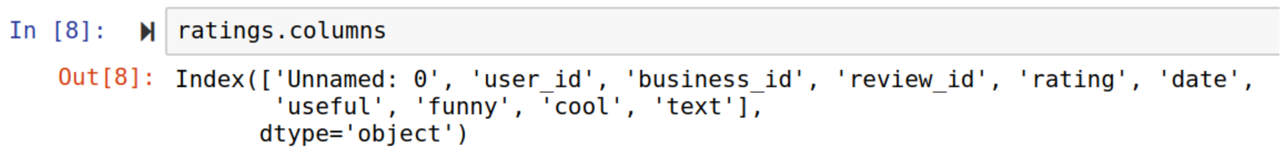
Predict the last rating of an active user i.e. for user with 5 or more reviews, to hold out their final review (by date) and make a prediction on the rating of this final review.

Business Objective

The objective is to predict the last rating of each active user. Three models have been implemented and user-item bias baseline model have also been implemented to compare how well the models are predicting. The 3 models implemented are ALS, deep learning model using embedding layers and Factorization Machine (LightFM).

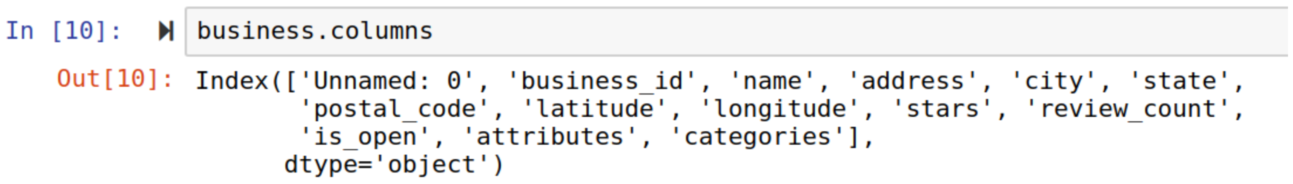
Dataset

The Yelp dataset can be downloaded from https://www.yelp.com/dataset/challenge .Ratings, User and Business dataset is used to predict the last rating for each active user. The Review dataset contains around 6.6 million ratings given by 1637139 unique users for 192607 unique businesses. Dataset has explicit ratings on the scale of 1-5. Columns in Ratings dataframe are:



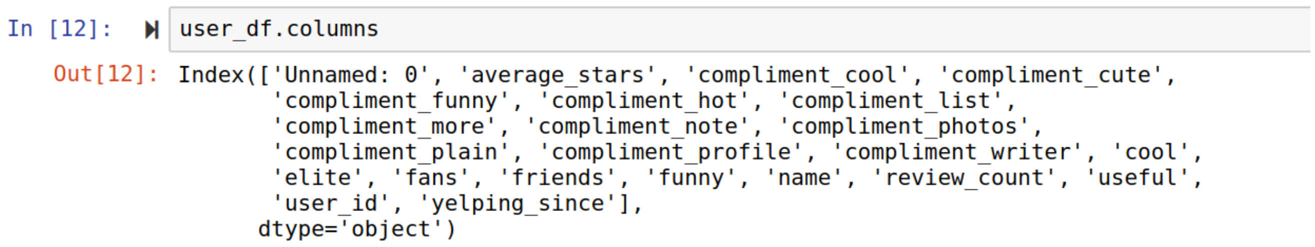
```
In [8]:  ratings.columns

Out[8]: Index(['Unnamed: 0', 'user_id', 'business_id', 'review_id', 'rating', 'date',
               'useful', 'funny', 'cool', 'text'],
              dtype='object')
```

Business dataset contains the information about all the businesses. Columns in Business Dataset are:



```
In [10]:  business.columns

Out[10]: Index(['Unnamed: 0', 'business_id', 'name', 'address', 'city', 'state',
                'postal_code', 'latitude', 'longitude', 'stars', 'review_count',
                'is_open', 'attributes', 'categories'],
               dtype='object')
```

