Food.com Interaction and Rating Prediction

CSE 158 Fall 2022

Introduction

When forming recommendations for users, it can be a crucial step to predict what the result of an interaction between a user and an item would be before recommending it. Even further, it can be important to predict whether the user will use that recommendation and even interact with the item. To examine this, we used a dataset containing over 180000 recipes and 700000 recipe ratings on Food.com spanning from 2000 to 2018. In this project, we created a model that will predict is a specific user will interact with a recipe, and subsequently predict the rating left by the user if an interaction is predicted to occur. This would be a useful task since we would not want to recommend recipes to users that they would not use, and we would recommend recipes that have higher predicted ratings to improve the user experience.

Exploratory Data Analysis

Before generating predictive models with the recipe data, we first sought out to understand the fields within the data and use exploratory data analysis to gain a preliminary understanding of the datasets. In this project, we are using five datasets: recipes, users, three interactions datasets (train, validation, test).

The recipes dataset has 8 fields. The *id* and *i* fields contain the unique recipe identifier, where *i* is just encoded so the id's begin from 0. The *name_tokens* is a BPE-tokenized recipe name. The *ingredient_ids* and *steps_tokens* have lists of ingredients and specific steps required, both represented as lists. The *techniques* field is a one-hot encoded field to express the cooking technique that a particular recipe follows.

```
recipes = pd.read_csv('data/PP_recipes.csv').sort_values(by = 'i').set_index('i')
recipes.head()
```

i	id	name_tokens	ingredient_tokens	steps_tokens	techniques	calorie_level	ingredient_ids
0	40893	[40480, 1454, 16201, 2056, 955, 541, 11332, 82...	[[1424, 8876, 11007], [3464, 21453], [38966, 2...	[40480, 40482, 23667, 11007, 240, 21453, 240, ...	[0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, ...	0	[3384, 7979, 2127, 3502, 3217, 1257, 2778, 500...
1	44394	[40480, 34712, 22683, 11274, 5409, 29868, 40481]	[[5343, 535, 2044, 5409, 7087], [17869, 6020],...	[40480, 40482, 12172, 1281, 5409, 7087, 240, 6...	[1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...	0	[912, 7557, 2499, 5382]
2	85009	[40480, 12187, 13995, 571, 14719, 40481]	[[1061, 494, 813, 2141], [31843], [30645, 4785...	[40480, 40482, 2572, 19472, 31757, 512, 823, 4...	[1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, ...	2	[4623, 6265, 1168, 6016, 3597, 3440, 7213, 169...
3	134728	[40480, 27, 12, 263, 4017, 1494, 273, 14122, 4...	[[5867, 34056], [17918], [15022, 8361], [6953]...	[40480, 40482, 7087, 17918, 240, 6953, 488, 11...	[1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...	1	[1304, 2683, 3217, 6270, 3532, 869, 7557, 3698...
4	200236	[40480, 16212, 11880, 556, 19811, 12772, 22075...	[[16212, 1535], [6953], [1430, 1301, 11332], [...	[40480, 40482, 2525, 10906, 485, 29848, 40479,...	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...	2	[4130, 6270, 3486, 7557, 5010, 3203, 2683, 125...

The users dataset has 6 fields. Similar to the recipes dataset, the *u* column has user id's beginning from 0. The *techniques* field contains all the cooking techniques for the user from interacting with past recipes. The *items* and *ratings* fields are lists containing the items users interacted and their ratings, respectively. The *n_items* and *n_ratings* fields have the total number of recipes rated.

```
users = pd.read_csv('data/PP_users.csv', index_col = 'u')
users.head()
```

u	techniques	items	n_items	ratings	n_ratings
0	[8, 0, 0, 5, 6, 0, 0, 1, 0, 9, 1, 0, 0, 0, 1, ...	[1118, 27680, 32541, 137353, 16428, 28815, 658...	31	[5.0, 5.0, 5.0, 5.0, 5.0, 5.0, 4.0, 4.0, ...	31
1	[11, 0, 0, 2, 12, 0, 0, 0, 0, 14, 5, 0, 0, 0, ...	[122140, 77036, 156817, 76957, 68818, 155600, ...	39	[5.0, 5.0, 5.0, 5.0, 5.0, 5.0, 5.0, 5.0, 5.0, ...	39
2	[13, 0, 0, 7, 5, 0, 1, 2, 1, 11, 0, 1, 0, 0, 1...	[168054, 87218, 35731, 1, 20475, 9039, 124834,...	27	[3.0, 4.0, 4.0, 4.0, 4.0, 4.0, 4.0, 5.0, ...	27
3	[498, 13, 4, 218, 376, 3, 2, 33, 16, 591, 10, ...	[163193, 156352, 102888, 19914, 169438, 55772,...	1513	[5.0, 5.0, 5.0, 5.0, 4.0, 4.0, 5.0, 5.0, 5.0, ...	1513
4	[161, 1, 1, 86, 93, 0, 0, 11, 2, 141, 0, 16, 0...	[72857, 38652, 160427, 55772, 119999, 141777, ...	376	[5.0, 5.0, 5.0, 5.0, 4.0, 4.0, 5.0, 4.0, 5.0, ...	376

