Recommender System using Decision Tree Classifier on Food Recipe Data

Tongxun Hu, Liuyao Zhang, Zhuoxuan Ju, and Bofu Zou
University of California, San Diego
t9hu, liz035, zju, bzou@ucsd.edu

1 Introduction

In nowadays data-driven world, recommender system has become one of the most important algorithms for us to analyze dataset and predict future trends. In this project, we want to predict a user's rating to a new recipe based on the user's past recipe rating records. We will use an interaction and recipe dataset found on Kaggle to perform our task.

We will clean the dataset and perform EDA to seek for potential useful features and get more clear idea on model selection. Then, we will use various classification and collaborative filtering models to build our recommender system. Lastly, we fine-tune our model in order to increase the model accuracy and look for future improvements.

2 Data Set

2.1 Identify a data set[1]

The dataset we are going to use is the recipes and interactions data from Kaggle. The data is crawled from Food.com (GeniusKitchen) online recipe aggregator. We select this dataset to perform our task because it's a very large dataset that includes 180K+ recipes and 700K+ recipe reviews covering 18 years (2000-2018) of user interactions and uploads on Food.com. Such a large volume and long timespan allow us to avoid some fundamental biases present in the dataset and analyze data that is inclusive and credible.

2.2 Basic Characteristics and Properties

The datasets that we are going to use are RAW_recipes.csv, and intersections_(train/validation/test).csv. The intersections_train csv dataset consists of total 698901 rows and 6 columns, including user_id, recipe_id, date, rating, u (User ID, mapped to contiguous integers from 0), and i (Recipe ID, mapped to contiguous integers from 0). The validation and test interaction datasets have very similar characteristics as the train dataset, so we are not going to repeatedly present the dataset here. The raw recipe dataset consists of total 231637 rows and 12 columns, including name, id, minutes, submitted, tags, nutrition, n_stepss, n_ingredients, etc. After loading in the dataset, we perform some basic operations to get the statistics and properties of both dataset.

For the interaction dataset:

- The mean of rating is 4.411, with a standard deviation of 1.265 and a median of 5.
- The interactions are pretty consistent and spread-out overtime, so the interactions do not suddenly increase or decrease at certain time points.

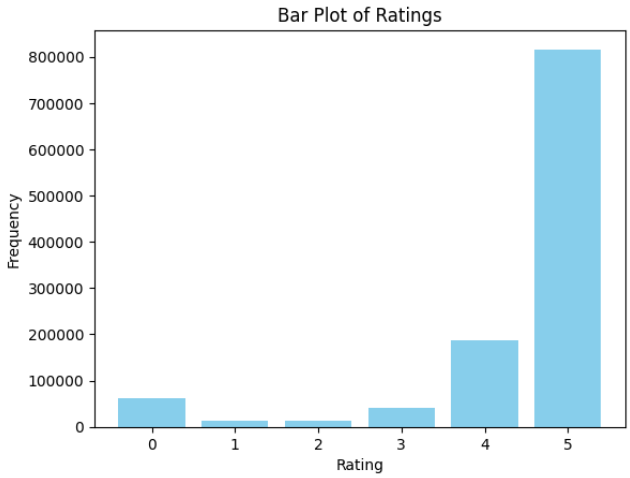
For the recipe dataset:

- The average number of steps for a recipe is 10 (9.765) with a standard deviation 5.995, a median of 9, a min of 0, and a max of 145.
- The mean number of ingredients for a recipe is 9 (9.051) with a standard deviation 3.735, a median of 9, a min of 1, and a max of 45.

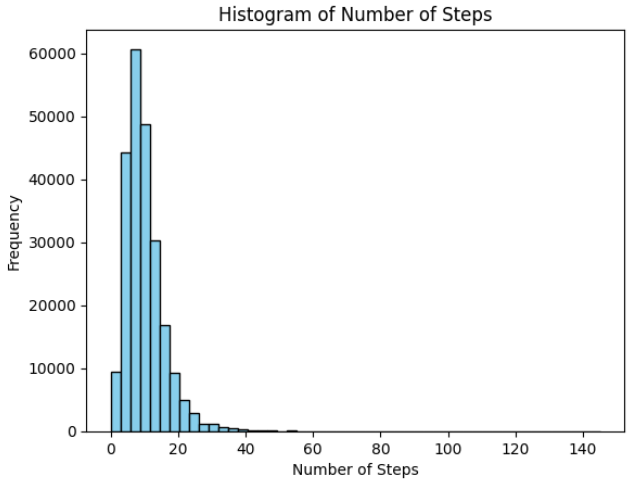
2.3 EDA and Data Cleaning

In the recipe dataset, the data in the nutrition column are lists of numbers. We cannot directly engage with such a data type, so we split the list and create new columns that give meaning to those numbers. The list of numbers stands for calories, total fat, sugar, sodium, protein, saturated fat, and carbohydrates. Therefore, we create seven new columns to store each number.