User dataset contains the information about all the users. Columns in User Dataset are:



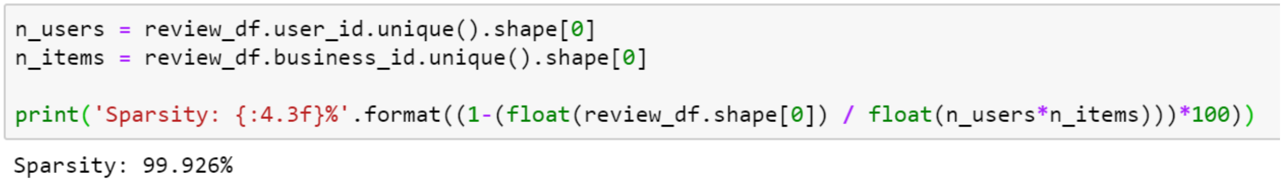
```
In [12]:  user_df.columns

Out[12]: Index(['Unnamed: 0', 'average_stars', 'compliment_cool', 'compliment_cute',
                'compliment_funny', 'compliment_hot', 'compliment_list',
                'compliment_more', 'compliment_note', 'compliment_photos',
                'compliment_plain', 'compliment_profile', 'compliment_writer', 'cool',
                'elite', 'fans', 'friends', 'funny', 'name', 'review_count', 'useful',
                'user_id', 'yelping_since'],
               dtype='object')
```

Sampling and EDA:

The most popular categories in the business dataset are ‘Restaurants’, ‘Food’ and the cities with the maximum ratings are ‘Las Vegas’ and ‘Toronto’. From the Business dataset select business having categories ‘Restaurants’, ‘Food’ and which are present in the cities ‘Las Vegas’ and ‘Toronto’. Merge this sampled business data with ratings data using business_id and merge this with user data using user_id. Finally, in this sampled data select only active_users (users who have rated more than 5 businesses) and their corresponding ratings. Total number of active users in the sampled data is 66968.

Business dataset contains column categories, which is used to create new business features. For each category in the business, a new column is created if the category is present in at least 10000 businesses.

Sparsity of the sampled data is: 99.926%



```
n_users = review_df.user_id.unique().shape[0]
n_items = review_df.business_id.unique().shape[0]

print('Sparsity: {:4.3f}%'.format((1-(float(review_df.shape[0]) / float(n_users*n_items)))*100))
```

```
Sparsity: 99.926%
```

The goal of the project is to predict last rating for each active user. First group the sampled data by user_id and sort by date for each user, the latest rating for each user will be in test data and remaining will be in the training data.

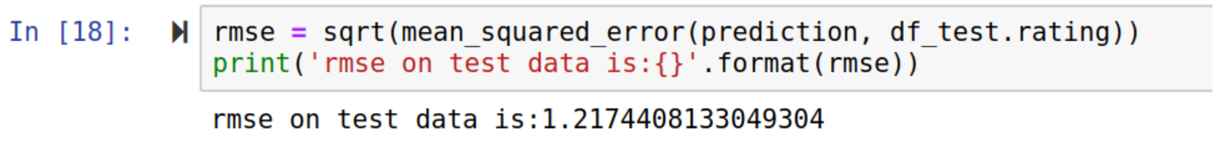
MODEL:

1. **Baseline Model**

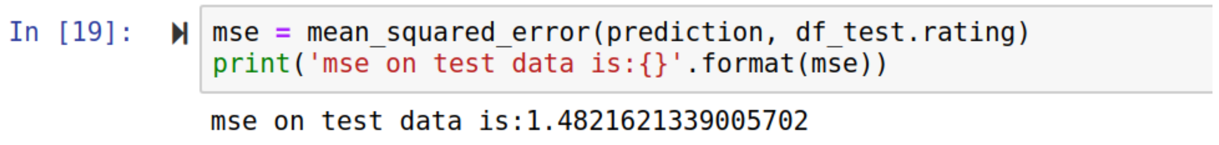
BaselineOnly model from Surprise package is used to build ‘user-item bias baseline’ model. μ is the overall average rating. A baseline estimates for an unknown rating r_ui is denoted by b_ui calculated as: **b_ui = μ + b_u + b_i.**

The parameters b_u and b_i indicate the observed deviations of user u and item i, respectively, from the average.