Each row in the interactions data contains an individual interaction between a specific user and recipe. The *user_id*, *recipe_id*, *u*, and *i* columns contain unique identifiers. The *date* field is a string representing the date. The *rating* column contains the rating from that interaction. If the rating is 0, that means there was an interaction, but the user did not rate the recipe.

```
train_interactions = pd.read_csv('data/interactions_train.csv').drop(columns = ['u', 'i'])
train_interactions.head()
```

	user_id	recipe_id	date	rating
0	2046	4684	2000-02-25	5.0
1	2046	517	2000-02-25	5.0
2	1773	7435	2000-03-13	5.0
3	1773	278	2000-03-13	4.0
4	2046	3431	2000-04-07	5.0

From a visual test, we noticed that the *n_items* and *n_ratings* column tended to have the same values. We verified this by using a Series equality to check if the two columns had matching values. Since their values matched, we dropped the *n_items* column from the users dataset.

There are 178265 recipes, 25076 users, and 718379 interactions. The training, validation, and test interactions comprise 97.28%, 0.98%, and 1.73% of the interactions, respectively.

After checking the fields and conducting some data processing, we now sought to generate insights about the dataset.

By sorting the dataset by the number of user ratings, we were able to find the 15 users who left the most ratings, and designated them as our 15 most active users. There was a single user who left 6437 ratings, which was significantly higher than majority of users.

```
top15 = users.sort_values(by = 'n_ratings', ascending = False).iloc[:15]
top15 # These are the 15 users with the most number of ratings
```

u	techniques	items	ratings	n_ratings
94	[2508, 26, 18, 1319, 1516, 1, 5, 123, 29, 2488...	[91791, 3798, 150824, 101819, 32984, 137191, 1...	[4.0, 5.0, 4.0, 5.0, 4.0, 5.0, 5.0, 5.0, 5.0, ...	6437
275	[977, 19, 15, 772, 981, 2, 9, 164, 28, 1556, 3...	[149428, 147083, 83459, 11578, 70918, 78548, 8...	[5.0, 5.0, 4.0, 5.0, 5.0, 5.0, 5.0, 5.0, 5.0, ...	4581
193	[997, 46, 2, 531, 613, 2, 3, 144, 21, 1206, 33...	[135512, 164305, 41269, 133049, 134610, 85304,...	[5.0, 5.0, 4.0, 4.0, 4.0, 4.0, 5.0, 5.0, 3.0, ...	3656
241	[1112, 15, 12, 638, 979, 2, 0, 72, 11, 1180, 3...	[76442, 164837, 5559, 149181, 3020, 39614, 148...	[0.0, 4.0, 4.0, 4.0, 0.0, 5.0, 5.0, 1.0, 5.0, ...	3465
208	[1175, 27, 6, 631, 713, 2, 1, 76, 22, 1187, 22...	[48637, 136663, 97290, 45833, 161663, 26019, 1...	[5.0, 5.0, 5.0, 5.0, 5.0, 5.0, 5.0, 4.0, 5.0, ...	3338
130	[1225, 16, 6, 606, 584, 3, 0, 81, 16, 1073, 16...	[136789, 122200, 46974, 15392, 61373, 95418, 3...	[5.0, 5.0, 5.0, 5.0, 5.0, 5.0, 5.0, 5.0, 5.0, ...	2851
564	[818, 3, 6, 429, 856, 0, 3, 59, 13, 855, 81, 6...	[90424, 14622, 152154, 35866, 53437, 165607, 6...	[5.0, 5.0, 5.0, 5.0, 5.0, 5.0, 5.0, 5.0, 5.0, ...	2798
164	[748, 19, 13, 401, 679, 1, 2, 86, 18, 930, 23,...	[29224, 40400, 155771, 9161, 6413, 97060, 6374...	[5.0, 5.0, 4.0, 5.0, 4.0, 4.0, 4.0, 3.0, 4.0, 5.0, ...	2733
319	[518, 14, 11, 492, 679, 3, 4, 63, 17, 795, 15,...	[165762, 79467, 125, 150899, 30739, 84214, 488...	[5.0, 0.0, 0.0, 5.0, 5.0, 5.0, 5.0, 5.0, 4.0, ...	2602
267	[592, 37, 13, 275, 616, 1, 4, 117, 18, 803, 19...	[4143, 55906, 53667, 112057, 104729, 121380, 4...	[5.0, 5.0, 5.0, 4.0, 5.0, 5.0, 5.0, 5.0, 5.0, ...	2457
100	[623, 27, 11, 228, 466, 3, 3, 140, 20, 667, 64...	[141404, 146307, 128907, 90616, 11113, 119974,...	[5.0, 4.0, 4.0, 4.0, 5.0, 3.0, 3.0, 5.0, 2.0, ...	2432
56	[568, 41, 8, 263, 559, 3, 3, 106, 24, 662, 30,...	[176781, 8047, 160999, 111487, 101135, 130048,...	[5.0, 5.0, 5.0, 5.0, 5.0, 5.0, 5.0, 5.0, 5.0, ...	2377
260	[658, 10, 3, 199, 501, 2, 1, 75, 7, 607, 107, ...	[165343, 66375, 110295, 177845, 74581, 61836, ...	[5.0, 5.0, 5.0, 3.0, 4.0, 4.0, 4.0, 4.0, 4.0, ...	2242
245	[622, 32, 8, 302, 430, 1, 1, 81, 13, 752, 17, ...	[65040, 102532, 28949, 112245, 3300, 56058, 12...	[5.0, 5.0, 5.0, 5.0, 5.0, 5.0, 4.0, 5.0, 5.0, ...	2232
345	[681, 28, 5, 222, 455, 1, 1, 93, 15, 645, 23, ...	[119921, 69764, 146496, 152585, 177180, 54464,...	[4.0, 4.0, 0.0, 5.0, 5.0, 5.0, 5.0, 5.0, 4.0, ...	2046

