CSE 587 | Data Intensive Computing

Project – Phase 2

Prashant Godhwani

Akhilesh Goriparthi

Problem Statement

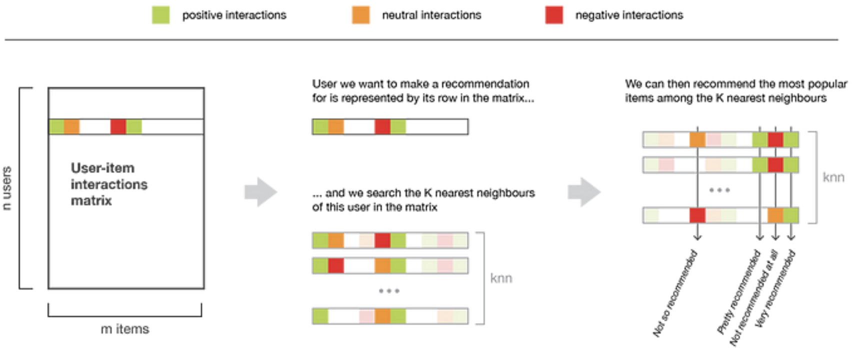
With more and more places to eat being opened every day, people want to make informed decisions and have a good experience when they do choose a restaurant. Yelp, an open crowd-source platform, contains information about restaurants including but not limited to reviews, categories, locations and other relevant features that can be leveraged to recommend users places to eat based on their preferences. The challenge is how to use this big data to make reasonable inferences and model it to build a recommendation system that can save people the hassle and energy while ensuring a better experience.

Link to Dataset - https://www.yelp.com/dataset

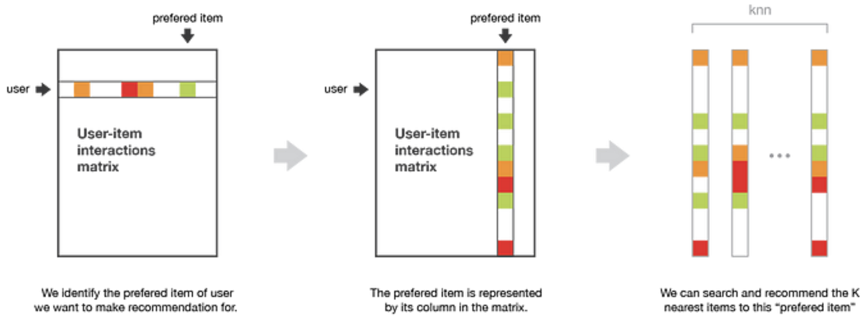
Introduction to Recommendation Systems

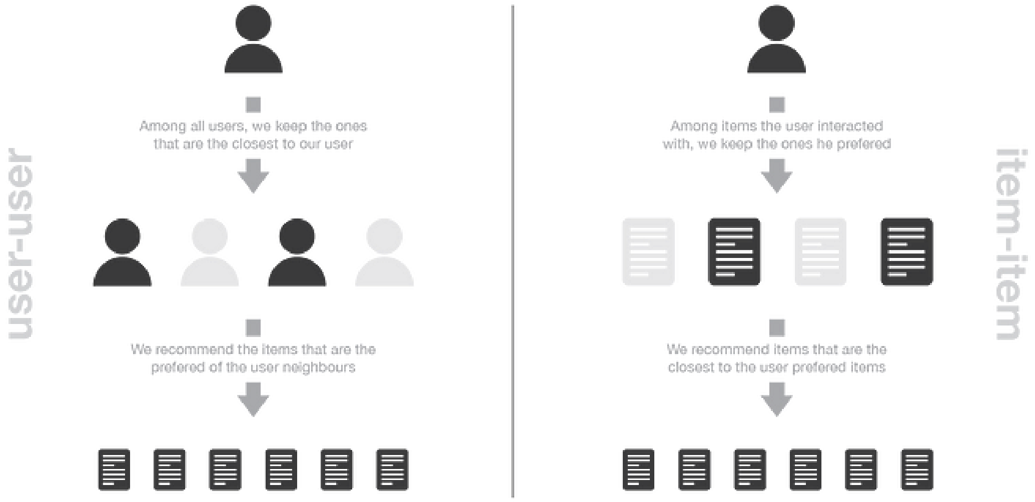
Our problem statement deals with building a Recommendation System to recommend Restaurants to users based on the crowd-sourced Yelp Dataset. There are 2 major strategies when it comes to Recommendation Systems –

1) **Collaborative Filtering** – Recommends based on past interactions between Users and Restaurants. This is further of 2 types –
    a. **User based Collaborative Filtering** – Recommends based on preferences of other similar users. For example – if user A rated Restaurant A highly and user A and B are similar, then the algorithm may recommend Restaurant A to user B.
    Useful when users have a lot of interactions with the Restaurants. As in our case through ratings and reviews.


positive interactions
neutral interactions
negative interactions
n users
User-item interactions matrix
m items
User we want to make a recommendation for is represented by its row in the matrix...
... and we search the K nearest neighbours of this user in the matrix
knn
We can then recommend the most popular items among the K nearest neighbours
knn
Not so recommended
Pretty recommended
Not recommended at all
Very recommended

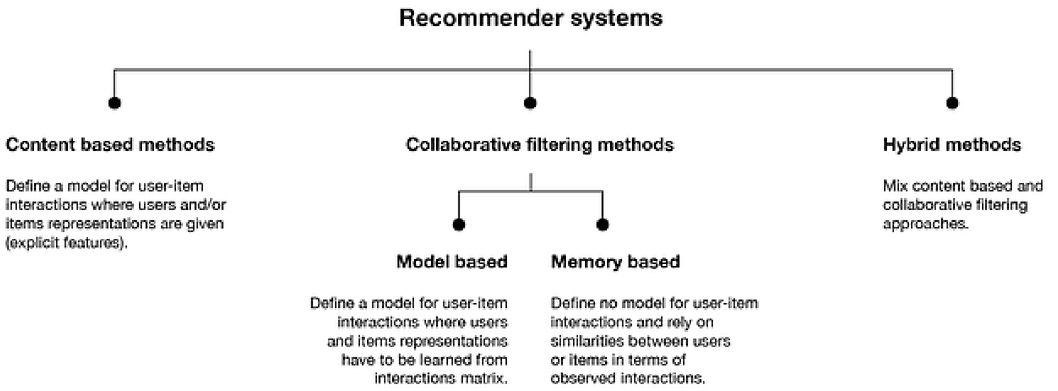
    b. **Item based Collaborative Filtering** – Recommends based on similarity of items with the items user has rated highly. For example – if user A rated Restaurant A highly and Restaurant B is like Restaurant A, then we recommend Restaurant B to user A.


prefered item
user
User-item interactions matrix
We identify the prefered item of user we want to make recommendation for.
prefered item
user
User-item interactions matrix
The prefered item is represented by its column in the matrix.
knn
We can search and recommend the K nearest items to this "prefered item"


user-user
Among all users, we keep the ones that are the closest to our user
We recommend the items that are the prefered of the user neighbours
item-item
Among items the user interacted with, we keep the ones he prefered
We recommend items that are the closest to the user prefered items

2) **Content Filtering -** Recommends based on users' past interactions with Restaurants and characteristics of those Restaurants.
For example – If user A likes Indian cuisine, rates restaurants highly within a particular price range, if it has live music, and is in Downtown. This algorithm will recommend them restaurants with similar cuisine, in downtown, in that price range and have live music.

3) **Hybrid Filtering –** Recommends by using both collaborative and content based filtering approaches.


Recommender systems
Content based methods
Define a model for user-item interactions where users and/or items representations are given (explicit features).
Collaborative filtering methods
Hybrid methods
Mix content based and collaborative filtering approaches.
Model based
Define a model for user-item interactions where users and items representations have to be learned from interactions matrix.
Memory based
Define no model for user-item interactions and rely on similarities between users or items in terms of observed interactions.

Evaluation Metrics we are considering

Below are the evaluation metrics we are considering for measuring the correctness of our recommendations and the performance of our model.

1) **RMSE –** stands for Root Mean Squared Error. In context of recommendation systems, we try to predict ratings for restaurants that the user hasn't visited. RMSE gets the squared root of average of squared differences between actual and predicted ratings. The lower the RMSE, the better the recommendations.
2) **MAE –** stands for Mean Absolute Error. Like RMSE, we find out the difference between the actual ratings and the predicted ratings for the restaurants. For example – if MAE after running one of the recommendation algorithms came 0.028, then that means that for a rating of 3, our algorithm predicted 3 (+/-) 0.028, which is not that bad.
3) **FCP –** stands for Fraction of Concordant Pairs. This is a statistical measure to analyse the accuracy/goodness of the recommendation system. FCP measures how many times the algorithm recommended correct restaurant relative to other restaurants. More the FCP, better the recommendation. A FCP of 1 means that the recommendation algorithm correctly predicts the user preferences and suggests based on that. In other words, if the algorithm recommends item A over item B, and the user rates item A higher than item B, then this is considered a "concordant pair".
4) **MSE -** stands for Mean Squared Error. This is like RMSE and MAE and measures the error in predicted ratings and the actual ratings. The lower the MSE, better the recommendations.

What did we do?

Work to train the model –

In our dataset, we had the ratings users gave to restaurants and we could predictions using that. Along with that we calculated the sentiment score of the reviews by analysing the review text and generating a score between -1 to 1 where 1 means positive, 0 means neutral and -1 means negative sentiment.

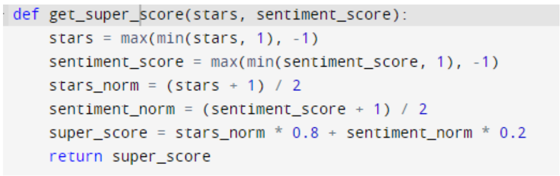
Inspired from (1), We decided to make use of the ratings and the sentiment score and combine that to a super_score and employ that score for our recommendations.

How did we calculate Super Score?

We took the sentiment score, normalized it to lie between (0 – 1), took the rating, normalized it (0 – 1) and then combined to form a super score based on –

Super_Score = 0.7 * rating + 0.3 * sentiment_score

Then we normalized the super score to lie between 0 and 1 and used that for all our recommendations.



```
def get_super_score(stars, sentiment_score):
    stars = max(min(stars, 1), -1)
    sentiment_score = max(min(sentiment_score, 1), -1)
    stars_norm = (stars + 1) / 2
    sentiment_norm = (sentiment_score + 1) / 2
    super_score = stars_norm * 0.8 + sentiment_norm * 0.2
    return super_score
```

Below are the ML models and algorithms we used for Collaborative, Content and Hybrid recommendation techniques –

Collaborative Filtering

The algorithms we used for Collaborative Filtering are unsupervised or semi-supervised learning algorithms that do not need labelled data for training but learn patterns in the user-restaurant interactions to make recommendations.