Metric rmse and mse is used to evaluate model performance.



```
In [18]:  rmse = sqrt(mean_squared_error(prediction, df_test.rating))
          print('rmse on test data is:{}'.format(rmse))

          rmse on test data is:1.2174408133049304
```



```
In [19]:  mse = mean_squared_error(prediction, df_test.rating)
          print('mse on test data is:{}'.format(mse))

          mse on test data is:1.4821621339005702
```

2. ALS (Model-based collaborative filtering)

In model-based methods, machine learning and data mining methods are used in the context of predictive models. **Matrix Factorization algorithm** is used to reduce latent space of the algorithm. Matrix factorization algorithm factorizes a huge user-item interaction matrix R into two smaller rank matrices U and V sharing a joint latent vector space (U: User feature matrix and V: item feature matrix),

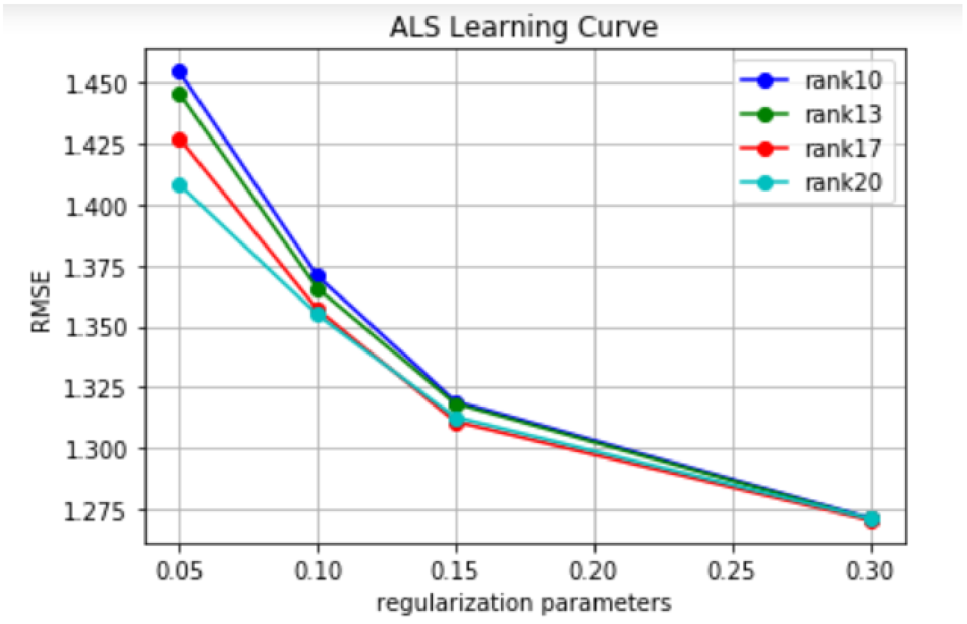
such that R=UV^T , R has dimensions m*n (m: number of users, n: number of items)