Next, we sought to find the average ratings of the top 15 users. The motivation behind this is because there may be users who tend to leave ratings with a broad range of values, while others either solely leave exceed-

ingly positive reviews or scathing reviews. Since the ratings for an individual user were stored as a string representation of a list, we applied functions to first evaluate the string and then compute the mean of the list. It was visible that user 94 had a high mean rating of around 4.966, while user 260 had a lower mean rating of around 4.238.

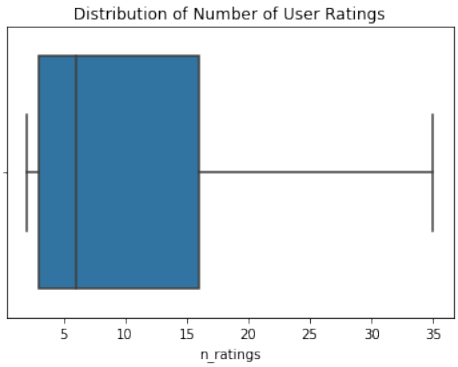
```
top15['ratings'] = top15['ratings'].apply(ast.literal_eval)
top15['mean_rating'] = top15['ratings'].apply(np.mean)
```

```
top15
```

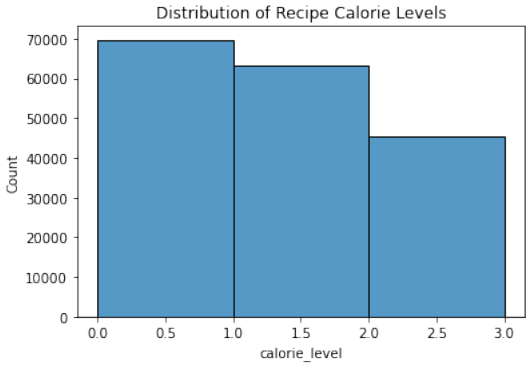
u	techniques	items	ratings	n_ratings	mean_rating
94	[2508, 26, 18, 1319, 1516, 1, 5, 123, 29, 2488...	[91791, 3798, 150824, 101819, 32984, 137191, 1...	[4.0, 5.0, 4.0, 5.0, 4.0, 5.0, 5.0, 5.0, 5.0, ...	6437	4.966755
275	[977, 19, 15, 772, 981, 2, 9, 164, 28, 1556, 3...	[149428, 147083, 83459, 11578, 70918, 78548, 8...	[5.0, 5.0, 4.0, 5.0, 5.0, 5.0, 5.0, 5.0, 5.0, ...	4581	4.783672
193	[997, 46, 2, 531, 613, 2, 3, 144, 21, 1206, 33...	[135512, 164305, 41269, 133049, 134610, 85304,...	[5.0, 5.0, 4.0, 4.0, 4.0, 4.0, 5.0, 5.0, 3.0, ...	3656	4.754923
241	[1112, 15, 12, 638, 979, 2, 0, 72, 11, 1180, 3...	[76442, 164837, 5559, 149181, 3020, 39614, 148...	[0.0, 4.0, 4.0, 4.0, 0.0, 5.0, 5.0, 1.0, 5.0, ...	3465	4.696104
208	[1175, 27, 6, 631, 713, 2, 1, 76, 22, 1187, 22...	[48637, 136663, 97290, 45833, 161663, 26019, 1...	[5.0, 5.0, 5.0, 5.0, 5.0, 5.0, 5.0, 4.0, 5.0, ...	3338	4.908328
130	[1225, 16, 6, 606, 584, 3, 0, 81, 16, 1073, 16...	[136789, 122200, 46974, 15392, 61373, 95418, 3...	[5.0, 5.0, 5.0, 5.0, 5.0, 5.0, 5.0, 5.0, 5.0, ...	2851	4.938969
564	[818, 3, 6, 429, 856, 0, 3, 59, 13, 855, 81, 6...	[90424, 14622, 152154, 35866, 53437, 165607, 6...	[5.0, 5.0, 5.0, 5.0, 5.0, 5.0, 5.0, 5.0, 5.0, ...	2798	4.834882
164	[748, 19, 13, 401, 679, 1, 2, 86, 18, 930, 23,...	[29224, 40400, 155771, 9161, 6413, 97060, 6374...	[5.0, 5.0, 4.0, 5.0, 4.0, 4.0, 4.0, 3.0, 4.0, 5.0, ...	2733	4.719356
319	[518, 14, 11, 492, 679, 3, 4, 63, 17, 795, 15,...	[165762, 79467, 125, 150899, 30739, 84214, 488...	[5.0, 0.0, 5.0, 5.0, 5.0, 5.0, 5.0, 5.0, 4.0, ...	2602	4.636434
267	[592, 37, 13, 275, 616, 1, 4, 117, 18, 803, 19...	[4143, 55906, 53667, 112057, 104729, 121380, 4...	[5.0, 5.0, 5.0, 4.0, 5.0, 5.0, 5.0, 5.0, 5.0, ...	2457	4.812373
100	[623, 27, 11, 228, 466, 3, 3, 140, 20, 667, 64...	[141404, 146307, 128907, 90616, 11113, 119974,...	[5.0, 4.0, 4.0, 4.0, 5.0, 3.0, 3.0, 5.0, 2.0, ...	2432	4.773438
56	[568, 41, 8, 263, 559, 3, 3, 106, 24, 662, 30,...	[176781, 8047, 160999, 111487, 101135, 130048,...	[5.0, 5.0, 5.0, 5.0, 5.0, 5.0, 5.0, 5.0, 5.0, ...	2377	4.944468
260	[658, 10, 3, 199, 501, 2, 1, 75, 7, 607, 107, ...	[165343, 66375, 110295, 177845, 74581, 61836, ...	[5.0, 5.0, 5.0, 3.0, 4.0, 4.0, 4.0, 4.0, 4.0, ...	2242	4.237734
245	[622, 32, 8, 302, 430, 1, 1, 81, 13, 752, 17, ...	[65040, 102532, 28949, 112245, 3300, 56058, 12...	[5.0, 5.0, 5.0, 5.0, 5.0, 5.0, 4.0, 5.0, 5.0, ...	2232	4.670699
345	[681, 28, 5, 222, 455, 1, 1, 93, 15, 645, 23, ...	[119921, 69764, 146496, 152585, 177180, 54464,...	[4.0, 4.0, 0.0, 5.0, 5.0, 5.0, 5.0, 5.0, 4.0, ...	2046	4.740469

