A Restaurant Recommendation System

By Data Reply 2

Arish, Adam, An, Josh and Jameelah

Introduction

During the last few decades, with the rise of Youtube, Amazon, Netflix, and many other such web services, recommender systems have taken more and more place in our lives. From e-commerce (suggest to buyers articles that could interest them) to online advertisement (suggest to users the right contents, matching their preferences), recommender systems are today unavoidable in our daily online journeys.

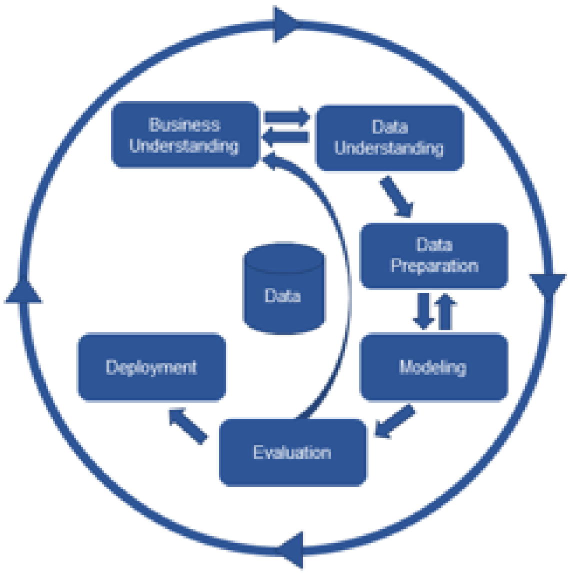
In a very general way, recommender systems are algorithms aimed at suggesting relevant items to users (items being movies to watch, text to read, products to buy or anything else depending on industries).

Recommender systems are really critical in some industries as they can generate a huge amount of income when they are efficient or also be a way to stand out significantly from competitors. As proof of the importance of recommender systems, we can mention that, a few years ago, Netflix organised a challenge (the “Netflix prize”) where the goal was to produce a recommender system that performs better than its own algorithm with a prize of 1 million dollars to win.

In this report, we will go through different algorithms to build a restaurant recommender system. We would like to recommend users the restaurant based on its ratings, at the same time providing insights about different restaurants and areas, and therefore understand how such businesses can become more profitable using our recommender system. For each of them, we will present how they work, describe their theoretical basis, and discuss their strengths and weaknesses.

Analytical Plan

This report aims to explore and analyse a dataset that would produce the best results for a collaborative filtering recommender system. Analysis of the dataset will be performed using the CRISP-DM technique.



The report starts with an initial description of recommender systems followed by the chosen dataset used to build our collaborative filtering system. The following analysis will be discussed within the report:

- An initial statistical analysis of quantitative variables such as mean, median and implement charts. A second analysis of each variable will be conducted to correlate them, and a correlation matrix parallelogram will be constructed.
- Data preparation of the chosen dataset will involve selecting the data which we will use to form our recommender system, such as user id, place id, ratings, etc. In order to address the quality of the data, it will be cleansed to identify anomalies and make our recommender provide more accurate results to the users.
- As part of our modelling analysis, a series of KNN algorithms will be analysed. The RMSE, RMPSE, and MAE scores will be used when testing and training the dataset to accurately pick out the best model.
- In order to model this, the dataset will be split into 4 different datasets including training features, training labels, testing features, and testing labels. These will then be used to get the RMSE, RMSPE, and MAE scores which will then be used to accurately test the data performance.
- The best performing model will be hyper-tuned using different parameters to further optimise the algorithm.
- Evaluating the model to identify the most optimal solution will be the last stage in the analysis, followed by a final thought on the scalability of the dataset used in our recommender system.