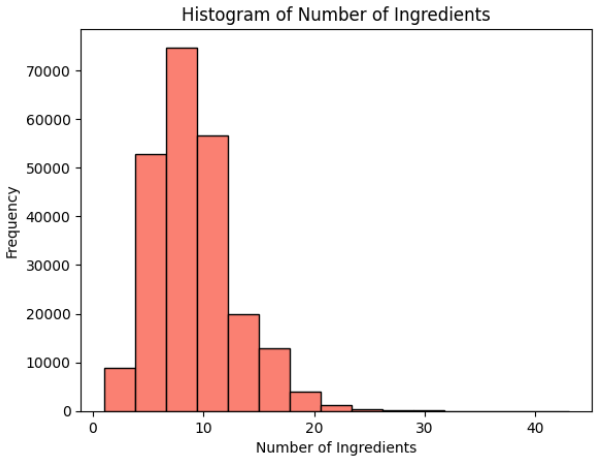
[1] **Generating Personalized Recipes from Historical User Preferences**
Bodhisattwa Prasad Majumder*, Shuyang Li*, Jianmo Ni, Julian McAuley
EMNLP, 2019



The above plot is drawn using the rating column in the interaction dataset. It indicates that most of the users give rating 5, and there is a decreasing trend in the frequency as the rating decreases. However, we notice a relatively large amount of 0 rating, greater than rating 1, 2, and 3. We can conclude from this graph that our interaction dataset is imbalanced since most of the observations have 5-star rating.



The above graph indicates that the majority of the number of steps for a recipe range from 0 to 20, with a center peak at around 10. The graph is right-skewed, which makes sense because most of the recipe that people regularly make and give reviews to are not that complicated.



The above graph shows that most of the number of ingredients for the recipes range from 1 to 20 with a majority centered from 5-12.

We also draw several bar charts to compare different nutrition with rating, such as calories, total fat, and protein. We discover that different nutrition does not have a very large impact on the rating since the height of each bar is relative the same with no observable trend. Therefore, we do not consider nutrition columns when performing feature engineering later.

3 Predictive Task

For this project, our goal is to predict rating based on users' past preferences/ratings on recipes and certain features of the recipes. Since the rating is a discrete number range from 0 to 5, we believe our task is a classification problem; to be more specific, it's multi-class classification. When dealing with classification problems, we are going to implement classification models and collaborative filtering, such as logistic regression, decision tree classifier, Non-negative Matrix Factorization (NMF), etc.

In order to evaluate our model performance, we have three possible value outputs when running the classification_report function from sklearn, which are accuracy, macro average, and weighted average. Accuracy measures the overall correctness of predictions across all classes. However, in imbalanced datasets, accuracy can be misleading since a classifier predicting only the majority class can have high accuracy but might perform poorly on the minority classes. Macro average computes metrics for each class independently and then takes the unweighted mean of the measures. It gives equal weight to each class, regardless of its number of occurrences. Weighted average computes metrics for each class independently but takes the weighted average based on the number of instances for each class. It's useful when you have class imbalance, as it

considers the imbalance by giving more weight to classes with more instances. Therefore, to evaluate our models' performances, we use the weighted average (except for NMF, which is measured in MSE).

The features that we are going to use are n_steps, n_ingredients, popularity_score, description, and minutes. We believe that these five features are correlated with the rating since they contribute to the complexity and popularity of a recipe, which in turn may affect the successfulness of the recipe. We will first need to process the dataset to get the popularity_score column. The popularity_score column classifies the recipes into 5 ranks, with 5 being the most popular recipe and 1 being the least popular recipe. We first get a list of most popular recipes by counting the number of appearance of a recipe in the dataset, then we sort the most popular list and split the list into 5 portions, and label each portion with the popularity score. For the description column, we will calculate word count for each recipe description and use the feature as it is without further processing. In order to process the left four features, we build a pipelines and column transformer to transform the numerical and categorical data separately. We consider minutes and number of steps as numerical data because minutes are continuous, and the number of steps has a very large range, in which it's inappropriate to perform one-hot-encoding. We consider number of ingredients and popularity score as categorical data because they have a small range, and they are discrete numbers. For numerical data that include minutes and number of steps, we standardize them using the StandardScaler function. For categorical data that include number of ingredients and popularity score, we transform them by using the OneHotEncoder function.