Next, we compared the average mean rating of the 15 most active users with the average mean rating of all users. The average for the top 15 users was 4.761, which was 0.309 greater than the average for all users.

Another important analysis was for the number of ratings per user. Since we would factor in past interactions for users in order to predict new interactions, it was important to understand how many ratings users have posted. Since the number of ratings is heavily right skewed, the mean, 27.871, was significantly higher than the median, which was 6.000. The first and third quadrants were 3 and 6, respectively. We subsequently visualized the spread of the number of user ratings while ignoring outliers. Outliers were values above 35.5, which is the $Q3 + 1.5 * IQR$.

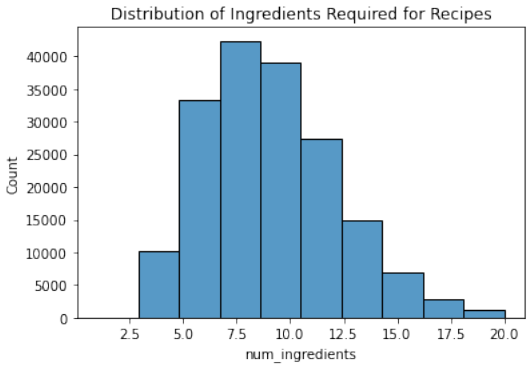


Next, we wanted to plot the distribution of calorie levels present in the recipes. It may be possible that higher calorie recipes yield dishes that taste better, so they may be rated higher. However, a low calorie recipe can also have very high ratings if there are individuals on diets using the recipe and they like the steps. From our analysis, there was a good representation of the three calorie levels in the dataset, although not uniformly distributed. Around 40% of the recipes had calorie level 0, which was the most common category. The least common calorie level was 2.

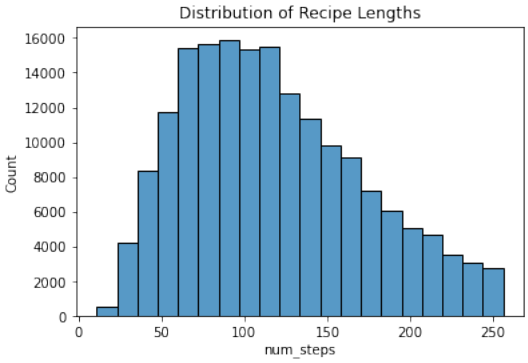


Two other features we believe can have a strong effect on the rating were the number of ingredients and number of steps required to cook the dish. For example, a recipe with too many ingredients may have negative ratings due to the amount of preparation. Recipes with fewer ingredients and steps may have higher ratings since it is more convenient for working individuals. Since both the 'ingredient_ids' and 'steps_tokens' columns were stored as string representations of lists, we evaluated them as lists and applied the length function. We wanted to use a histogram to visualize the continuous distribution.