1) K Nearest Neighbours

This is the first class of algorithms we explored for our problem statement. This class includes – KNN, KNN Baseline, KNN with Means, and KNN with ZScore.

What is KNN?

The k-nearest neighbours' algorithm, sometimes referred to as KNN or k-NN, is a supervised learning classifier that employs proximity to produce classifications or predictions about the grouping of a single data point.

Why we chose this class of algorithms?

We chose this class of algorithms owing to the limited available computational resources and huge amount of data that we are dealing with. Also, it seemed the most intuitive based on our use case and is easy to implement.

What is expected?

KNN or K Nearest Neighbours looks at the nearest neighbours to determine which restaurant to predict. Here Similarity between users is calculated based on the ratings they give to restaurants. This follows with the algorithm picking the k nearest neighbours and recommending the restaurants they rated highly to the user.

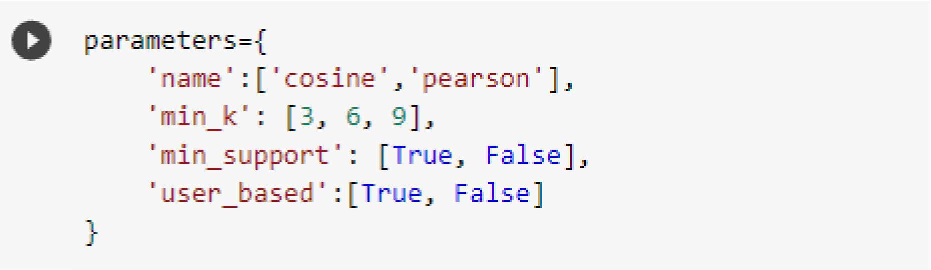
How was it applied?

To apply this algorithm to our data, we need to find out the user-restaurant interaction matrix based on a certain similarity metric. Then the algorithm picks the n-most correlated entries and recommends based on that. In our case, we tried with different values of similarity metric, different values of k, user based or restaurant-based collaborative, and ran the model with all these combinations to figure out which works the best for our use case.

2.1) KNN Baseline

KNN Baseline is an extension of Simple KNN that uses baseline rating instead of normal ratings in an attempt to account for biasness in data as some users only give extreme ratings (low or high). The ratings are then subtracted from the baseline ratings and the deviations are used to plot the similarity between users or restaurants.

Work to tune the model –
The Hyperparameters for KNN are different than in part 1. We tried with several hyper-parameters to see what configuration gave the best performance with KNN. These parameters included –
1) Similarity Metric (name)
2) Number of Nearest Neighbours (min_k)
3) Only consider users with threshold number of ratings (min_support)
4) User Based or Restaurant Based Collaborative (user_based)

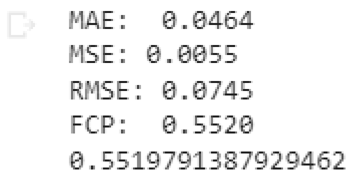


```
parameters={
    'name':['cosine','pearson'],
    'min_k': [3, 6, 9],
    'min_support': [True, False],
    'user_based':[True, False]
}
```

After Running all these parameters with GridSearchCV, we arrived at the below hyper-parameters being the most optimal.

```
{'rmse': 0.08584473631343172, 'mae': 0.0542386929363972}
{'rmse': {'name': 'cosine', 'min_k': 9, 'min_support': True,
'user_based': True}, 'mae': {'name': 'cosine', 'min_k': 6,
'min_support': True, 'user_based': True}}
```

Training the model with highlighted results, we can see that the RMSE decreased and FCP increased. Which is the best we can do for our computational capacity and data.



```
MAE:  0.0464
MSE: 0.0055
RMSE: 0.0745
FCP:  0.5520
0.5519791387929462
```

2.2) KNN With Means

KNN With Means is an extension of Simple KNN that uses mean rating instead of normal ratings to calculate neighbours by taking the weighted average of the k-nearest neighbours for that item. This is better as it requires less computational overhead than KNN Baseline.

Work to tune/train the model –

The Hyperparameters for **KNN With Means** are similar to before.

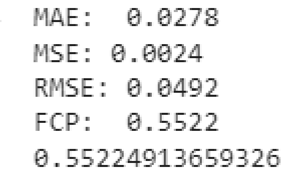
After Running all these parameters with GridSearchCV, we arrived at the below hyper-parameters being the most optimal.

```
{'rmse': 0.09193787271590248, 'mae': 0.05631053452733085}
{'rmse': {'name': 'cosine', 'min_k': 9, 'min_support': True,
'user_based': True}, 'mae': {'name': 'cosine', 'min_k': 6,
'min_support': True, 'user_based': True}}
```

Instead of training algorithm by prioritizing RMSE, we instead use MAE as the criteria and it gives better Results.

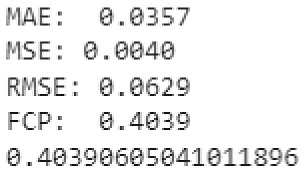
Training the model with highlighted results, we can see that the all the errors decreased and FCP increased. Which is the best we can do for our computational capacity and data.

With MAE-



```
MAE:  0.0278
MSE: 0.0024
RMSE: 0.0492
FCP:  0.5522
0.55224913659326
```

WITH RMSE –



```
MAE:  0.0357
MSE: 0.0040
RMSE: 0.0629
FCP:  0.4039
0.40390605041011896
```

2.3) KNN With ZScore

KNN With Means is an extension of Simple KNN that normalizes the ratings by subtracting the mean rating and dividing by the standard deviation of ratings of a user. This reduces the impact of users who rate too highly or poorly.

Work to tune/train the model –

The Hyperparameters for **KNN With ZScore** are like before.

After Running all these parameters with GridSearchCV, we arrived at the below hyper-parameters being the most optimal.

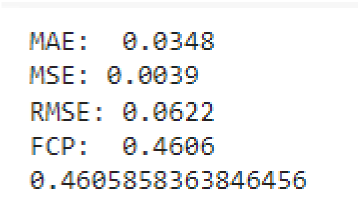
```
{'rmse': 0.09178353153173999, 'mae': 0.05613787630050772 6}
```

```
{'rmse': {'name': 'cosine', 'min_k': 9, 'min_support': True, 'user_based': True}, 'mae': {'name': 'cosine', 'min_k': 6, 'min_support': True, 'user_based': True}}
```

Instead of training algorithm by prioritizing RMSE, we instead use MAE as the criteria, and it gives better Results.

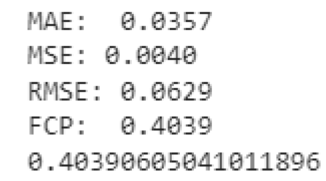
Training the model with highlighted results, we can see that the all the errors decreased and FCP increased. Which is the best we can do for our computational capacity and data.

With MAE-



```
MAE:  0.0348
MSE: 0.0039
RMSE: 0.0622
FCP:  0.4606
0.4605858363846456
```

WITH RMSE –



```
MAE:  0.0357
MSE: 0.0040
RMSE: 0.0629
FCP:  0.4039
0.40390605041011896
```

2.4) KNN Basic

This is the simple KNN implementation that makes recommendations by trying to group users with similar preferences. K here represents the number of similar users to consider when making a recommendation.

Work to tune/train the model –

The Hyperparameters for **KNN** are like before.

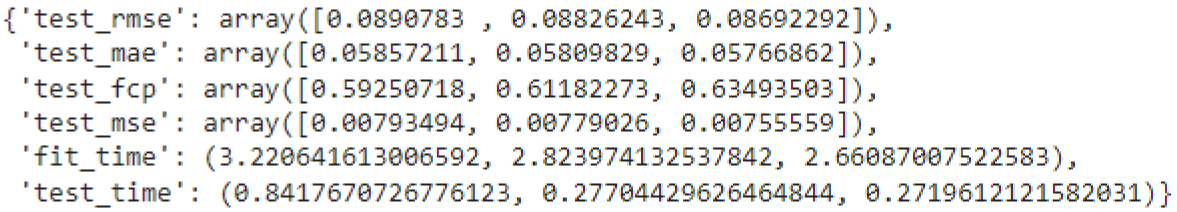
After Running all these parameters with GridSearchCV, we arrived at the below hyper-parameters being the most optimal.

```
{'rmse': 0.08790605647149062, 'mae': 0.05774443005380259}
{'rmse': {'name': 'cosine', 'min_k': 6, 'min_support': True, 'user_based': True}, 'mae': {'name': 'cosine', 'min_k': 3, 'min_support': True, 'user_based': True}}
```

Instead of training algorithm by prioritizing RMSE, we instead use MAE as the criteria, and it gives better Results.

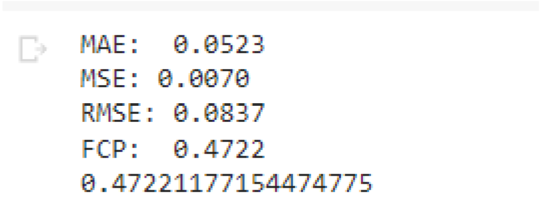
Training the model with highlighted results, we can see that the all the errors decreased and FCP increased. Which is the best we can do for our computational capacity and data.