Exploratory Data Analysis

Dataset

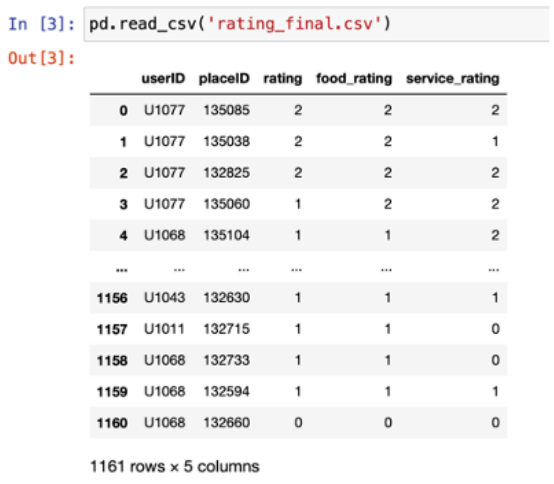
In this project, we will be using a Restaurant Reviews dataset, one based on users giving a rating of a given restaurant

Dataset: Restaurant Ratings (rating_final)
- User ID
- Restaurant ID
- Ratings: Overall rating

The ‘Restaurant Ratings’ dataset has 1161 reviews by 138 users for 130 restaurants. By having a large dataset for our system, this enables us to have a diverse range of restaurants to be recommended.

Below is a small section of the beginning and end of our Restaurant Ratings dataset.

Restaurant Reviews Dataset



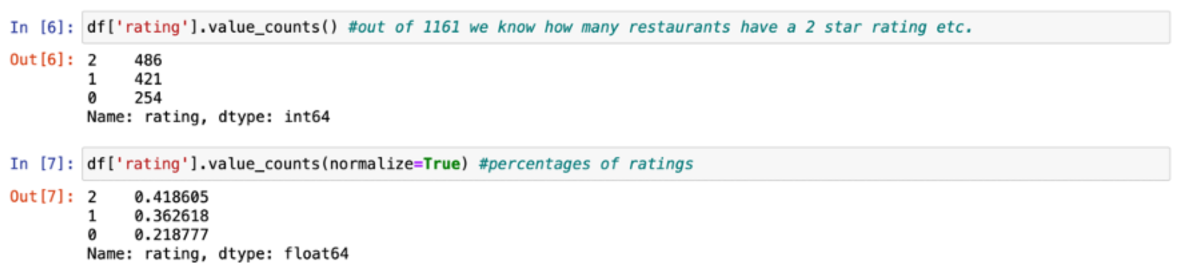
```
In [3]: pd.read_csv('rating_final.csv')
Out[3]:
```

	userID	placeID	rating	food_rating	service_rating
0	U1077	135085	2	2	2
1	U1077	135038	2	2	1
2	U1077	132825	2	2	2
3	U1077	135060	1	2	2
4	U1068	135104	1	1	2
...	...	...	...	...	...
1156	U1043	132630	1	1	1
1157	U1011	132715	1	1	0
1158	U1068	132733	1	1	0
1159	U1068	132594	1	1	1
1160	U1068	132660	0	0	0

1161 rows × 5 columns

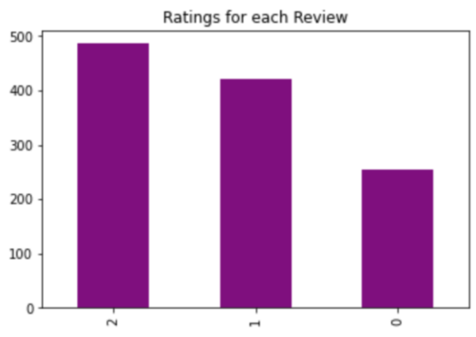
As you can see from above, each unique review is represented in a row. By looking at the size of our table we are able to learn from the data that there are 1161 reviews in this dataset. Restaurants are uniquely identified with their placeID and each review is given by a person with their userID. The ratings within each review is given by a numerical value of either 0, 1 or 2.

By analysing the dataset, we are able to find how many reviews have a rating of 2, 1 or 0, and then find out the percentages of the restaurants.