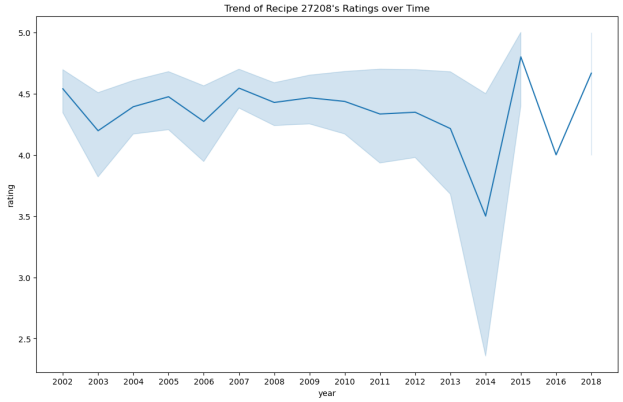
The distribution of ingredients required was slightly skewed right, with the mean only 0.006 greater than the median. On average, recipes had 9 ingredients and the most ingredients required for an individual recipe was 20.



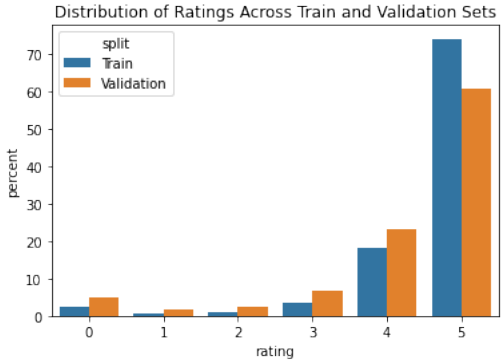
The distribution of steps required was also skewed right, with the mean being 7.855 greater than the median. On average, recipes had 110 steps and the highest number of steps required was 256, while the smallest number of steps was 11.



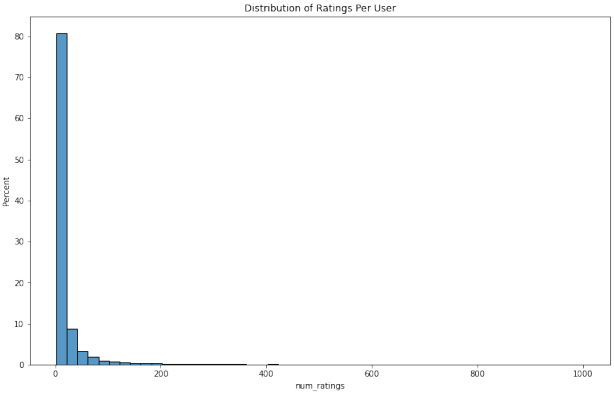
Since the interactions span from 2000 to 2018, there is a high probability that older recipes would have a chance to be revised based on feedback from reviews. For example, if a rating posted in 2005 was rated poorly for not having clearly written steps, the recipe author could revise this recipe and subsequently see improvement in the ratings. We decided to see if the most frequently rated recipe's ratings improved over time. After constructing a hash map of recipe frequencies in the training interactions and sorting the recipes by frequency, we found that recipe 27208 was reviewed the most, for a total of 1091 times. We extracted the years from the date field, and saw that the ratings spanned from 2002 to 2018 for this recipe. From the graph below, we can see the rating for this recipe, on average, was around 4.4, except in 2014, when the ratings were significantly lower. There were many more low ratings for this recipe, which explains why the mean rating dropped to around 3.5.



When training and testing a model, we want the validation set to be representative of the overall interactions, so we wanted to check that the distribution of ratings across the train and validation sets did not differ significantly. We grouped by the split and found the counts of each rating. We then converted these counts to percentages since the totals are vastly different, and plotted a grouped bar chart to display the rating distributions for train and validation splits. From the figure below, we can see that for both splits, there is a higher percentage of 4's and 5's. The general trends were the same, indicating no significant difference in the splits on the basis of rating.



Next, within the training set, we plotted the distribution of number of ratings per user while removing outliers. As expected, there was a strong right skew to this distribution since users can rate as many recipes as they like, but the minimum is one. The mean was around 21 higher than the median of 6 recipes rated.



In this dataset, ratings from 1 to 5 mean there was an interaction and a rating. However, our dataset contains occurrences of interactions without ratings, which are ratings of 0. In the last part of our exploratory data analysis, we wanted to check that this was equally represented across the splits. Around 2.43% of the train interactions had no ratings, while 5.07% of the validation interactions had no ratings. This seemed like a significant difference due to the sample size.

Predictive Task

Based on the selected dataset, a predictive task can be studied regarding user interaction and rating. In this assignment, we will create a model that, given a unique user ID and recipe ID pair, will predict if that specific user would interact with the recipe, and if so, what rating they would leave. If the model predicted the interaction as existing, our model will also indirectly predict whether or not the user would leave a rating on the recipe, since based on the dataset, rating predictions are in the interval [0, 5], where a rating of 0 means that a user interacted with the recipe but did not leave a rating.

To evaluate our model for this predictive task, we will split the dataset into training, validation, and testing sets. Using the training set for model development, the vali-

dation set for hyperparameter tuning, and the test set to check model performance, will allow us to evaluate our model. Specifically, since the values for interaction in a user ID recipe ID pair are known, in addition to the rating, for each split of the dataset, we will calculate the accuracy of the model on the validation and testing sets, and compare the accuracy to that of our baseline.