With MAE-



```
{'test_rmse': array([0.0890783 , 0.08826243, 0.08692292]),
 'test_mae': array([0.05857211, 0.05809829, 0.05766862]),
 'test_fcp': array([0.59250718, 0.61182273, 0.63493503]),
 'test_mse': array([0.00793494, 0.00779026, 0.00755559]),
 'fit_time': (3.220641613006592, 2.823974132537842, 2.66087007522583),
 'test_time': (0.8417670726776123, 0.27704429626464844, 0.2719612121582031)}
```

WITH RMSE –

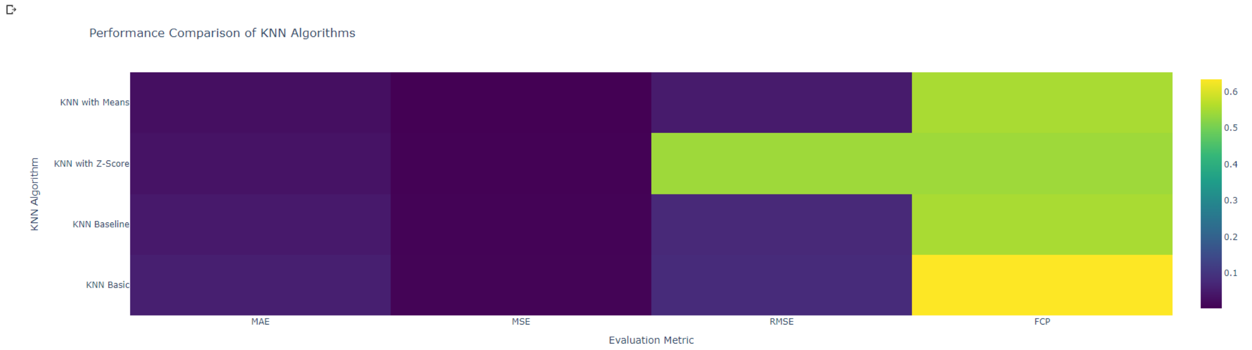


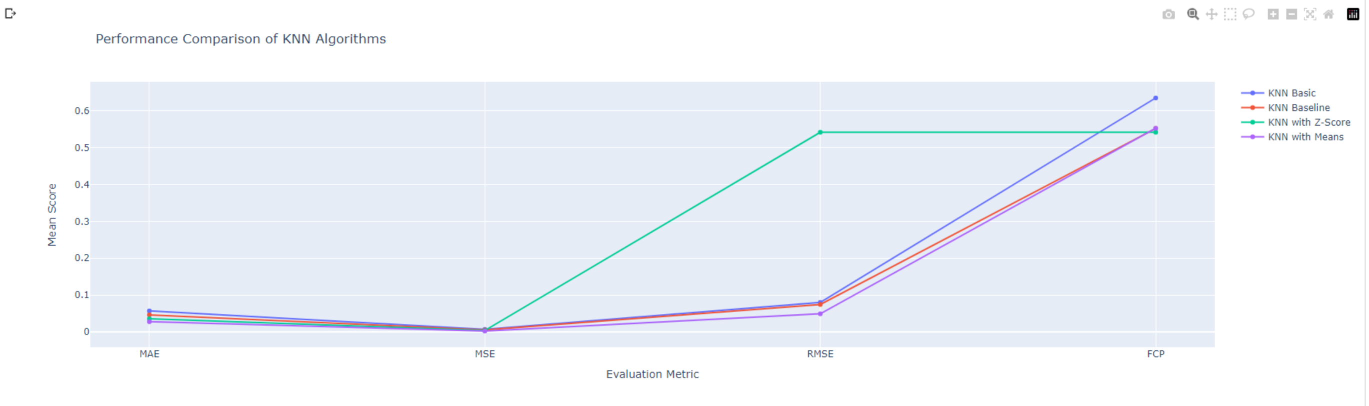
```
MAE:  0.0523
MSE: 0.0070
RMSE: 0.0837
FCP:  0.4722
0.47221177154474775
```

Best KNN Technique (KNN Basic)

We drew multiple visualizations to analyse and pick out the best algorithm for our use case from the KNN class of algorithms. Looking at the visualizations we can see that KNN Basic is the best choice amongst others it promotes higher FCP and lower RMSE, MAE and MSE amongst all the other variations.

With an MAE of just 0.057, if the actual rating is 4, our model will recommend restaurants considering the rating to be 4.057 which isn't bad.





What have we learnt?

We can conclude that KNN Basic dealt with missing values efficiently, took care of overfitting (missing in case of KNN Baseline) and isn't as sensitive to outliers (like KNN with ZScore) which makes it stand out as the best amongst the lot.

2) Matrix Factorization Algorithms

This is the second class of algorithms we explored for our problem statement. This class includes Matrix Factorization Algorithms like – SVD, SVDpp, and NMF.

What are Matrix Factorization Algorithms?

Matrix Factorization Algorithms are used to decompose the user-item matrix to lower dimensions that capture the general trends of the items and user's correlation.

Why we chose this class of algorithms?

We leaned towards using such an algorithm because after forming the user-restaurant interaction matrix, we observed that the data was sparse i.e., not all users had rated all restaurants owing to which most of the fields in the matrix were NaN. Matrix Factorization algorithms are a good choice for such a use-case as they can effectively work with sparse data and capture latent patterns effectively.

What is expected?

Matrix Factorization Algorithms are used to decompose the user-restaurant matrix to lower dimensions that capture the general trends of the restaurants and user's correlation. In our case, we are trying several different configurations to efficiently capture all the variation and latent features of data for better recommendations. This class of algorithms should capture the variation effectively and give us better results than the former class. We expect to use the ratings predicted by the algorithm to suggest restaurants not rated by user (that we predicted high ratings for) to the user.

How was it applied?

To apply this algorithm to our data, we need to find out the user-restaurant interaction matrix. Then the algorithm breaks down the user-item interaction matrix into the product of two rectangular matrices with lesser dimensions via matrix factorization methods. We can then use the lesser dimensions by doing a dot product of the latent vectors to get the predicted ratings. The ratings are then used to pick out most restaurants not rated by them, that we assume would be rated highly and recommend them. Here we tried different hyperparameters to fine tune the model and get the best results.

2.1) SVD

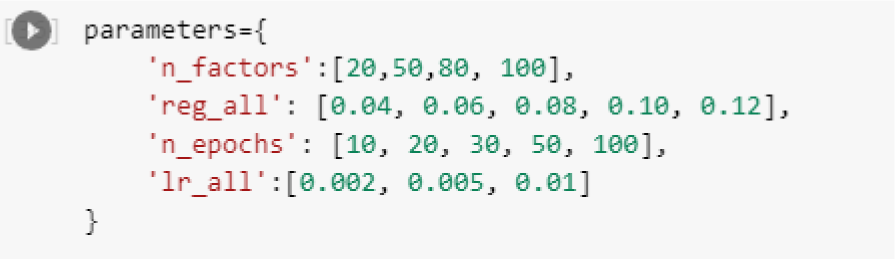
Work to tune/train the model –

We tried with a number of hyper-parameters to see what configuration gave the best performance with vanilla SVD. These parameters included –

1) Number of reduced factors to capture variation (n_factors)

2) Regularization parameter to prevent overfitting (reg_all)

3) Number of Epochs for the model to train (n_epochs)
4) Learning Rate (lr_all)



```
parameters={
        'n_factors':[20,50,80, 100],
        'reg_all': [0.04, 0.06, 0.08, 0.10, 0.12],
        'n_epochs': [10, 20, 30, 50, 100],
        'lr_all':[0.002, 0.005, 0.01]
    }
```

Grid Search CV – was used to evaluate accuracy metrics for SVD on various combinations of parameters, over a cross-validation procedure. After running over above combinations we picked out the hyper parameters that gave the best result. In our case -

```
{'rmse': 0.08918589319325369, 'mae': 0.05693582246718209б}
```

```
{'rmse': {'n_factors': 20, 'reg_all': 0.1, 'n_epochs': 100,
'lr_all': 0.002}, 'mae': {'n_factors': 20, 'reg_all': 0.1,
'n_epochs': 50, 'lr_all': 0.01}}
```

Now, using parameters based on RMSE, we got the following results on the evaluation metrices –

MAE: 0.0439

MSE: 0.0045

RMSE: 0.0670

FCP: 0.5509

0.5509170379259515

2.2) SVD Plus Plus

SVD Plus Plus is an extension to Vanilla SVD that considers the implicit feedback (check-ins to restaurant) in our case.

Why SVD++ over SVD?

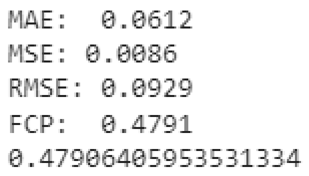
Though more computationally expensive, SVD++ usually accurately captures more variations and is usually better at recommending. We used this because we the dataset contained feedback in the form of Checkins made by users to a restaurant. This should work in our favor and provide better recommendations.

Variations

We tried SVD++ taking checkins and ratings as implicit feedback –

2.2.1) Checkins as Implicit Feedback

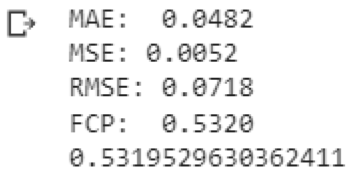
As per the evaluation metrices, using implicit feedback didn't work in our favor and produced high MAE, RMSE and MSE, even the FCP was quite low.



```
MAE:  0.0612
MSE: 0.0086
RMSE: 0.0929
FCP:  0.4791
0.47906405953531334
```

2.2.2) Ratings as Implicit Feedback

Using ratings as implicit Feedback using SVDpp gave us better results than feedback using checkins, although not as much we would like –

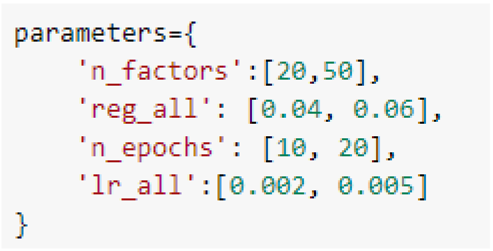


```
MAE:  0.0482
MSE: 0.0052
RMSE: 0.0718
FCP:  0.5320
0.5319529630362411
```

Work to tune/train the model –

We tried with a number of hyper-parameters to see what configuration gave the best performance with SVDpp. These parameters included –

1) Number of reduced factors to capture variation (n_factors)
2) Regularization parameter to prevent overfitting (reg_all)
3) Number of Epochs for the model to train (n_epochs)
4) Learning Rate (lr_all)



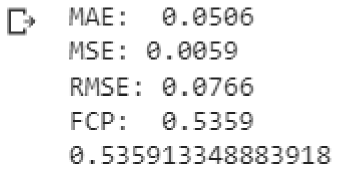
```
parameters={
    'n_factors':[20,50],
    'reg_all': [0.04, 0.06],
    'n_epochs': [10, 20],
    'lr_all':[0.002, 0.005]
}
```

After testing the model on the above-mentioned combinations, we got the below result.

```
{'rmse': 0.09188648675557219, 'mae': 0.059843975031419916}
```

```
{'rmse': {'n_factors': 20, 'reg_all': 0.06, 'n_epochs': 20, 'lr_all': 0.005}, 'mae': {'n_factors': 20, 'reg_all': 0.06, 'n_epochs': 20, 'lr_all': 0.005}}
```

Training the model with highlighted results, we can see that our FCP increased, although costing us with a smaller increase in RMSE, MAE and MSE.



```
MAE:  0.0506
MSE: 0.0059
RMSE: 0.0766
FCP:  0.5359
0.535913348883918
```

2.3) NMF

Non-Negative Matrix Factorization is another one of the matrix factorization techniques that is often used in recommender systems.

Why over SVD and SVD++?

1) It is a nonlinear method as opposed to SVD and SVDpp that can only capture Linear relationships.
2) NMF can handle missing values in the data, which is our case as the crowdsourced Review data has a lot a missing data owing to which it is sparse.

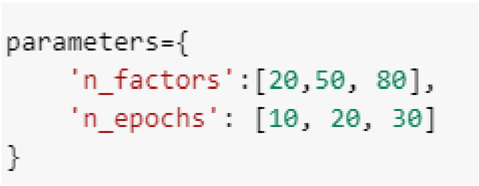
What could go wrong?