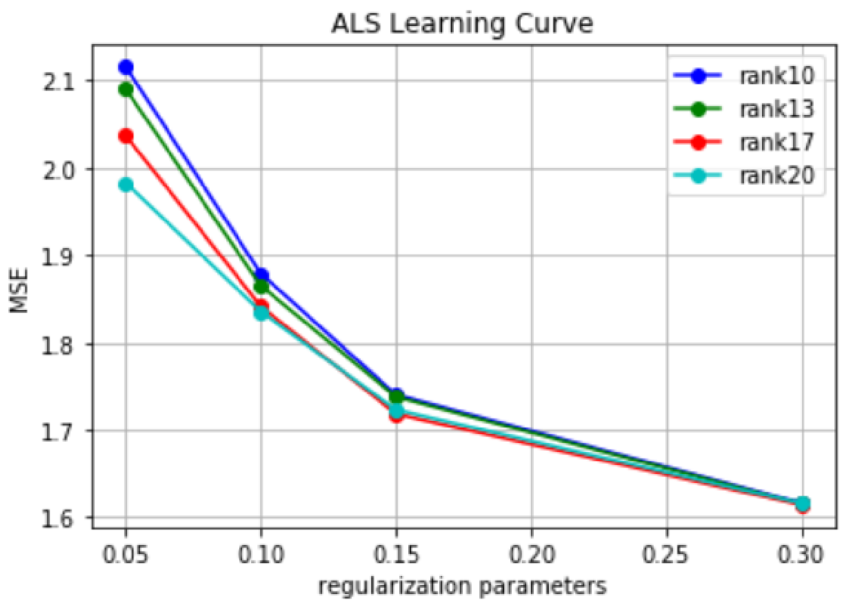
U has dimensions m*r and V has dimensions n*r. (r: number of latent factors)

ALS (Alternating Least Squares) method is used to find matrix factorization. In ALS, 1st U is kept as constant and the optimization equation is solved for V. Then V is kept as constant and the optimization equation is solved for U. These two steps are iterated to convergence.

Implementation:

1. Loading the sample
2. ALS function from pyspark library in python is used for implementation.
3. **Hyperparameters** tuned for the model are: **number of latent factors and regularization parameter.** Since RMSE is primary evaluation metric, the set of hyperparameters for which model gives lowest RMSE values for test dataset is considered as the best model. Attached below is the plot of RMSE vs number latent factors(rank) and regularization parameter. The number of latent factors=17 and regularization parameter=0.3 are selected to build the final model.





4. Ratings for each user in test data is predicted using best model.
5. Metric rmse and mse is used to evaluate model performance.

```
print('The ALS model gives rmse value of {} and mse value of {} on test data'.format(RMSE,MSE))

The ALS model gives rmse value of 1.2704955345064337 and mse value of 1.6141589032007888 on test data
```

3. NEURAL NETWORK USING EMBEDDINGS:

Use of Embeddings to predict the latest rating given by the user

The deep learning model using Embeddings and fully connected layers is built to predict the latest rating of a user for a business.

A **neural network** is a simplified **model** of the way the human brain processes information. It works by simulating a large number of interconnected processing units that resemble abstract versions of neurons. The processing units are arranged in layers.

In this model we are going to use embeddings to represent the various features of the dataset.

An embedding is a mapping from discrete objects, such as words or ids of books in our case, to a vector of continuous values. This can be used to find similarities between the discrete objects, that wouldn't be apparent to the model if it didn't use embedding layers.

The training dataset consists of the information for the users from cities 'Las Vegas','Toronto' and the ratings given by them to the restaurants in these two cities.

The model has the following structure:

1. Input: Input consisting of the users, business units and the various other features like stars,Restaurant take out, good for kids,etc.

2. Embedding layers: Embeddings for users, businesses and the above mentioned features.
3. Neural Network: Constructed a neural network using the created embedding layers and adding **fully connected layers** having activation function relu(Rectified Linear Unit).
4. Modelcheckpoint was used to store the model which gives the least validation loss in all the epochs while training the model. Validation loss is the value of cost function for your cross-validation data. Test data has been used as validation data in the model.
   Callbacks used to store the models and subsequently to store the best model.
5. Optimizer used: Adam
   Adam is an optimization algorithm that can be used instead of the classical stochastic gradient descent procedure to update network weights iterative based on training data.
6. Metric used: mse(Mean squared error)
   The **mean squared error** (**MSE**) of an estimator measures the average of the squares of the **errors** — that is, the average **squared** difference between the estimated values and what is estimated. Here the average squared difference between the predicted values of rating and the actual rating given by the user for the latest rating case is calculated
7. Learning rate:
   The **learning rate** is a tuning parameter in an optimization algorithm that determines the step size at each iteration while moving toward a minimum of a loss function. Tuning of the learning rate was done to get the least value of validation loss value.