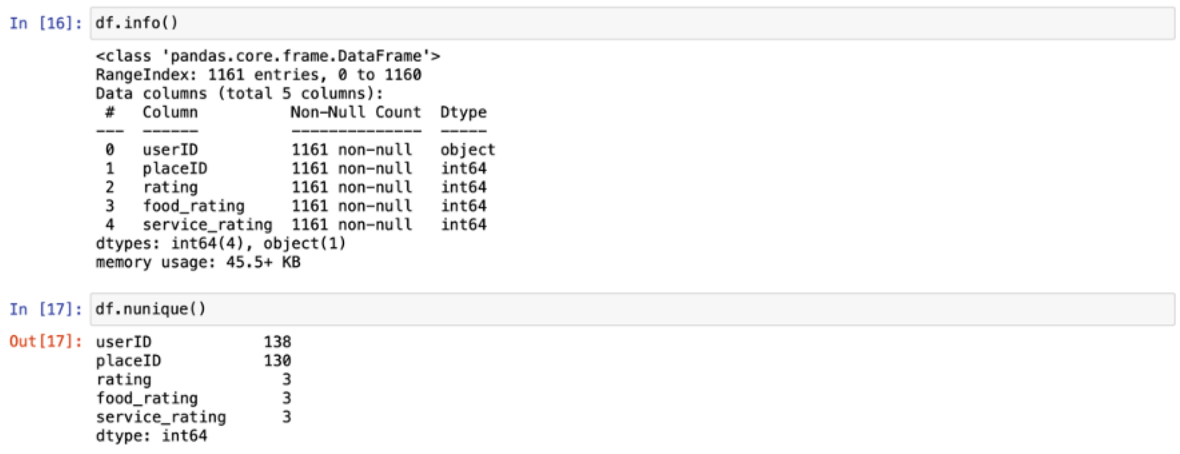
```
In [6]: df['rating'].value_counts() #out of 1161 we know how many restaurants have a 2 star rating etc.

Out[6]: 2    486
        1    421
        0    254
        Name: rating, dtype: int64

In [7]: df['rating'].value_counts(normalize=True) #percentages of ratings

Out[7]: 2    0.418605
        1    0.362618
        0    0.218777
        Name: rating, dtype: float64
```

Here is a visual representation of the ratings for each review.



Most reviews have a rating of 2, and the least reviews have a rating of 0.