3) NMF is sensitive to noisy data and outliers in which case it can really perform poorly when compared to SVD and SVDpp.

Work to tune/train the model –

We tried with several hyper-parameters to see what configuration gave the best performance with NMF. These parameters included –

1) Number of reduced factors to capture variation (n_factors)
2) Number of Epochs for the model to train (n_epochs)



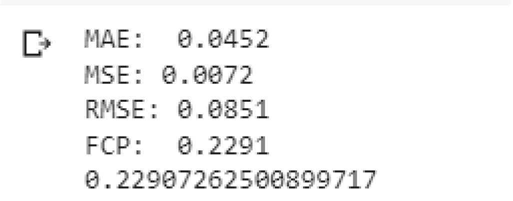
```
parameters={
    'n_factors':[20,50, 80],
    'n_epochs': [10, 20, 30]
}
```

After testing the model on the above-mentioned combinations, we got the below result.

```
{'rmse': 0.09545103974628169, 'mae': 0.05320359396420875}
```

```
{'rmse': {'n_factors': 50, 'n_epochs': 10}, 'mae': {'n_factors': 80, 'n_epochs': 10}}
```

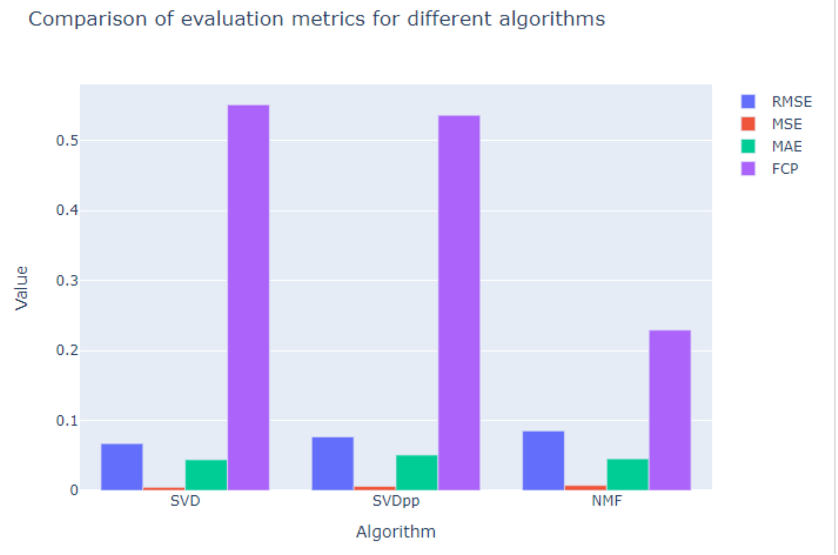
Training the model with highlighted results, our RMSE decreased and FCP increased. The results are not as impressive as we thought they we would be, which can be owed to outliers and noisy data.



```
MAE:  0.0452
MSE: 0.0072
RMSE: 0.0851
FCP:  0.2291
0.22907262500899717
```

Best Matrix Factorization Technique (SVD)

We drew the below visualization to help figure out which **Matrix Factorization** algorithm is the best suited for our dataset and **concluded that SVD** is the best suited for our use case and dataset out of (SVD, SVDpp, NMF) for our dataset.



3) Baseline only

What is Baseline Only?

Baseline Only is a simple yet algorithm for recommendation engines. It calculates a baseline of ratings for given user and item.

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

Why did we chose this class of algorithms?

When compared with Matrix Factorization algorithm family – that requires factorizing the user-restaurant matrix, and the KNN family – that requires calculating similarity between all pairs of user-restaurants, this approach is less computationally intensive, can handle missing data and is more efficient. This is necessary to us since we want to take feedback and incorporate it to our recommendations as efficiently and fast as possible.

What is expected?

It is expected that baseline only may not perform up to standards as compared to other algorithms mentioned before. This is owing to its inability to accurately capture the user-restaurants correlation. Baseline Only model also suffers with the Cold Start problem – Cold start occurs when we must recommend items to users without having any information about any user-item interactions. In our dataset we observe that this is the case since the data is very sparse and we don't have user interactions with a lot of restaurants.

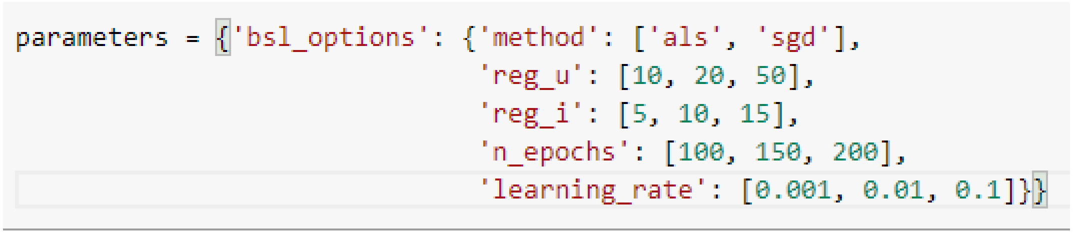
How was it applied?

To apply this algorithm to our data, baseline estimate was calculated for each user and restaurant was calculated using the average rating for each user and restaurant and the user/restaurant specific deviation from that average. These are called baseline estimates. What followed was them being used to predict user ratings for restaurants which were further used to recommend businesses not tried by the user. Luckily for us, this was handled by the library we are using.

Work to tune/train the model –

We tried with several hyper-parameters to see what configuration gave the best performance with NMF. These parameters included –

1) Optimization Algorithm for Baseline estimation (method) – We decide between 2 available optimization algorithms – ALS (Alternate Lease Squares) and SGD (Stochastic Gradient Descent)
2) Regularization Parameter for User features (reg_u) – This is the Regularization parameter for User features in the data to prevent overfitting.
3) Regularization Parameter for Restaurant features (reg_i) – This is the Regularization parameter for Restaurant features in the data to prevent overfitting.
4) Number of Epochs(n_epochs)
5) Learning Rate (learning_rate) – Step size at each iteration of the optimization algorithm.



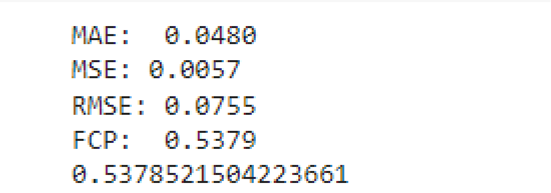
```
parameters = {'bsl_options': {'method': ['als', 'sgd'],
                              'reg_u': [10, 20, 50],
                              'reg_i': [5, 10, 15],
                              'n_epochs': [100, 150, 200],
                              'learning_rate': [0.001, 0.01, 0.1]}}
```

To get the best hyperparameters that can be used for the algorithm to give best recommendations, we use above mentioned configurations with GridSearchCV to filter out the best possible combination to minimize RMSE and MAE. Below are the results -

```
{'rmse': 0.08570833508748459, 'mae': 0.053565584358635135}
```

```
{'rmse': {'bsl_options': {'method': 'als', 'reg_u': 10, 'reg_i': 15, 'n_epochs': 100, 'learning_rate': 0.001}}, 'mae': {'bsl_options': {'method': 'sgd', 'reg_u': 10, 'reg_i': 5, 'n_epochs': 100, 'learning_rate': 0.001}}}
```

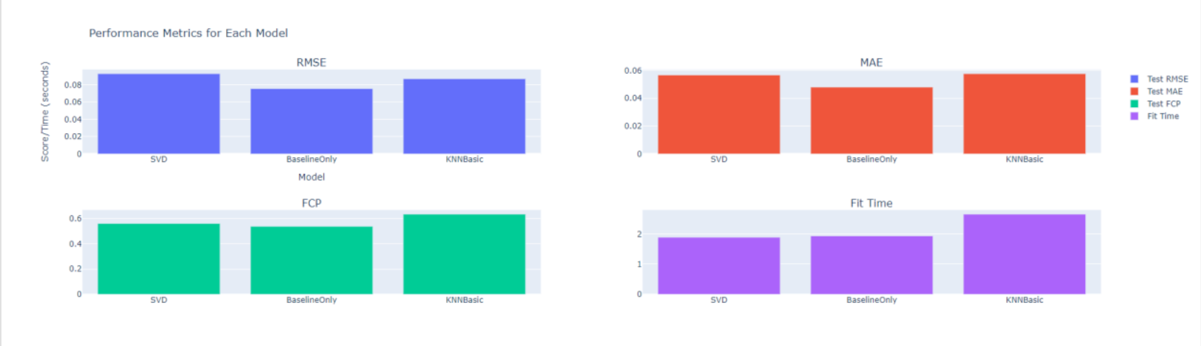
Training the model with highlighted results, our RMSE decreased and FCP increased. The results are not as impressive as we thought they we would be, which can be owed to outliers and noisy data.



```
MAE:  0.0480
MSE: 0.0057
RMSE: 0.0755
FCP:  0.5379
0.5378521504223661
```

Conclusion of the best Collaborative Filtering Algorithm using K-Fold

Before proceeding to visualization, we wanted to confirm our results by doing cross validation. We used KFold with cv = 3 and made the following observations post obtaining the results –



Discoveries-

1) BaselineOnly has the smallest RMSE in predicting ratings as well as smallest average error in predicting ratings.
2) KNNBasic follows closely with SVD performing worst of all.

Expected-

3) Baseline Only and SVD have short fit time suggesting they are computationally efficient and are faster at making recommendations.
4) KNNBasic has the highest FCP, meaning that it is better than other models that recommending restaurants that the user will like.
5) KNNBasic had high test times suggesting it may not be best choice for real time recommendations.

Conclusion – We could prefer **either BaselineOnly or KNN Basic** as BaselineOnly offers highest prediction accuracy whilst being computationally efficient, whereas KNN Basic has the highest FCP meaning that it is good at predicting which restaurants our users will prefer.

Content Filtering

Content based recommender system tries to find similarities between items *(based on item attributes)* using the items that the user has positively reacted to. These items are then recommended to the users based on the magnitude of their similarity.

Content over Collaborative Filtering

Content based techniques mitigate the Cold Start problem to some extent as they try to find the correlation between the restaurants and recommend users based on their previous actions or explicit feedback. Cold Start is a problem we witnessed with our dataset as the data was sparse owing to users not interacting with the restaurant in form of check-ins or ratings. Content Filtering is better than Collaborative Filtering in our case as we don't have a lot of significant user-restaurant interaction data leading to cold-start.

What is Vector Space and TF-IDF?

Vector Space is a mathematical concept to represent textual data in multi-dimensional space. Here each dimension corresponds to a unique word and the dimensional value is decided by the occurrences of that word in the corpus.