Our baseline will be as follows: Given a unique user ID rating ID pair, if the recipe ID is in the top 50% of most interacted with recipes, we predict that the given user interacted with the recipe. In the case that an interaction is predicted, we then use the average of all ratings of that user across all recipes they interacted with in the training set as their ratings prediction. If the user never interacted with any recipes in the training set, we predict that their rating is the global average rating.

Using the train, validation, and test split of our data, we can calculate multiple measures of validity that would indicate the quality of our model's predictions, and improvement compared to the baseline. Since we know the true values for interaction and ratings for all users and recipes in our train, validation, and test data splits, we can utilize this to understand how well or not our model is performing.

We will check the accuracy of our baseline and our model on the train and validation sets, and use results from these tests to further optimize our model and gauge if improvements are being made. Once we decide on a model that we think will work best, we will calculate the accuracy of the baseline and model on our test data, to let us know the outcome of our prediction attempts, and if the features and methods we used were reasonable choices.

We will similarly calculate the RMSE of the baseline and model on the train and validation sets, finalize our model, and then calculate the RMSE again of the test set to gauge how our model performed. We will also calculate the recall and precision of our model on the test set.

The features used for the baseline include the number of times a recipe was interacted with, which was extracted using groupby on recipe IDs on the complete dataset and the count concatenation method. We also used the global average rating, which was calculated by taking the mean of the ratings column from the complete dataset.

We also used interactions and relative popularity of recipes as a feature in our baseline, and this was extracted from the complete dataset by grouping the training dataframe by recipe id, getting the counts of associated user IDs, presenting the number of interactions per recipe, and then calculating the count for each recipe. We then sorted the recipes by these counts and made a list of recipe IDs that had the top 50% of interactions, using cumulative sum.

We created a dictionary with users as keys and their ratings as a list that was the value for each key, and used this dictionary to find the proportion of times a user left a rating, and used this as a feature in our baseline as well.

The features used for the model include the number of times each recipe was interacted with, extracted as described for the baseline. The dictionary with users as keys and the list of their ratings as values was used to generate many features: the proportion of times a user left a rating in their past interactions, their average rating, and the number of interactions they had. A dictionary was also created with keys being recipe ids, and values being their corresponding ratings, and the average rating per recipe calculated from this data structure was also used as a feature.

A dictionary of interaction per user was created, with user IDs as keys, and a list of tuples of corresponding interactions with recipe IDs and ratings as values. A dictionary of ratings per recipe was also created, with recipe IDs as keys, and a list of tuples of corresponding interacted user IDs and ratings as values. These data structures were used to calculate user-user and recipe-recipe Jaccard similarities, that were used as features in our models.

Model Description and Evaluation

For the first part of our model, we tackled the task of predicting given a user ID and a recipe ID, if that user interacted with that recipe or not. First, we concatenated our training and validation sets into a larger combined set, to give us more data on which to train our model. In addition, because our dataset is such that the existence of a record containing a unique user ID recipe ID combination insinuates that an interaction occurred between that user and the corresponding recipe, we added negative interaction user ID recipe ID combination for each user in the concatenated training and validation dataset.

The features used for the model included popularity, gauged by the number of times a recipe was interacted with. This was extracted for each recipe ID from the complete dataset as described previously.

We also created a dictionary of interactions per user, that, given a key of user ID, gave a list of that user's interactions as a list of tuples of recipe IDs and corresponding ratings. Similarly, we used a dictionary of ratings per recipe, that, given a recipe ID, gave a list of that recipe's interactions as a list of tuples of user IDs and their corresponding ratings.

We then created helper functions to calculate Jaccard similarity. Specifically, we created a Jaccard similarity function to calculate user-user similarity using passed user ID and recipe ID, by finding the similarity between users who interacted with the passed recipe ID, and the users who interacted with all other recipes that the user

interacted with, in the concatenated training and validation dataset, using the aforementioned dictionaries. Similarly, we created a Jaccard similarity function to calculate recipe-recipe similarity using passed user ID and recipe ID, by finding the similarity between recipes interacted with by the passed user ID, and the recipes interacted with by all other user that the recipe was interacted with by, in the concatenated training and validation dataset, again using the aforementioned dictionaries

These features, number of times the recipe was interacted with, the maximum user-user Jaccard similarly, and the maximum recipe-recipe Jaccard similarity, were all used as features for this model, and were used to generate a balanced logistic regressor with C=1.

We first incorporated raw interaction numbers for the passed recipe ID and the logistic regressor to make predictions about whether a recipe was interacted with or not. If the passed recipe ID was in the top 50 most interacted with recipes (has at least 9 interactions), we predicted that the user interacted with it. If the passed recipe ID had not been interacted with that many times, then we used the regressor to predict whether an interaction occurred or not. This gave an accuracy on our training-validation set of 0.932. We then created a predator that would only use the regressor, and accuracy on the training-validation set increased to 0.977. These show a huge improvement compared to the baseline, which had an accuracy of 0.457.