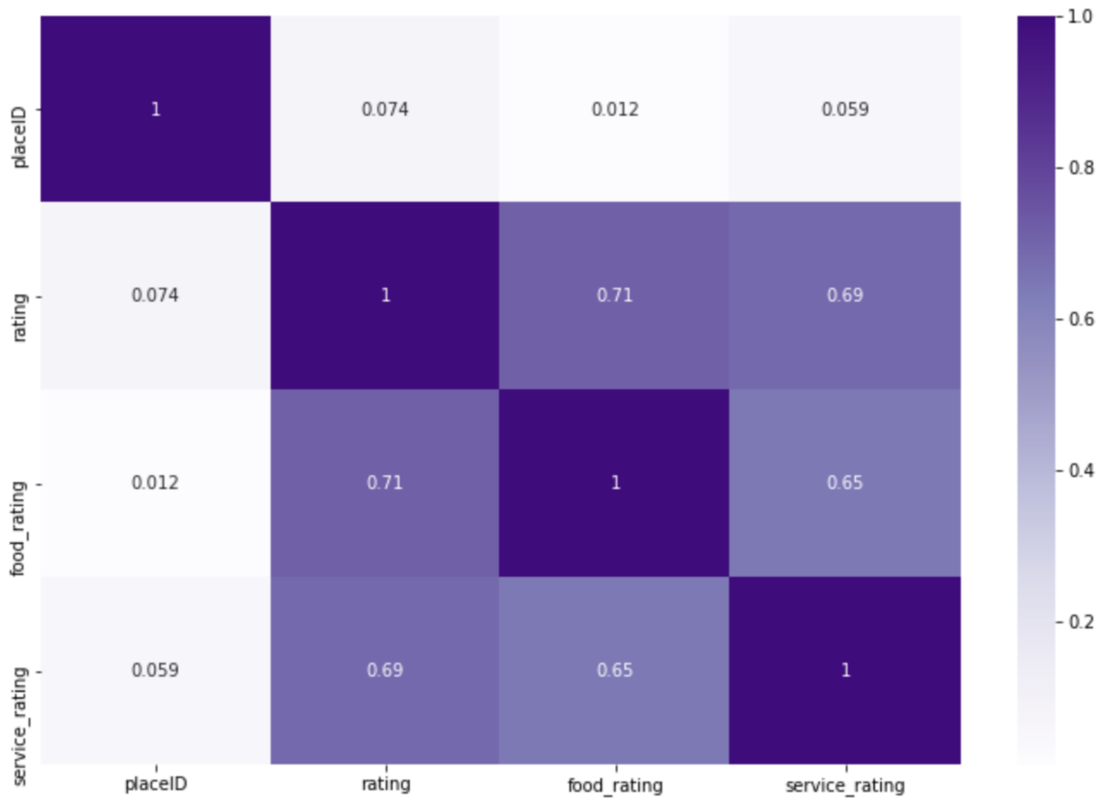
Uniqueness of Variables in Restaurant Ratings Dataset



```
In [16]: df.info()

         <class 'pandas.core.frame.DataFrame'>
         RangeIndex: 1161 entries, 0 to 1160
         Data columns (total 5 columns):
          #   Column          Non-Null Count  Dtype
         ---  ------          --------------  -----
          0   userID          1161 non-null   object
          1   placeID         1161 non-null   int64
          2   rating          1161 non-null   int64
          3   food_rating     1161 non-null   int64
          4   service_rating  1161 non-null   int64
         dtypes: int64(4), object(1)
         memory usage: 45.5+ KB

In [17]: df.nunique()

Out[17]: userID            138
         placeID           130
         rating              3
         food_rating         3
         service_rating      3
         dtype: int64
```

The uniqueness of the variables is demonstrated here, which includes how many unique user IDs there are in the dataset. In comparison to the 1161 reviews we have in the dataset, there are 138 unique user IDs meaning 138 unique individuals have contributed to completing the reviews. There are also 130 unique restaurants. Again, we can see there are only 3 unique ratings of either a 0, 1 or 2 for each restaurant review. The only variables that we felt were necessary for our project were userID, placeID and rating, so this is what we used.

The correlation matrix between the variables is shown below.



Here the correlation matrix describes how strongly each pair of variables are related to each other. Restaurants with a high rating, also have a high food rating and service rating.

In this section we will examine the methods we used to create this model, the results we got and what these results mean.

Models

In order to find the optimal and most accurate solution we looked at different solutions in order to compare them. As this is collaborative filtering we decided to analyse a series of KNN (K- nearest neighbours) algorithms. The model which had the highest accuracy scores were chosen for hyperparameter tuning.

The variables used in the modelling are: userID, placeID, rating.

The K-nearest neighbour method involves using memory to store the training data and use this to classify each time, as opposed to supervised learning. (Zoltan, 2018)

This memory based approach was chosen due to the fact that the dataset we are working with is relatively small and does not have sparse data so more complex model based solutions are not necessary. Also we are not looking to incorporate latent factors(beyond the scope of this project), therefore the supervised learning based approach again is not needed. We did however look to combine these methods slightly as we used different variations of the KNN model which attempted to account for bias in ratings. The methods still involve using training data to calculate similarities and storing them in memory however these calculations also incorporated different ways to try and remove biases.

The methods we looked at are below:

KNN Basic

- The simplest collaborative filtering approach using the nearest neighbour approach to find similar users, no extra calculations to attempt to remove bias in ratings.