TF-IDF is a technique used to measure the importance of each word in the corpus, this is done by measuring the Term Frequency in the document and the Inverse Document Frequency (frequency of the word in the entire corpus). The goal is create a numerical measure representing the importance of the word while taking into account the common words (a, an, the). This is then combined with vector space to create a numerical vector to represent a user or an item *(in case of recommender systems)*

$$\text{similarity} = \cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|} = \frac{\sum\limits_{i=1}^{n} A_i B_i}{\sqrt{\sum\limits_{i=1}^{n} A_i^2} \sqrt{\sum\limits_{i=1}^{n} B_i^2}},$$

Why we chose this class of algorithms?

In our dataset, we deal with 4GB of text reviews which gives us a solid foundation to consider this technique as it efficiently represents and compares the text data. Our earlier algorithms suffered with the problem of cold start owing to sparse and unreliable interactions between user and restaurants. However, in this case we use review text, cuisine, categories ***(all readily available without depending on user interaction)*** to find correlation between restaurants. These restaurants are then recommended to the users.

What is expected?

We expect this algorithm to be more accurate in giving recommendations *(restaurants that are most similar to the user's previously highly rated restaurants in the vicinity)* as this algorithm doesn't struggle with cold-start problem and should boast better precision and recall owing to large corpus of textual reviews, features (cuisines, tags), and geographical coordinates. We expect this algorithm to be more computationally expensive as it trains on large corpus of reviews.

Work to train the model –

We needed to get the data available to us in the correct form before training the model. Sampling of the data was also required owing to the large corpus of textual reviews and businesses across the United States. Below are the steps we performed to get the data ready to be modelled-

1) Combined all the reviews for every business_id by grouping.
2) We grouped the text so that the transforming the text into vectors with grouping can save run time.
3) Merged the grouped text into business table.
4) Added cuisines columns to our review table. So that it can be used for our Content based Recommendation.

To tune, we tried with different similarity metrics – cosine, jaccard, hamming and different combinations of features to improve the recommendations.

Evaluation metrics

There are no common evaluation matrices for measuring the content based recommendation system's with no ground truth (https://stats.stackexchange.com/questions/573798/model-performance-when-ground-truth-is-not-available, https://cs.stackexchange.com/questions/19699/how-to-evaluate-recommendation-engine-without-ground-truth, https://onlinelibrary.wiley.com/doi/full/10.1002/aaai.12055).

However, we wrote some custom evaluation measures based on our problem statement and information about the already existing eval measures. These are mentioned below -

Precision:

Precision gives us the information about how exact and accurate our model is. It is obtained by dividing true positive by false positives. Here our true positives are the items that we recommended and are in the relevant items. False positive in our model is number of recommendations subtracting with our true positives. It is one of the key evaluation metric that determine the performance of our Content based model. the formula as follows:

$$\text{Precision} = \frac{TP}{TP + FP}$$

In terms of recommendation system high precision indicates the recommendations are more close to the User's relevant items.

Recall

Recall is a measure of the percentage of relevant restaurants that the system suggested. In other words, it counts the number of how many of the user's preferred restaurants were actually recommended. The formula is as follows:

$$\text{Recall} = \frac{TP}{TP + FN}$$

where the number of pertinent restaurants that were not recommended is FN (false negative).

In terms of recommendation system high recall indicates the recommendations are more close to the User's preferred restaurants*.*

F1-Score

F1-score is the harmonic mean of precision and recall. It is the combined effect of both precision and recall which gives us the single value that determine the system's performance. The F1-score formula is as follows:

$$F_1\text{–score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} = \frac{2\text{TP}}{2\text{TP} + \text{FP} + \text{FN}}$$

In terms of recommendation system high F1-score indicates the good balance between precision and recall which is our desired model.

How was TF-IDF applied?

We tried 2 different ways of implementing our recommendation system – based on past preferences and vicinity and based on textual input and past preferences.

1) Recommending Restaurants based on Past Preferences and Vicinity

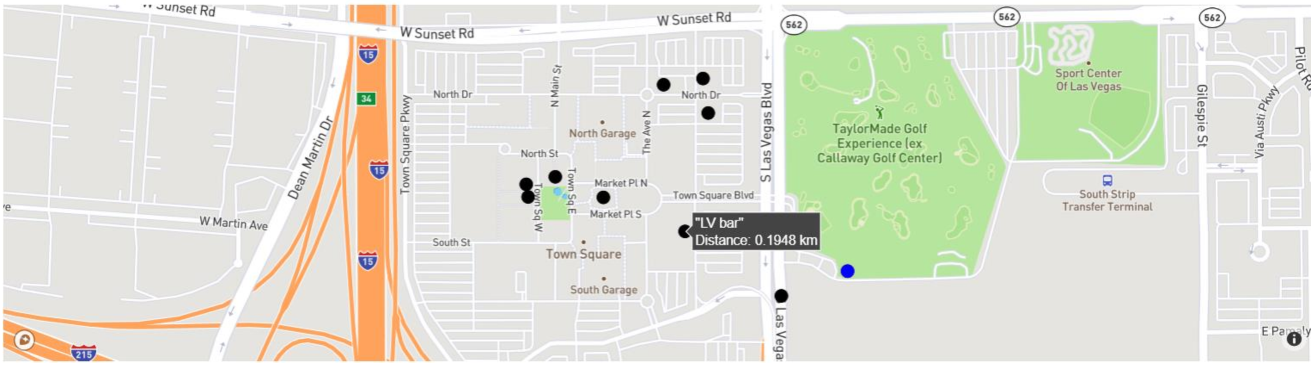
In this recommendation system, we are recommending restaurants to users that are similar to restaurants that are highly rated by the said user and in user's geographic vicinity.

We started by removing the stop words from the restaurant reviews. What followed was creating a TF-IDF matrix taking features (categories, cuisine, review text and stars_scaled(*normalized restaurant stars))* and applying Cosine Similarity function to obtain the cosine similarity matrix.

We then create a function to return top recommendations based on the user_id, and the geographical location of the user (lat, lang), which helps us in returning the restaurants that are most similar to users previously highly rated ones (>=3) in the vicinity.

Recommendations

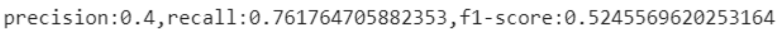
Here we can see that the user (represented by blue trace) is highlighted 10 restaurants that are most similar to his highly rated restaurants in their vicinity.



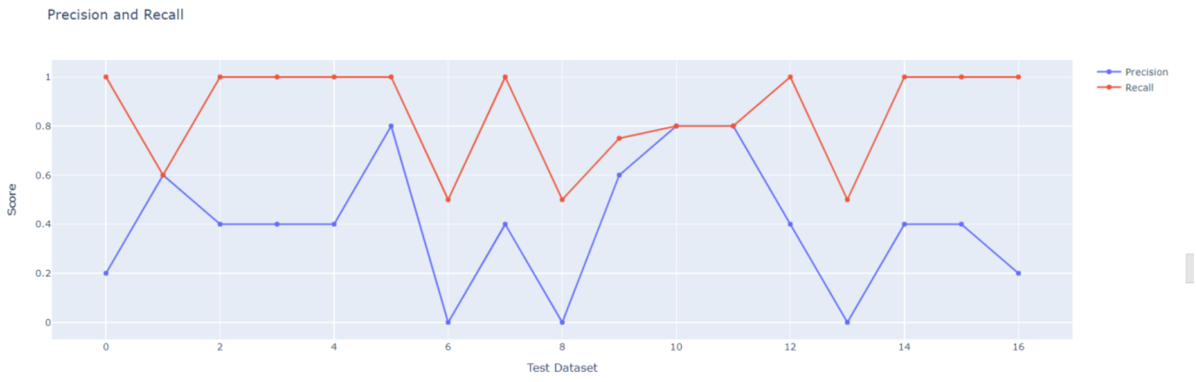
Results

With Cosine Distance –

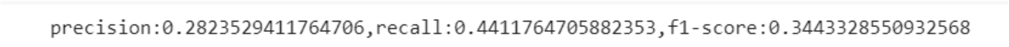
We plotted values of Precision and Recall for every user in the sampled test dataset, and observed that for across every user, our Precision and Recall were inversely proportional (which is expected). However, for most users, our precision was less than the recall suggesting that our algorithm was able to capture relevant restaurants for the user but is also recommending them a lot of irrelevant restaurants.



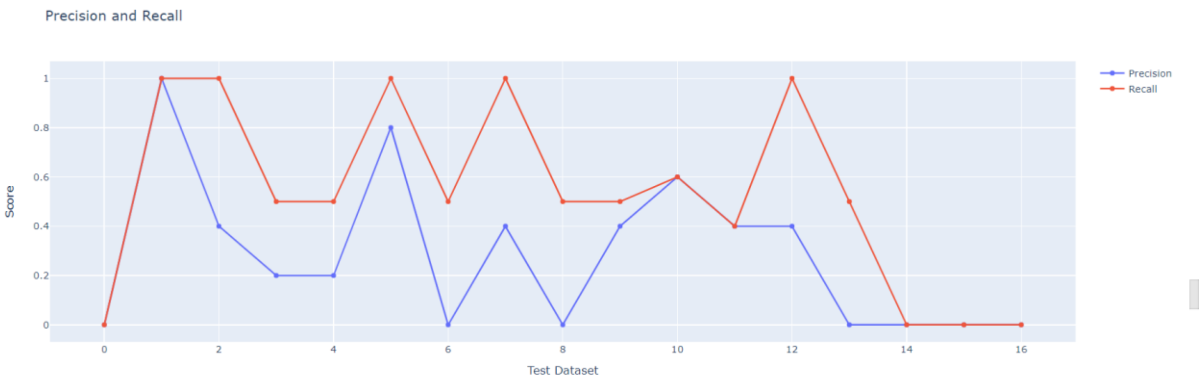
```
precision:0.4,recall:0.761764705882353,f1-score:0.5245569620253164
```



With Eucledian Distance –



```
precision:0.2823529411764706,recall:0.4411764705882353,f1-score:0.3443328550932568
```

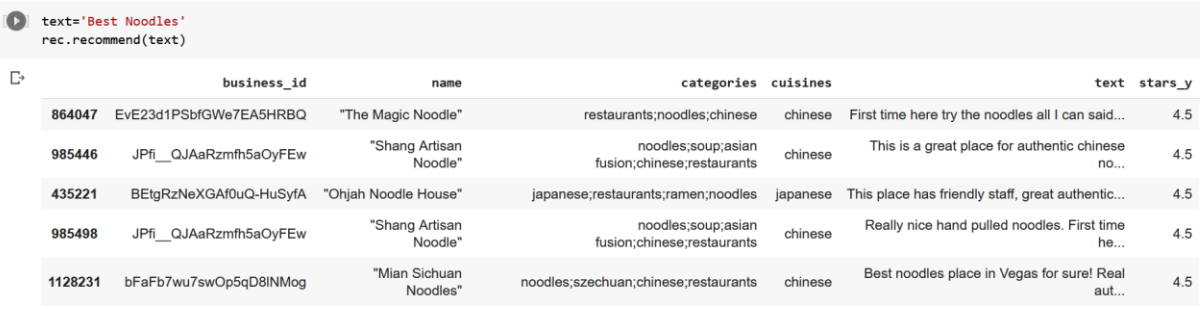


2) Recommending Restaurants based Text Input.