Learning Rate	Validation Loss
0.0001	1.3194
0.001	1.31820
0.005	1.30774
0.01	1.31605

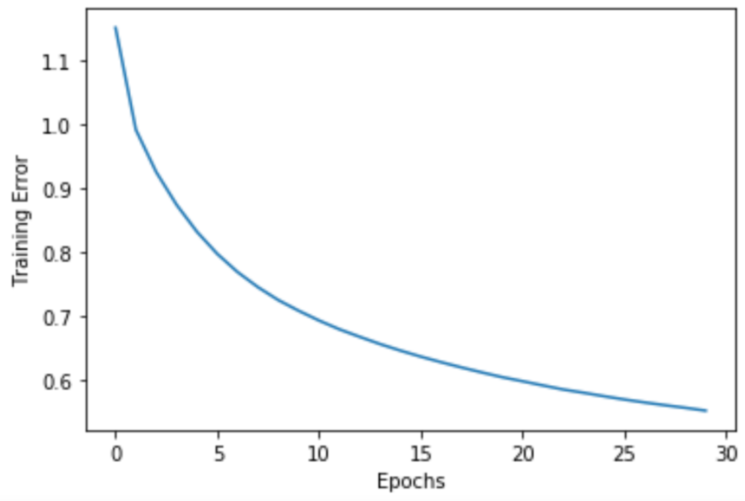
8. Batch_size: Batch size of 64 was used for training the model and the model was run for 30 epochs.
9. Root Mean Square Error (RMSE):

$$RMSE = \sqrt{\frac{\Sigma^{n}{}_{J=1}(r_{uj}-o_{uj})^2}{n}}$$

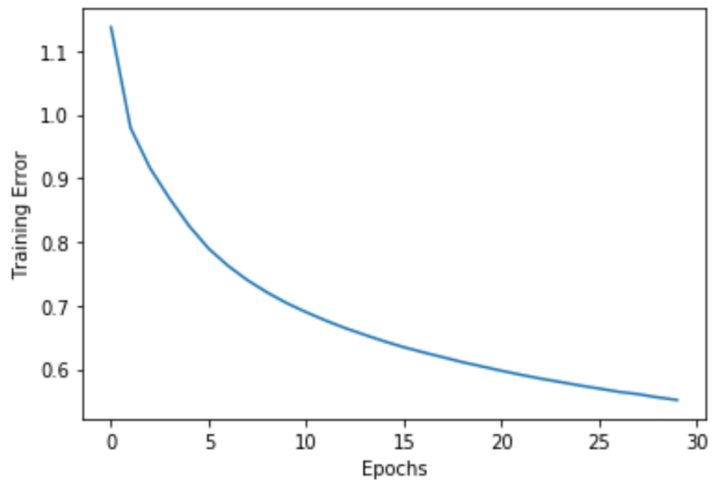
   Where r_{uj} is the predicted latest rating for user j.
   And o_{uj} is the actual latest rating of the business given by user.

TRAINING ERROR VARIATION

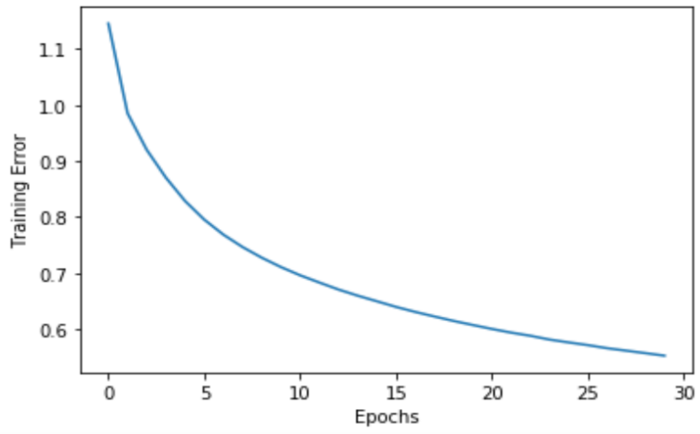
- Training error variation across the epochs for the learning rate 0.001:



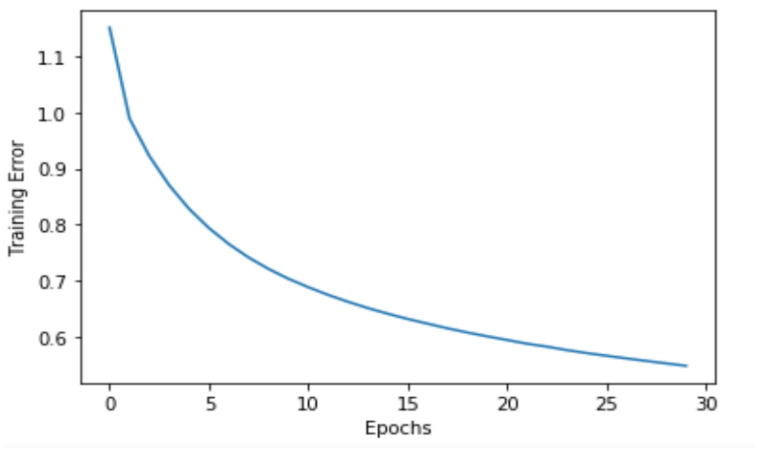
- Training error variation across the epochs for the learning rate 0.005:



- Training error variation across the epochs for the learning rate 0.01:



- Training error variation across the epochs for the learning rate 0.0001:



10. The learning rate of 0.01 is used to build the final model which gives rmse 1.1471 on test data.

ACCURACY

Predictions for the latest rating given by the user were made on the test dataset.

1. **Mean Squared Error**

   The mean squared error (MSE) was calculated for both 0.001 and 0.005 learning rates:

Learning Rate	MSE
0.0001	1.319
0.005	1.307
0.001	1.325
0.01	1.316

2. **Root Mean Square Error:**

   The Root mean squared error (RMSE) was calculated for both 0.001 and 0.005 learning rates:

Learning Rate	RMSE
0.0001	1.1486
0.005	1.413
0.001	1.1513
0.01	1.1471

Conclusion
We can see that these RMSE values are less than the baseline model considered earlier. Thus the prediction of the latest rating of a user can be well achieved by using a neural network consisting of embedding layers.

4.LIGHTFM MODEL

A factorization machine is a general-purpose supervised learning algorithm that can be used for both classification and regression tasks. It is an extension of a linear model that is designed to capture interactions between features within high dimensional sparse datasets economically. Factorization machines are a good choice for tasks dealing with high dimensional sparse datasets, such as click prediction and item recommendation. They are a state of the art solution to problems including multiple entity types. They are similar to other collective approaches in that they allow for multiple relationships between multiple entities. Feature generation for factorization machines one-hot-encodes users and items, and uses the rating variable itself as a target variable, much like regression or classification. We implemented FM using lightfm https://lyst.github.io/lightfm/docs/home.html .

LightFM is a Python implementation of a number of popular recommendation algorithms for both implicit and explicit feedback. It also 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 the latent representations of their features, thus allowing recommendations to generalise to new items (via item features) and to new users (via user features). Based on our dataset, item metadata stands for all business features and user metadata stands for user features.

Item features

Most features had missing values across restaurants so we picked number of stars the restaurant was rated (0-5 scale), review counts of the restaurants, and the categories of the restaurants as item features. Usually each item (restaurant) has an average of 10 categories/tags assigned by Yelp, and there were 436 tags in total across all restaurants, such as seafood, vegetarian, breakfast & brunch, bars and so on. We decided to select top 60 tags with highest popularities since including some tags that only appear a few times out of more than 10k restaurants would add more noise in the recommender. Furthermore, TF-IDF (*term frequency-inverse document frequency*) values for each tag was calculated which would were used as weights in model fitting later.