We did not really have many issues with scalability or overfitting at this step. We decided that heuristics would be sufficient for this step, given the size of our training-validation dataset, the vast range of values in the interaction matrix, and the fact that we were using a logistic regressor combined with features that are sensibly correlated to whether an interaction occurred or not.

We did however choose to use the maximum of Jaccard similarities for user-user and recipe-recipe interactions, since the data was sparse and many Jaccard similarities were 0 or close to 0. We also chose to decide on using a model that only used the number of interactions as a feature in the logistic regressor, instead of having the number of interactions be a conditional, which if not passed, would only then use the logistic regressor for prediction. This is because within the recipes that had the top 50% of interactions, there was a huge range of interactions, with the most interacted recipe having 1091 interactions, while the last one still in the list had only 9 interactions. Using these values as the first method of prediction, before using the regressor, did not seem to be a choice that would give the model high accuracy, and was therefore discarded in favor of only using the regressor.

For the second part of the model, which was predicting given an user ID and recipe ID pair, what rating the user would leave on the recipe, we first began by reusing the model we decided on to predict whether an interaction occurred, retraining it on the output we want this time, which is a number in the range [0, 5], instead if True or False for interaction prediction, where 0 means that no rating was left by the user who interacted with the recipe. This model gave an accuracy of 0.596, which was still an improvement from the 0.457 of the baseline model.

We decided to continue using a balanced logistic regressor with C=1, and using the same setup, but decided to change old features slightly, or add more features that could relate to the rating. First, we changed the Jaccard features to use the mean user-user and recipe-recipe similarities as compared to the max that was being used previously. This still resulted in an accuracy of 0.596.

We then tried to add more features to our logistic regressor. We reverted the jaccard similarities to using the max, and then used the previously mentioned dictionary with users as keys and their values being a list of all that user's ratings, to generate more relevant features. We calculated the proportion of times that user left a rating, and their average rating as features. This resulted in an accuracy of 0.711.

Since the preprocessed dataset contained many more details that we could use as features, such as the number of steps in each recipe, its calorie level, and the number of techniques in a recipe and the number of techniques in the recipes each user interacted with, we wanted to try and extract these details to use in our training-validation subset. However when conducting a sanity check to map all the user IDs and recipe IDs in our training-validation subset to those in the preprocessed user and recipe datasets, the user IDs mapped back to the preprocessed user dataset, but the recipe IDs did not appear to, so we could not use these features.

We decided to calculate more features and add them on to the logistic regressor we were using, using the dictionary of user IDs and their corresponding lists of ratings. We added the number of interactions for each user, and the average recipe rating. However, this decreased our accuracy to 0.650.

This is likely due to the fact that the number of interactions for a user and the average rating for the recipe do not give us as much information about how that specific user is likely to rate the recipe, as compared to the proportion of times the user left a non-zero rating and the average rating they left, which is more specific to the user, and therefore more relevant predicting their rating.

This is why we settled on the logistic regressor that used the number of times the recipe was interacted with, the max of user-user and recipe-recipe Jaccard similarities, the proportion of times that user left a rating, and their average rating, which gave us an accuracy of 0.711.

Additionally, the reason for not using N most similar users or ratings as a subset to extract features

from in any predictions is because the interaction matrix is sparse, and because the number of similar users that have non-zero Jaccard similarity to a specific user ranges widely, weighting the N most similar users equally for all users does not make sense. Similarly, different recipes have a number of interactions that have values in a wide interval, and using data from N users for each recipe, and weighting data from each interaction equally does not make sense. This is because the N most similar users to a given user would give us more important information where the total non-zero similarity users are close to N, as compared to where there are many more than N similar users, and their similarities have negligibly differing values. With a similar logic for the similarity between recipes, we can explain why we did not use an N most similar-based model in generating our predictions.

Table 1: Training-Validation Set Accuracy

Model Type	Baseline	Final Model
Interaction Prediction	0.457	0.977
Rating Prediction	0.457	0.711

Literature Review

This dataset was introduced in Generating Personalized Recipes from Historical User Preferences, published by the Association for Computational Linguistics in 2019 (Majumder et al., 2019).In this paper, the dataset was used to create new recipes based on user history, specifically recipe names and ingredient details. The personalized model used both natural language processing and recommender systems, with an encoder, decoder, and attention layers, and was preferred over the non-personalized baseline. Specifically, the algorithm takes in a recipe name, an ingredients list, and caloric level and outputs recipe instructions; user history can also be used to personalize this output. The data set used in this paper and in the Majumder, et. al. paper contains over 180,000 recipes and 700,000 user reviews with information on the recipes and users. For recipes, ingredient lists, tags, steps, techniques, and estimated time in both raw and tokenized form are provided. For users, review text, ratings out of five, and date of review are included in the data set.

In a paper focusing on using recurrent neural networks to generate complete recipes based on titles and ingredient lists, a dataset with 158,000+ recipes was used (Kiddon et al., 2016). However, this paper and dataset lacked any user interaction data, focusing more on types of recipes (e.g., appetizer recipes and dessert recipes) than user history.