We started by creating a Content Initializer where we selected our features, joined them and transformed into vector using TFIDF vectorizer from sklearns. Next, we created a function to recommend restaurants that takes "search text" as input. The algorithm, then transforms the input text into a TFIDF vector and finds similar businesses using Cosine Similarity. We then finally sort the result based on ratings and return the top 5 restaurants.

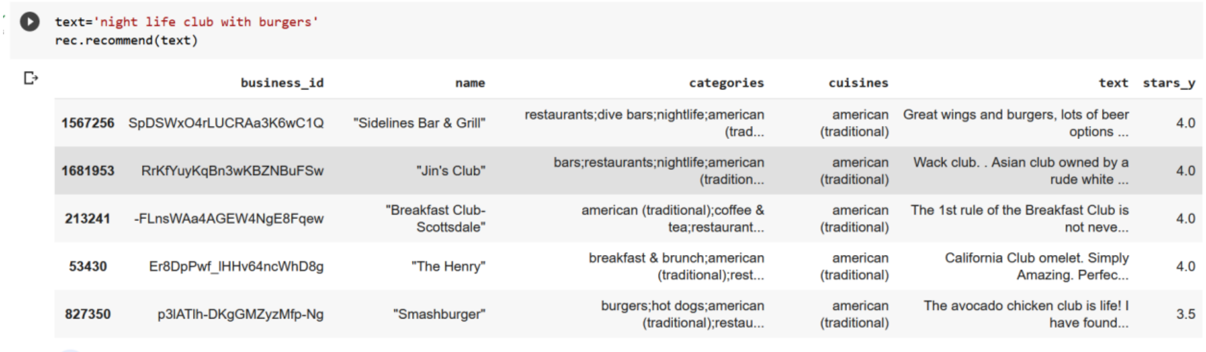
Limitation Encountered: We tried to do the above model by additionally recommending restaurants within user's vicinity. But computing the distances between the user's location and all other businesses location and transforming text to vectors in the same Class is computationally expensive as our dataset is huge. We further tried by sampling data but the results are not up to the mark.

Recommendations



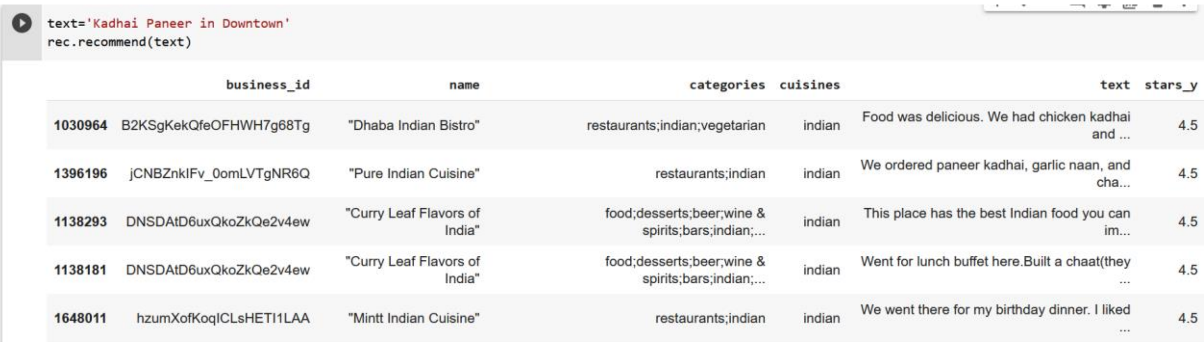
```
text='Best Noodles'
rec.recommend(text)
```

	business_id	name	categories	cuisines	text	stars_y
864047	EvE23d1PSbfGWe7EA5HRBQ	"The Magic Noodle"	restaurants;noodles;chinese	chinese	First time here try the noodles all I can said...	4.5
985446	JPfi__QJAaRzmfh5aOyFEw	"Shang Artisan Noodle"	noodles;soup;asian fusion;chinese;restaurants	chinese	This is a great place for authentic chinese no...	4.5
435221	BEtgRzNeXGAf0uQ-HuSyfA	"Ohjah Noodle House"	japanese;restaurants;ramen;noodles	japanese	This place has friendly staff, great authentic...	4.5
985498	JPfi__QJAaRzmfh5aOyFEw	"Shang Artisan Noodle"	noodles;soup;asian fusion;chinese;restaurants	chinese	Really nice hand pulled noodles. First time he...	4.5
1128231	bFaFb7wu7swOp5qD8INMog	"Mian Sichuan Noodles"	noodles;szechuan;chinese;restaurants	chinese	Best noodles place in Vegas for sure! Real aut...	4.5



```
text='night life club with burgers'
rec.recommend(text)
```

	business_id	name	categories	cuisines	text	stars_y
1567256	SpDSWxO4rLUCRAa3K6wC1Q	"Sidelines Bar & Grill"	restaurants;dive bars;nightlife;american (trad...	american (traditional)	Great wings and burgers, lots of beer options ...	4.0
1681953	RrKfYuyKqBn3wKBZNBuFSw	"Jin's Club"	bars;restaurants;nightlife;american (tradition...	american (traditional)	Wack club. . Asian club owned by a rude white ...	4.0
213241	-FLnsWAa4AGEW4NgE8Fqew	"Breakfast Club-Scottsdale"	american (traditional);coffee & tea;restaurant...	american (traditional)	The 1st rule of the Breakfast Club is not neve...	4.0
53430	Er8DpPwf_IHHv64ncWhD8g	"The Henry"	breakfast & brunch;american (traditional);rest...	american (traditional)	California Club omelet. Simply Amazing. Perfec...	4.0
827350	p3IATlh-DKgGMZyzMfp-Ng	"Smashburger"	burgers;hot dogs;american (traditional);restau...	american (traditional)	The avocado chicken club is life! I have found...	3.5



```
text='Kadhai Paneer in Downtown'
rec.recommend(text)
```

	business_id	name	categories	cuisines	text	stars_y
1030964	B2KSgKekQfeOFHWH7g68Tg	"Dhaba Indian Bistro"	restaurants;indian;vegetarian	indian	Food was delicious. We had chicken kadhai and ...	4.5
1396196	jCNBZnkIFv_0omLVTgNR6Q	"Pure Indian Cuisine"	restaurants;indian	indian	We ordered paneer kadhai, garlic naan, and cha...	4.5
1138293	DNSDAtD6uxQkoZkQe2v4ew	"Curry Leaf Flavors of India"	food;desserts;beer;wine & spirits;bars;indian;...	indian	This place has the best Indian food you can im...	4.5
1138181	DNSDAtD6uxQkoZkQe2v4ew	"Curry Leaf Flavors of India"	food;desserts;beer;wine & spirits;bars;indian;...	indian	Went for lunch buffet here.Built a chaat(they ...	4.5
1648011	hzumXofKoqlCLsHETI1LAA	"Mintt Indian Cuisine"	restaurants;indian	indian	We went there for my birthday dinner. I liked ...	4.5

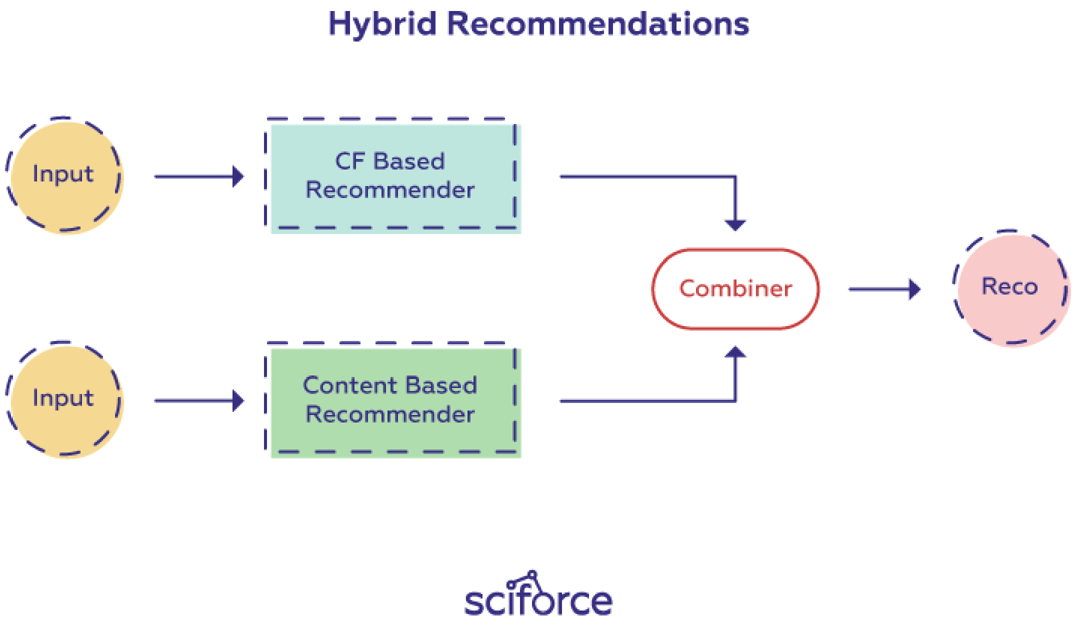
Conclusions

We made the following conclusions –

1) After running our model with Euclidean and Cosine similarity metrics, we observed that our F1 Score was better with cosine similarity which may be due to cosine effectively capturing the similarity in our high dimensional text data, which is not the case with Eucledian as the latter can prioritize features that are not relevant.
2) In both the cases, our precision(0.4) is lower than the recall(0.7567), which is due to our algorithm being able to capture relevant restaurants for the user but also recommending them a lot of irrelevant restaurants.
3) Our F1 Score was 0.5234, suggesting that our algorithm has an average performance in recommending restaurants, and more fine-tuning can improve the quality of predictions. This can also be owed to the fraction (0.0001) data we trained these models on could also imply that they are underfitting.
4) We can observe that the model that recommends based on text performs better than all followed by the algorithm that recommends based on the cosine similarity in the vicinity of the user.

Hybrid Filtering

Hybrid Filtering is a combination of both the previous approaches in the recommendation system. Hybrid filtering combines the strengths of both techniques and aims to mitigate their weaknesses. It can also be more resistant to data sparsity and the cold-start issues faced by independent algorithms discussed earlier. The two methodologies can be combined in a number of ways, such as weighted hybrid, cascading hybrid, and mixed hybrid.