User features
We used all the features in the user dataframe for user features except for cool, funny and year.

Implementation:

1. Three separate sampled data frames were used namely user_df,business_df and review_df(ratings).
2. Dataset was built from the review dataframe using user_id and business_id.
3. Item features were used to fit the dataset using .fit_partial() . Similarly, user features were used to fit the dataset using .fit_partial().
4. According to the condition regarding predicting the latest rating of a user, we made our own train and test dataset where test dataset has the last review given by the user and we have to predict the rating for the same.
5. Sparse matrix for train and test dataset were built using .build_interactions() where rows represented the user_id, columns represented business_id and the value represented the rating.
6. Item features were built using .build_item_features() using the business dataframe.
7. User features were built using .build_user_features() using the user dataframe.
8. Following parameters were used to initialize the model:
    - Loss : Loss functions 'logistic', 'bpr', 'warp' were tested and loss function 'warp' was found to be the best and hence used.
    - Random_state: The seed of the pseudo random number generator to use when shuffling the data and initializing the parameters. We set it to '123'.
    - No_components: The dimensionality of the feature latent embeddings.
    - Learning_rate: Initial learning rate for the adagrad learning schedule.

9. Training interactions were used to fit the model along with the parameter epochs.
10. **Precision_at_k** and **AUC**(Area under curve) were calculated for both train and test interactions. Precision_at_k is the primary metric here.
11. **Learning rate** and **number of components** were tuned and graph for the same has been made and can be seen below.
12. Steps 8 to 11 were performed again with features/metadata i.e. train interactions were used to fit the model along with item features, user features and epochs.

Evaluation Metrics:

precision_at_k

Measure the precision at k metric for a model: the fraction of known positives in the first k positions of the ranked list of results. In our case k =5. For example, if you measure precision_at_5 but there is only one positive item, the maximum score you can achieve is 0.2.

Auc_score

Measure the ROC AUC metric for a model: the probability that a randomly chosen positive example has a higher score than a randomly chosen negative example.

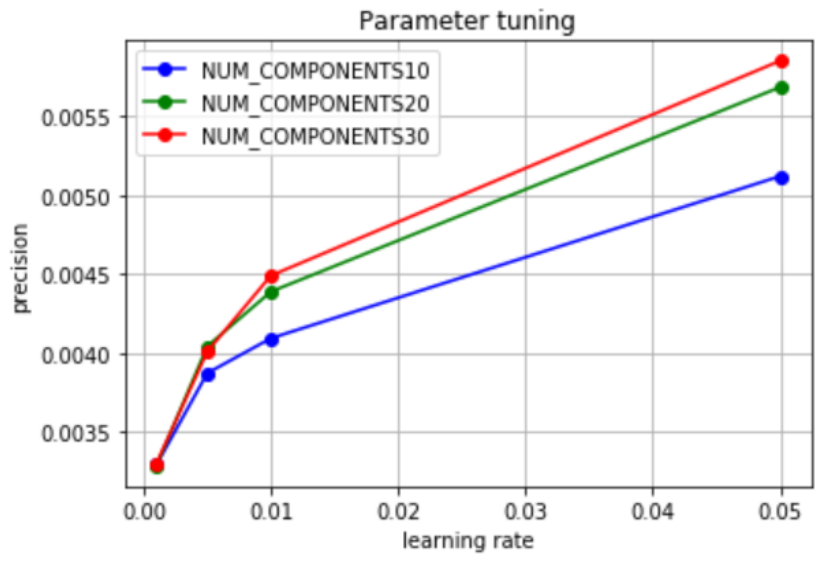
In LightFM, the AUC and precision@K routines return arrays of metric scores: one for every user in your test data. Therefore, we have taken an average of these to get a mean AUC or mean precision@K score: for example if some of the users have score 0 on the precision@5 metric, it is possible that the average precision@5 will be between 0 and 0.2(given there was only one positive item).