KNN With Means

- Calculated the same as the above basic algorithm except also factors in the average rating for that user and uses it as a weighting to normalise the result.

KNN With Baseline

- This algorithm uses a baseline to normalise the results. The baseline rating is calculated for each user-item interaction and then used as part of the basic algorithm to improve accuracy and remove bias.

KNN With ZScore

- This algorithm uses a specific type of normalisation, called z-score normalisation. It takes into account the average and standard deviation which helps avoid issues relating to outliers skewing the result.

Parameter options for the chosen model:
Once the most accurate model is determined, we must then compare the different parameters for the algorithm, in order to further optimise the algorithm for our dataset. Note that before specifying parameters, our initial parameters by default are set to user-based and the msd rule (mean square difference) for calculating similarity.

The parameters we will test below are:

Parameter	Overview
Similarity option	Either the cosine rule of similarity, mean square difference or Pearson baseline.
Minimum support	The minimum number of common items or users needed to consider them for similarity. It is common items if it is item based and common users if user based (see next parameter)
Item/user based	If item based, you are finding similarity between items, focusing on how items are rated by users. With user based, you are finding similarity between users, focusing on how a user rates items. (Abhinav Ajitsaria, n.d.)

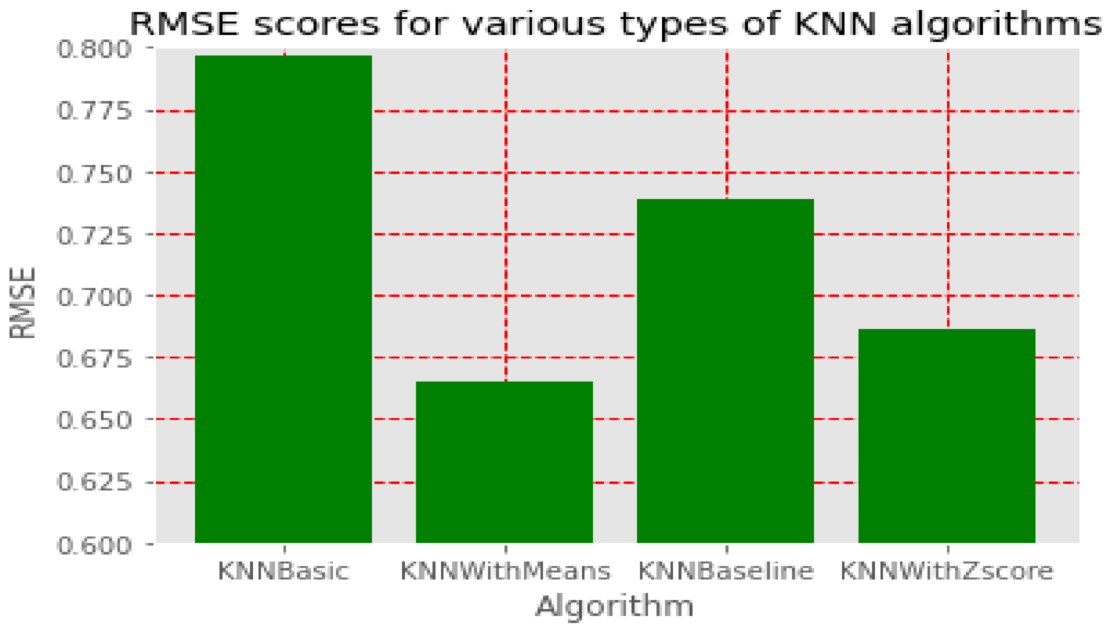
Testing and accuracy:
When testing the methods mentioned above, we used the method of k-fold cross validation. This works by first shuffling the dataset randomly and then splitting it into k-many groups. For each group we take it as the training data, fit it to the model and evaluate it against the test data (the remaining groups). The accuracy of this is calculated (RMSE) and the model is discarded. The process is repeated for the next group. The final accuracy is given by the average of the RMSE scores for each group. Due to the shuffling, splitting into groups and finding an average accuracy score as opposed to just one with test-split, this method of testing is seen to be less biased and more realistic (Brownlee, 2020).