Generally, it seems that many recipe recommenders leverage neural networks or deep learning. Earlier this year, a paper proposed a hierarchical graph attention network for recipe recommendation; it is a neural network based model that leverages user history, recipes, and relational information (Tian et al., 2022). This paper used a dataset based on recipes form Food.com that contains information on ingredients, measurements, and user-recipe interactions. The authors built a recipe graph relating ingredients, recipes, and users to each other for their graph learning approach.

Bangale, et. al. utilized a recipe dataset with over 4,000 recipes and content-based filtering algorithm to turn a user's inputted ingredients into a recommendation for a recipe (Bangale et al., 2022). The algorithm used involves similarity between recipes a user has previously interacted with and looks at different types of interactions (e.g., a user favorites a recipe). Specifically, cosine similarity was used, as well as natural language processing techniques like TF-IDF to extract recipe features.

It seems most approaches from previous work seem to benefit from user input and interaction. Additionally, utilizing similarity metrics seems to be a promising technique given its use by the Bangale, et. al. paper. However, we did not consider the power of natural language processing nor neural networks in our approach. Based on our strong increases in accuracy from our baseline model to our final model, which relies on popularity metrics and similarity metrics, we did not see the need to include raw text features. Additionally, no other existing work found seemed to utilize a recipe's popularity as a feature.

Results

Initially, to build off of the baseline model that utilizes popularity, a predictor with a popularity heuristic (given recipe was interacted with more than 9 times, or is in the top 50% of recipes) and a regression model based on user and recipe similarity and recipe popularity was used. This model was much better than the baseline and yielded a validation accuracy of 0.932. However, when the population was removed and regression was used exclusively, validation accuracy increased to 0.977.

When creating the rating predictor, several models were tried, including ones that incorporate the N most similar users or ratings. N most similar was not used due to sparse matrices and lack of clarity on how to weight different interactions. Similar to the interaction predictor, it was found that validation accuracy increased when only using regression and dropping any population heuristics. The final model used for predicting rating yielded a validation accuracy of 0.71 (up from a validation accuracy of 0.457 with the baseline) and a validation root mean squared error of 1.67 (compared to a validation RMSE on the baseline of 4.00).

With the final model to predict user-recipe interac-

tions, the test set of 12,455 interactions yielded a perfect accuracy of 1.0. Even with the model being trained on a balanced dataset, the model was able to predict the test set of all pairs of users and recipes that interacted correctly. This is a massive improvement from the baseline model, where the test set got an accuracy of 0.

The rating prediction model also saw great improvements. Since the baseline model yielded an accuracy of 0, the root mean squared error for the baseline rating model was extremely high at 5.38 and accuracy was again 0. With the final model, test root mean squared error improved to 2.16, and accuracy went up to 0.458. While the test results may not be as strong as the validation results, this performance is a massive improvement from the baseline. See Tables 1 and 2 for a summary of the test results.

It makes sense that models without heuristics performed better because heuristics are likely not very generalizable and too dependent on human bias. Incorporating popularity directly as a feature, was a much stronger option. With incorporating Jaccard similarity as a feature, both mean and max similarity was explored. Ultimately, maximum Jaccard similarity yielded higher accuracies.

For the interaction prediction model, the parameters are the popularity of the recipe, the maximum Jaccard similarity of the user and other users that interacted with the given recipe, and the maximum Jaccard similarity between the given recipe and other recipes that the user interacted with. The popularity of the recipe is simply how many times the recipe was interacted with, so as this increases, the likelihood the user interacted with the given recipe increases. If the given user is very similar to a user that interacted with the given recipe, this corresponds to a higher likelihood that the given user interacted with the given recipe. The same can be said about the recipe similarity.

For the rating prediction model, the parameters are the same 3 features used in the interaction model, along with the proportion of users that left a rating on the given recipe and the average rating a user has used. The interpretation of the 3 shared features is similar; more popular recipes likely have a higher rating, and higher similarities make the given user more likely to leave a rating for the given recipe. If a higher proportion of users left a rating on the recipe, this coincides with a higher chance that the given user will also leave a rating. Lastly, the average rating a user has left provides a "default" value for the model, where the other features inform the model on how to adjust the rating.

The final model succeeded relative to the others tried because the final model relied on a learned model trained off of a balanced dataset, whereas previous models tried to incorporate heuristics that perhaps don't generalize well.

Table 2: Test Set Accuracies

Model Type	Baseline	Final Model
Interaction Prediction	0.0	1.0
Rating Prediction	0.0	0.458

Table 3: Test Set Root Mean Squared Error

Model Type	Baseline	Final Model
Rating Prediction	5.38	2.16

References

Bangale, S., Haspe, A., Khemani, B., & Malave, S. (2022). Recipe recommendation system using content-based filtering. *SSRN Electronic Journal*.

Kiddon, C., Zettlemoyer, L., & Choi, Y. (2016). Globally coherent text generation with neural checklist models. *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, 329–339.

Majumder, B. P., Li, S., Ni, J., & McAuley, J. (2019). Generating personalized recipes from historical user preferences. *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*.

Tian, Y., Zhang, C., Metoyer, R., & Chawla, N. V. (2022). Recipe recommendation with hierarchical graph attention network. *Frontiers in Big Data*, *4*.