LIMITATIONS OF LightFM;
LightFM does not have a function to predict ratings. Instead it gives a recommendation score for every user in the test data.

RESULTS:

The results for user_item interactions without features and one with features have been plotted below. Two parameters that were tuned are learning rate and number of components. As the primary metric was precision_at_k plot best model or best parameters have been obtained by seeing that model that had the highest precision_at_k value.

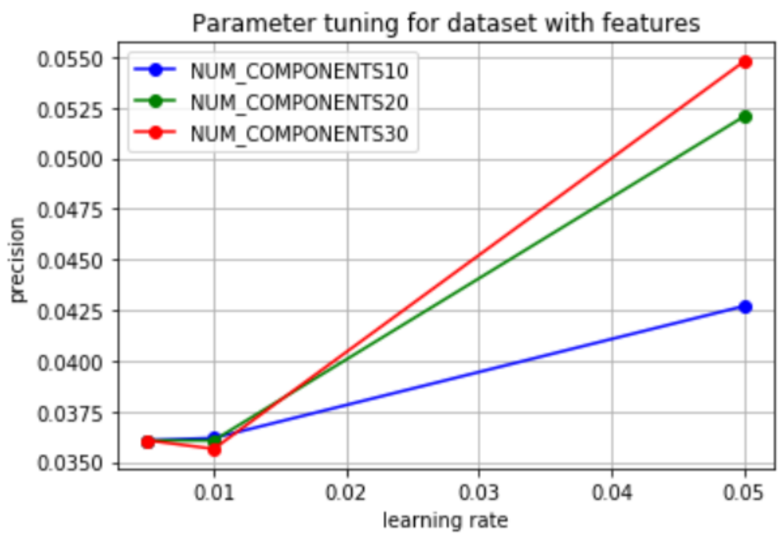
- For user_item interactions without features/metadata



```
The best model has 30 NUM_COMPONENTS and learning_rate = 0.05
```

Precision_at_5 for the best model was 0.59% and AUC for this was 0.9256.

- For user_item interactions with features/metadata



```
The best model has 30 NUM_COMPONENTS and learning_rate = 0.05
```

Precision_at_5 for the best model was 5.48% and AUC for this was 0.8888.

Conclusion:

In this case, we compared LightFM model (with features) with LightFM model (without feature) for same number of components (30) and learning rate 0.05, model with features outperformed model without features. The former gave Precision_at_5:0.0548 AUC:88.88% and later gave Precision_at_5: 0.0059 AUC: 0.9256

FINAL RESULTS:

Model	**Parameter**	**Metric**	**Value of Metric**
Baseline		RMSE	1.217
		MSE	1.482
ALS	Number of latent factors=17	RMSE	1.2704
	Regularization parameter=0.3	MSE	1.614
Deep Learning	Learning Rate: 0.01	RMSE	1.1471
		MSE	1.316
LightFM(Without Features)	Learning Rate: 0.05 Components number:30	Precision_at_5	0.0059
		AUC	0.9256
LightFM(With Feature)	Learning Rate: 0.05 Components Number: 30	Precision_at_5	0.0548
		AUC	0.8888

FUTURE WORK AND POTENTIAL WATCH OUTS

Potential watch outs

- The scalability is an issue. As the sample size increases, the time to fit the model increases for both the models. We had to run one of our models on Google Colab. One watchout for colab is that it runs into runtime error after 6 hours.
- Sparsity is high i.e- the percentage of people who have rated the businesses is quite low.
- Every time different features were tried and tested, they had to be fitted again.

Future work

In terms of design choices, we can consider context-aware features like location by using latitude and longitude. For example, users living in India and users living in USA will have some differences in the type of recommendations they receive.
The model can be further improved by using the text column in ratings dataframe by using natural language processing.
If machine with larger memory would have be there, we could have not restricted ourselves to top categories and top two cities.