Accuracy measure:
Our accuracy measure will be RMSE- root mean square error. This unit of measure cannot be compared to different datasets as a general comparison of accuracy as the scale is relative to the data being tested. The main objective is to compare the different results we get from each model and see which produces a better score and then choose parameters to

lower this score as much as possible. The lower the RMSE value the better as it means the amount of error in our predictions is less. (Moody, 2019)

Results:

After running and testing each algorithm as stated above, we get the following results for each algorithm:



From this we see that initially, the algorithm best suited to our dataset and so producing the most accurate predictions is the KNNWithMeans algorithm, the KNNWithZScore was a very close second. Also, in general if we look at how the basic version compares to the three others which attempt to remove bias, we can see a big difference in accuracy.

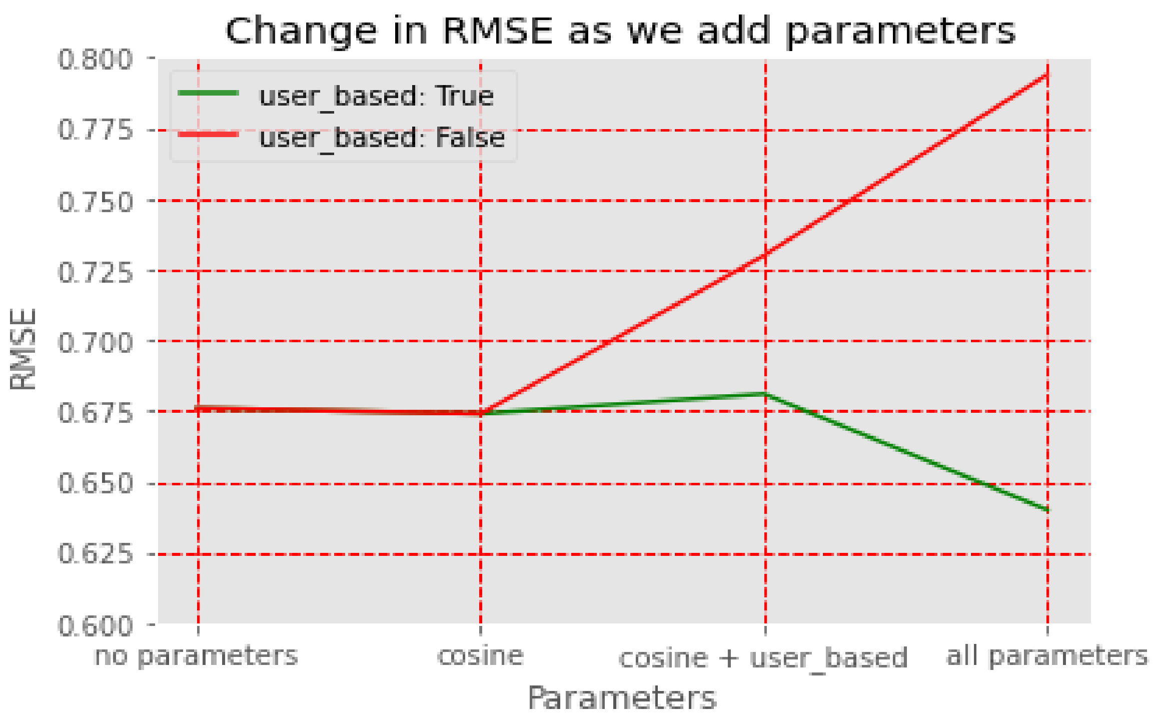
The next step is to test the different parameter combinations for this model to see which optimises the algorithm best.The algorithm has to test many different possibilities and as it automatically finds the most accurate combination, visualising the accuracy of all different possibilities was not necessary as we were able pick out the most accurate by finding which combination produced the lowest RMSE score. The outcome of this was the following parameters and the final RMSE score:

Hyperparameter tuning the chosen model:

Parameter	Best option
Similarity option	Cosine rule of similarity

Minimum support	9
Item/user based	User based
FINAL RMSE:	**0.640 (3.s.f)**

To illustrate that these parameters do in fact optimise and improve the overall accuracy, the following graph illustrates how the RMSE varies as we add the chosen parameter above one by one.