Hybrid over Content and Collaborative Filtering

While both content-based filtering and collaborative filtering have their strengths and weaknesses, Hybrid filtering can offer benefits over both individual approaches. Here are few reasons why we preferred hybrid filtering:

1) Overcoming the cold start problem: Content and Collaborative both have cold start problem in their own ways like when the new items are introduced the Content based struggles to recommend that item based on the past preference. We observed these problems in our earlier implementations as Collaborative suffered with less user-restaurants interactions whereas Content suffered with lack of diversity. For collaborative when there is less data the recommendations are not accurate. By initially recommending new things using content-based filtering and then switching to collaborative filtering once there is sufficient user data, hybrid filtering can get over this problem.
2) Improved diversity of suggestions: Content based frequently suggests the products that are similar to user has already liked, so it can suffer from lack of diversity in recommendations. And Collaborative only makes recommendations based on users past activity. Hybrid filtering can address this by combining both approaches, and thus providing a more diverse set of recommendations.

Overall, hybrid filtering may be more efficient with our problem statement than content-based or collaborative filtering alone and can have advantages over both approaches.

The Model and How it combines Content & Collaborative?

The hybrid recommendation system uses latent features(embeddings) to capture user preferences over items. It combines the features of both content-based and collaborative filtering approaches. By estimating an embedding for each feature of the things and users, the model learns embeddings for both users and items. The final representation is then calculated by adding all of the estimated embeddings.

How it combines Content & Collaborative:

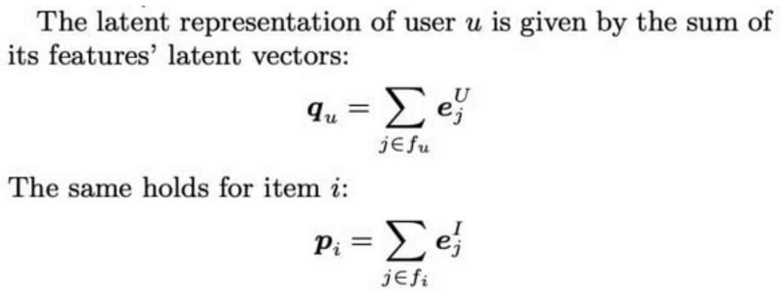
Content - To create the appropriate embeddings for users and items, the model captures the content (features) of each.

Collaborative - Based on the similarity between users and things in the latent space, the model predicts user-item interactions using the learnt embeddings.

Hence, by simultaneously utilizing the content information and user-item interaction data, the model combines the capabilities of both content-based and collaborative filtering approaches to deliver better recommendations.

The Model:

Matrix Factorization: In our model we decompose the interaction matrix into two matrices (user and item) these are also called latent features. The latent embeddings could record latent information about user and object characteristics that reflect their preferences. And by multiplication of these we obtain our original interaction matrix.



The latent representation of user u is given by the sum of its features' latent vectors:

$$\boldsymbol{q}_u = \sum_{j \in f_u} \boldsymbol{e}_j^U$$

The same holds for item i:

$$\boldsymbol{p}_i = \sum_{j \in f_i} \boldsymbol{e}_j^I$$

Figure 6: Formula for user and item latent vectors

Why we chose this class of algorithms?

In Collaborative Filtering the data is sparse and we were unable to use the features of items to recommend users based on their past preferences. It can only recommend restaurants similar users visited.

In content-based Filtering we can only use the item features to recommend restaurants to the user. There is no diversity in the recommendations. It cannot suggest user new items. And the evaluation of the content-based model is not straight forward without user feedback.

In hybrid filtering we get the best of both worlds. If a user is new and has only one review our model uses content based to suggest similar restaurants to the user. Once we have user interactions, the model can use a combination of similar users and similar items by using item interaction matrix.

Evaluation metrics

Since Hybrid Recommendation systems are made from a combination of both Content and Collaborative Filtering algorithms, we can evaluate them using evaluation metrics used in both the cases. We will specifically be using – ROC Score, Precision@k, RMSE. We will give priority of ROC Score because we are interested in knowing about the preference of the user(roc) rather than predicting the correct rating that the user might give to the restaurant(precision).

ROC Score:

ROC stands for Receiver Operating Characteristic, which is a measure of True Positive Rate against False Positive Rate meaning that it is the likelihood that a random relevant item being ranked higher than an irrelevant item. This is calculated by the ranks given by algorithm to all the items. AUC is the proportion of pairs where relevant items were ranked higher than irrelevant items.

$$\mathrm{AUC}(R)_n = \frac{1}{|R|(n - |R|)} \sum_{r \in R} \sum_{r' \in (\{1,\ldots,n\} \setminus R)} \delta(r < r')$$

In terms of recommendation system higher ROC Score means that the algorithm Is able to recommend items that users would rate higher and ignores ones that they wouldn't.

Average Precision@K

Precision @ k is a metric used in recommendation systems to measure the accuracy of top-k recommendations by the system. It is the proportion of top-k recommendations that are relevant to the ones that were recommended.

Average precision@k calculates the precision at every position from 1 to k where we have a relevant item to take care of cases where the position is important. For example if Model A was able to suggest relevant recommendations at position 1 and 2 and model B suggested them at positions 3, 4, Precision@k for A would be higher than B.

$$\mathrm{AP}(R)_k = \frac{1}{\min(|R|, k)} \sum_{i=1}^{k} \delta(i \in R) \mathrm{Prec}(R)_i.$$

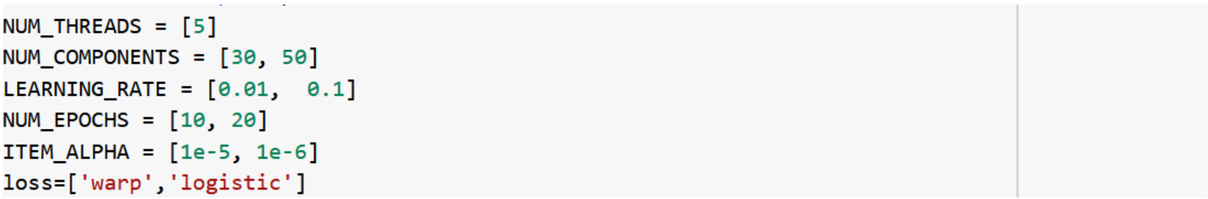
We don't take into account Precision@k for our use case, as we want to capture the restaurants that were preferred by the user and are putting their ranking on the back burner.

What is expected?

We expect this algorithm to be more accurate in giving recommendations *(restaurants that have higher probability that a user might visit them based on empirical user-user and item-item interaction)* as this algorithm combines the strength of both techniques it is expected to improve performance compared to use a single approach. It is expected to achieve better accuracy, robustness and scalability.

Work to tune/train the model –

We tried with several hyper-parameters to see what configuration gave the best performance with Hybrid Recommender Model. These parameters included –



```
NUM_THREADS = [5]
NUM_COMPONENTS = [30, 50]
LEARNING_RATE = [0.01,  0.1]
NUM_EPOCHS = [10, 20]
ITEM_ALPHA = [1e-5, 1e-6]
loss=['warp','logistic']
```

1) **Num Threads** – number of threads to be used on the CPU while training. More threads can speed up the process but increase the memory. For our case, our Colab notebook ran out of memory fast, which is why we used less number of threads.
2) **Num Components** – This is dimensionality of the User and Restaurant embedding space.
3) **Learning Rate** – Learning Rate of the algorithm while optimization is going on.
4) **Number of Epochs** – Number of epochs that the algorithm is run for. A higher number can lead to better model performance.
5) **Item Alpha** – Magnitude of L2 regularization applied on the item embeddings to prevent overfitting.
6) **Loss** – loss function used while training the model. Weighted approximate rank pairwise (WARP) optimizes precision of top-k recommendations whilst Logistic focuses more on optimizing the AUC.

To get the best hyperparameters that can be used for the algorithm to give best recommendations, we use above mentioned configurations with GridSearchCV to filter out the best possible combination to maximize roc score and precision. Below are the results -

```
item_alpha=1e-06, test_precision=0.0003, test_auc=0.5756,best_func=logistic
num_threads=5, num_components=50, learning_rate=0.1, num_epochs=10,
item_alpha=1e-05, test_precision=0.0091, test_auc=0.7940,best_func=warp
num_threads=5, num_components=50, learning_rate=0.1, num_epochs=10,
item_alpha=1e-05, test_precision=0.0003, test_auc=0.5749,best_func=logistic
num_threads=5, num_components=50, learning_rate=0.1, num_epochs=10,
item_alpha=1e-06, test_precision=0.0079, test_auc=0.7919,best_func=warp
num_threads=5, num_components=50, learning_rate=0.1, num_epochs=10,
item_alpha=1e-06, test_precision=0.0003, test_auc=0.5750,best_func=logistic
num_threads=5, num_components=50, learning_rate=0.1, num_epochs=20,
item_alpha=1e-05, test_precision=0.0160, test_auc=0.8094,best_func=warp
num_threads=5, num_components=50, learning_rate=0.1, num_epochs=20,
item_alpha=1e-05, test_precision=0.0003, test_auc=0.5748,best_func=logistic
```

```
num_threads=5, num_components=50, learning_rate=0.1, num_epochs=20,
item_alpha=1e-06, test_precision=0.0143, test_auc=0.8090,best_func=warp
num_threads=5, num_components=50, learning_rate=0.1, num_epochs=20,
item_alpha=1e-06, test_precision=0.0003, test_auc=0.5750,best_func=logistic

Best hyperparameters: num_threads=5, num_components=50, learning_rate=0.1,
num_epochs=20, item_alpha=1e-05, test_precision=0.0160,
test_auc=0.8094,best_loss_func=warp
```

How was Hybrid Filtering Model applied to the dataset?

In this recommendation system, we started by generating the user-restaurant interaction matrix based on the reviews a user gave to the restaurants. However, for hybrid approach, we also needed the item features and the user features.

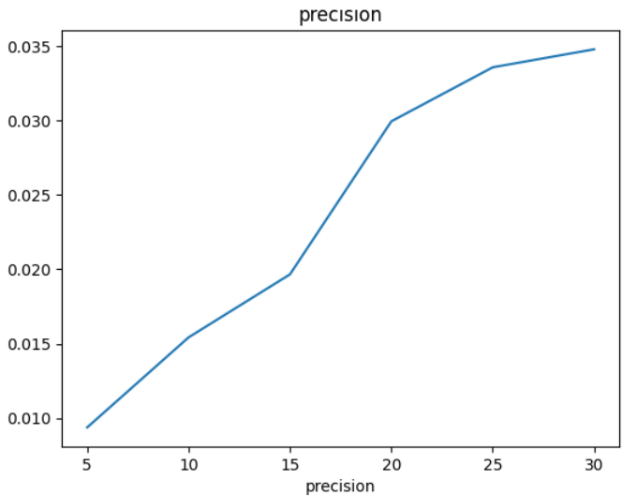
Item Features – taken all the features which are formed by splitting, review count of each restaurant and their stars. Then a function maps feature keys and its values and the result then is passed into a method that creates item features for an item id.