The graph above shows a negative correlation between the RMSE score and the addition of each parameter, showing that as we add each parameter, the RMSE becomes lower therefore more accurate. The green line represents the parameters which we went with in our final model, the red is without the user_based parameter being set to true. If we look at the green line, you will notice the negative correlation isn't consistent, there is an increase when we add the parameter user_based = true. Initially this appears to show that this parameter is in fact having a negative impact on the accuracy of the model however overall, with all parameters selected, this is not the case. This is why the red line is included, to illustrate what happens if we instead set the user_based parameter to false. The difference becomes clear and shows that although there appears to be a slightly negative impact on the accuracy initially, this is not the case when considering all parameters together. Overall,

the impact becomes positive as the final RMSE value is much lower when set to true. Setting the user_based parameter to false causes it to become less accurate than without the parameters* being added.
*Note that the no parameters does not mean the model has none, it has defaults. Here we mean no parameters changed or added.

Discussion:

To sum up our results above, we found that the best algorithm was the KNNWithMeans algorithm and the parameters that optimise its accuracy on our dataset are: using the cosine rule of similarity, having the minimum support set to 9 and making the algorithm user-based.

As this is our chosen model we will now look into how it works a little more. The formula which is in the algorithm to calculate similarity can be seen below:

$$\hat{r}_{ui} = \mu_u + \frac{\sum\limits_{v \in N_i^k(u)} \text{sim}(u, v) \cdot (r_{vi} - \mu_v)}{\sum\limits_{v \in N_i^k(u)} \text{sim}(u, v)}$$

*This formula is part of the surprise python package, the formula can be found here (*Surprise. n.d.)*, also a reference to the notation used above is found here* (Surprise, n.d.-b)

Firstly, there are two variations of this formula, the one above is for the user-based option which we used and the other would be item-based. The difference is that the mean is based on users not items and the similarity calculation (in our case done with cosine) will compare users not items.

Cosine rule of similarity:
The cosine rule is used in the formula where it says sim(U,V). The cosine rule involves modelling each users ratings as a vector (U and V), then finding the angle (and so the difference) between them. The formula can be seen like this:

$$cos\theta = \frac{U \cdot V}{||U||||V||}$$

Mean normalisation:
After calculating similarity between them using the cosine rule above, we then need to normalise the result. The part of the formula relating to the mean normalisation is the mention of μ_u. The similarity is normalised using the mean in order to remove bias in ratings (some people can rate harshly in general or more lenient) and so the results are adjusted accordingly to compensate for this.

Evaluation:

Of the 4 initial models considered, KNN Basic was the algorithm which included no attempt at normalisation and so can be considered a base model for comparison to our final model (the final model is an enhanced version of the KNN Basic model).

We can see that finding the most accurate of the enhanced versions (including some sort of normalisation on top of the base model), then fine tuning that version, proved to give much better accuracy. The RMSE of the base model we began with was 0.797(3.s.f), but by the end, we managed to produce a model which gave an RMSE of 0.640 (3.s.f), a decrease of 19.7%(3.s.f). This shows a clear improvement in accuracy. Note that this could be more had we used a standard test-split method. However this way of measuring accuracy is more optimistic, so not only did we see improved accuracy but this improved accuracy is more reliable and close to realistic.

The tests showing that a user-based option would be more accurate was in line with what we expected as it often is the case that item based is mainly better when dealing with sparse data and larger datasets (Abhinav Ajitsaria, n.d.). Also the cosine rule of similarity is generally considered more accurate than msd (mean square difference) as msd considers distance between points which can be inaccurate at times for similarity. This is because points that are more aligned would be considered more likely to be similar than ones which are just closer in distance (Abhinav Ajitsaria, n.d.). This is considered when calculating the angle between them (cosine rule of similarity) but not with msd which just finds distance between them. Finally the algorithm itself makes sense as although the KNN With ZScore algorithm is an extension of this one, that also factors in standard deviation, this dataset had a small range for ratings and as this was controlled (only values between 0-2 were selected so no anomalies), there was no need to normalise in this way, so it did not have a noticeable positive impact. Perhaps the reason it was slightly less accurate compared to KNNWithMeans was because it was overcomplicating the solution needed.

Further Recommendations

Looking to the future, it is important to consider how this model can be scaled as the dataset grows and evolves. Firstly, we would expect that as the dataset grows, it would be better to switch to an item-based model. This is because item-based models are often better suited when dealing with large datasets with a lot more ratings than items, it proves to be more stable and accurate (Abhinav Ajitsaria, n.d.). Very importantly too, the item-based model is much faster as when you have many ratings for many users it can be time consuming to keep comparing all users and their ratings. Item based however focuses on that particular item and its ratings and how that is similar to other items, making it much quicker as there are a lot less items. The reason it would be more stable and accurate is because again with many users and ratings, it can be easy for individual user ratings to fluctuate however a specific item which has been rated by many different users is not likely to exhibit this behaviour as you would need a bigger change in ratings from many users to see a big difference as opposed to individual users and there own smaller set of ratings.

Also another issue to discuss going forward is, as the dataset grows and you start to notice sparse data, switching the model altogether to an algorithm such as SVD (singular value decomposition) which uses matrix factorisation would make more sense. This is because this type of algorithm is better suited to dealing with sparse data and large datasets, also there is potential then, as you have more data, to start improving the model, introducing latent factors as features of the model (Grover, 2020), making the recommendations smarter. A potential latent factor that would be interesting to explore would be the impact of location of the restaurant. Perhaps this is something important to consider as nearer restaurants would be more reasonable recommendations or perhaps the user has a tendency to travel far to other restaurants in which case, further away restaurants might be better. Or maybe users have a tendency to eat in certain locations more than others so restaurants in those areas would be better to recommend higher.

Bibliography

References:
Abhinav Ajitsaria. (n.d.). Build a Recommendation Engine With Collaborative Filtering. Real Python. Retrieved 1 March 2021, from https://realpython.com/build-recommendation-engine-collaborative-filtering/

Grover, P. (2020, July 16). Various Implementations of Collaborative Filtering - Towards Data Science. Medium. https://towardsdatascience.com/various-implementations-of-collaborative-filtering-100385c6dfe0
Moody, J. (2019, September 6). What does RMSE really mean? - Towards Data Science. Medium. https://towardsdatascience.com/what-does-rmse-really-mean-806b65f2e48e

Surprise. (n.d.). k-NN inspired algorithms — Surprise 1 documentation. Retrieved 1 March 2021, from https://surprise.readthedocs.io/en/stable/knn_inspired.html#surprise.prediction_algorithms.knns.KNNWithMeans

Surprise. (n.d.-b). Notation standards, References — Surprise 1 documentation. Retrieved 1 March 2021, from https://surprise.readthedocs.io/en/stable/notation_standards.html

Brownlee, J. (2020, August 3). A Gentle Introduction to k-fold Cross-Validation. Machine Learning Mastery. https://machinelearningmastery.com/k-fold-cross-validation/#:%7E:text=Cross%2Dvalidation%20is%20a%20resampling,k%2Dfold%20cross%2Dvalidation.

Zoltan, C. (2018, November 3). KNN in Python - Towards Data Science. Towards Data Science. https://towardsdatascience.com/knn-in-python-835643e2fb53