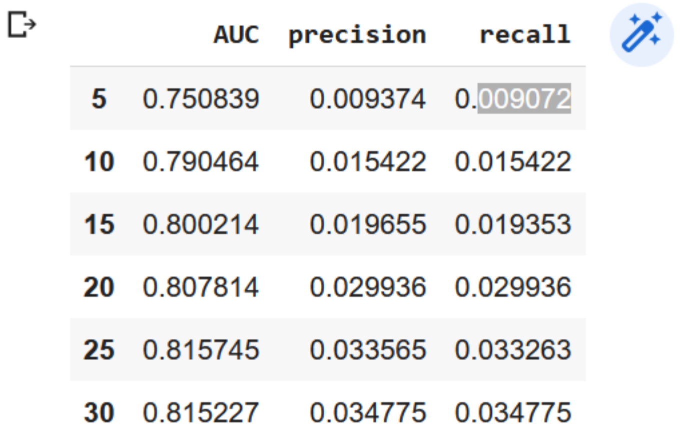
User Features – taken all the features which are formed by splitting, review count of each user and useful (Boolean) count from user table. Then a function maps feature keys and its values and the result is then passed to a method that creates user features w.r.t user_id.

After generating the item and user features, we fit the model using the best hyper-parameters we obtained from GridSearch (to maximize AUC ROC). And generated recommendations. Thus we utilized item and user metadata, which should result in this model perform better at recommending restaurants and mitigate some level of cold-start problems.

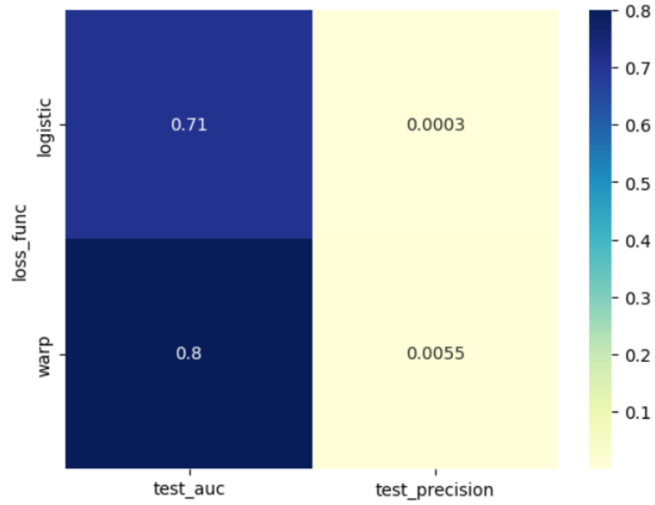
Observations

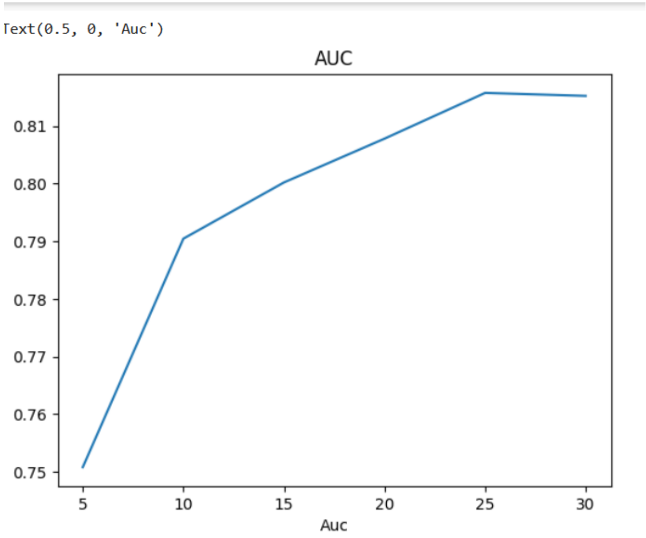
We also tried a greater number of epochs to see how the AUC, precision and recall change with the epochs. We observed that all the three metrics improved with a greater number of epochs suggesting that we need more computation and more epochs to achieve and train the recommendation system to its full potential.





	AUC	precision	recall
5	0.750839	0.009374	0.009072
10	0.790464	0.015422	0.015422
15	0.800214	0.019655	0.019353
20	0.807814	0.029936	0.029936
25	0.815745	0.033565	0.033263
30	0.815227	0.034775	0.034775



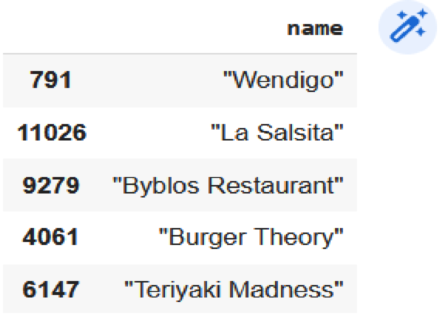


```
Text(0.5, 0, 'Auc')
```

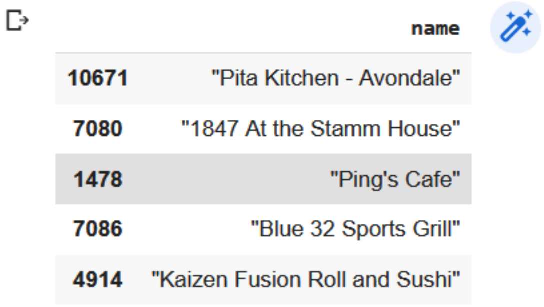
The above figure gives us the information about the test_auc and precision for all the users in test set after training with different losses (logistic, WARP). Based on above heat map, we can infer that the WARP loss function gives us the better precision and AUC and should be chosen to train our model that will make recommendations.

Recommendations

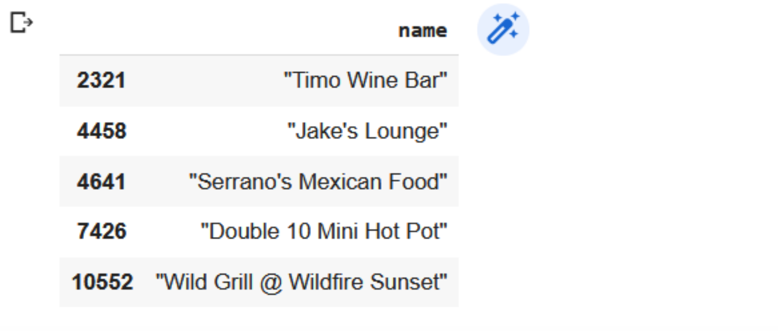
Here we can see recommendations based on user id using our hybrid recommendation model.



	name
791	"Wendigo"
11026	"La Salsita"
9279	"Byblos Restaurant"
4061	"Burger Theory"
6147	"Teriyaki Madness"



	name
10671	"Pita Kitchen - Avondale"
7080	"1847 At the Stamm House"
1478	"Ping's Cafe"
7086	"Blue 32 Sports Grill"
4914	"Kaizen Fusion Roll and Sushi"



	name
2321	"Timo Wine Bar"
4458	"Jake's Lounge"
4641	"Serrano's Mexican Food"
7426	"Double 10 Mini Hot Pot"
10552	"Wild Grill @ Wildfire Sunset"

Overall Conclusions

We made the following conclusions –

1) Collaborative Filtering using KNN Basic performed better than Baseline Approach as the RMSE and MAE were low for KNN Basic indicating that it provided more accurate recommendations than Baseline Only.
2) Content based Filtering has higher recall then precision, suggesting that it is better at identifying restaurants, but some of the identified restaurants are not user's preferences.
3) Hybrid model performed better than Content and at par with Collaborative as it has a test_auc of 0.80 suggesting that model was able to differentiate well between relevant and irrelevant items. However, the low precision can be owed to users not having a lot of interactions with the restaurants. These are cold start users, so we are not worried about them.
4) Overall, we have a choice between choosing KNN Basic and the Hybrid approach. On one hand, we have lower RMSE and MAE, on other an extremely high AUC score and hybrid nature of the algorithm.
5) We intend to proceed with **Hybrid Recommendation System** as the lower Precision can be covered up after the cold start is dealt with after sufficient user item interaction, and the algorithm boasts a very good AUC ROC score and can recommend restaurants based on user preferences and item similarities.

Why different Evaluation measures across Models?

1) **Collaborative Filtering –** Collaborative filtering algorithm works by predicting ratings that a user may give to a restaurant based on their similarity with other users. RMSE, MAE and FCP are used to measure the accuracy of predicted ratings and the error between the predicted and the actual ratings. Precision, Recall, F1 score cannot be used in such a case as they are not applicable to rating predictions.

2) **Content Filtering –** Content based algorithms recommends items based on the similarities between different items based on their features. Precision recall and F1 score are common evaluation measures that measure the performance of a classifier. As in content-based filtering our aim is to decide if an item should be recommended or not to the user based on its similarity score – RMSE, MAE and FCP are not the correct measures here.

3) **Hybrid Filtering –** In Hybrid we are calculating Precision@K because in our model there is no limit to number of recommendations so by using precision at k we can obtain k number of top recommendations. In contrast, content-based model takes input of number of items to recommend. So, we are using precision there.

   And we are not using RMSE and MAE as evaluation metric because we are not predicting ratings in the hybrid recommendation system so those evaluation measures will not be appropriate here.

References –

1) Yelp Restaurant Recommendation System — Data Science Capstone Project | by Dominic Ong | Towards Data Science
2) https://towardsdatascience.com/how-i-would-explain-building-lightfm-hybrid-recommenders-to-a-5-year-old-b6ee18571309
3) https://developers.google.com/machine-learning/recommendation/content-based/basics
4) https://towardsdatascience.com/introduction-to-recommender-systems-1-971bd274f421
5) https://blogs.sap.com/2021/03/29/evaluating-recommender-systems-in-absence-of-labeled-data/
6) https://stats.stackexchange.com/questions/573798/model-performance-when-ground-truth-is-not-available
7) https://surprise.readthedocs.io/
8) https://towardsdatascience.com/recommendation-system-in-python-lightfm-61c85010ce17
9) https://towardsdatascience.com/content-based-recommender-systems-28a1dbd858f5
10) https://neptune.ai/blog/recommender-systems-metrics
11) https://medium.com/sciforce/inside-recommendations-how-a-recommender-system-recommends-9afc0458bd8f
12) https://sites.northwestern.edu/msia/2019/04/24/personalized-restaurant-recommender-system-using-hybrid-approach/
13) https://flowthytensor.medium.com/some-metrics-to-evaluate-recommendation-systems-9e0cf0c8b6cf
14) https://towardsdatascience.com/recommendation-system-in-python-lightfm-61c85010ce17
15) https://medium.com/@bindhubalu/content-based-recommender-system-4db1b3de03e7