4 Models

4.1 Baseline Models

4.1.1 Logistic Regression

Our first attempt is to try to use the logistic regression model from sklearn, since it's a fairly simple model to implement and it could run in an effective rate when encountering large dataset. We can get an overall sense of the dataset using this model. In addition, logistic regression indicates the impact of each feature on the probability of belonging to a specific rating class. This can be valuable for understanding the individual feature importance in predicting ratings. However, logistic regression does not perform so well in an imbalanced dataset since class with less observations will be predicted not as efficiently.

4.1.2 Non-negative Matrix Factorization (NMF)

Inspired by earth's MealPlanner project, we incorporate NMF as one of our models to experience. NMF is a form of collaborative filtering, which is used to make predictions by collecting preferences from many users. It is one of the most effective and popular way to predict rating in recommender system. It also facilitates dimensionality reduction, which will in turn reduce the noise and focusing on essential information. It's also an effective tool since it's good for matrix factorization tasks, particularly with sparse and non-negative matrices, which often represent user-item interactions in recommendation systems. However, the performance of this model seems unsuccessful since the MSE is quite high even with the fine-tune, likely due to the highly imbalanced dataset. In addition, a potential challenge with this model is the cold start problem, since it might struggle when facing new user or recipe.

4.1.3 Minimizing Bias on BetaU, BetaI, and Alpha

We also attempt to implement the iterate function in our recommender system to predict rating. By forming two dictionaries that store rating per user and rating per item, we iterate through these dictionaries to minimize the biases for betaU, betaI, and alpha, then make predictions based on the updated terms. Although it's mostly used for regression tasks, we form the prediction by rounding the resulting number. Although such a function is quite complex and time-consuming, the performance of this model is relatively good as a baseline model. However, such an approach has its limitation, which is that it's hard to add features or further fine-tune the model, since the errors are already minimized.

4.1.4 Decision Tree Classifier

Our final attempt is to use the decision tree classifier. One of the most important reasons that we choose this model as our target model is because that it can model complex, non-linear relationships, which is crucial for our dataset since it's highly imbalanced. In addition, decision trees are robust to outliers or irregularities in the data since they make splits based on information gain, which might reduce the impact of outliers. We believe that this model will fit our dataset more appropriately since it takes the characteristics of our dataset into account, and it could reach good performance with hyperparameter tuning. However, a problem with decision tree classifier is potential overfitting, which we need be mind of.

4.2 Optimization

In order to optimize our models, we use both features and hyperparameter tuning. The three main features that we are going to add are minutes, number of steps, and number of ingredients. We can perform both feature engineering and fine-tuning in all of our dataset except the one that minimizes betaI, betaU, and alpha, since it's very hard to implement additional features and all terms have already updated. Besides adding features, we use GridSearchCV to find the best hyperparameter for logistic regression , decision tree classifier, and NMF. After adding the features and fine-tuned the models, we observe that there is improvement in performance for both logistic regression and decision tree model, but the improvement is not significant for NMF, so we decide to not further investigate in this model.

4.3 Limitation

One of our biggest limitations that we need to consider and be aware of is the nature of our dataset, which is highly imbalanced. Such a characteristic hinders our model selection and affect the way to access the validity of our model. Although we choose our model carefully to address this issue, the class imbalance problem could not be fully solved with such a large dataset. In addition, when fine-tuning our model, we need to be mindful of the overfitting problem, especially for the decision tree classifier, since it could easily overfit as the tree goes deeper.

5 Literature

5.1 User's Food Preference Extraction for Personalized Cooking Recipe Recommendation[1]

In this research paper, authors use a similar food recipe dataset as ours. They try to build a personalized recipe recommender system by analyzing users' food preferences. They focus on recommending the recipe by scoring the recipe based on users' likability toward certain ingredients. The authors employ a classification method to train their model and use precision, recall, and F-measure to evaluate the model performance. The way that the authors use to score the recipe would be a very helpful insight for our project since we can also employ some similar scoring method when evaluating the ingredients column, or using other classification method to predict the rating.

5.2 Generating Personalized Recipes from Historical User Preferences[2]

The dataset that our project use comes from this research article. The authors initially collect dataset of 230K+ recipe texts and 1M+ user interactions (reviews) from Food.com and discard users with fewer than 4 reviews in order to get the dataset that we are using now. In this research paper, authors use the exact same dataset as ours to develop a new task that helps the users with culinary preferences to expand name and incomplete ingredient details in a recipe according to user's historical preferences. The paper compares the personalized model with two baseline models, which are name-based Nearest-Neighbor model and simple Encoder-Decoder with ingredient attention. The main goal of this paper is quite different than what we are trying to predict, so there are limitations to use the approach presented in the research article.

5.3 MealPlanner Recommender System[3]

One of our attempted models is inspired by this recommender system performed on the same food recipe dataset written by earth. In his project, he uses NMF (Non-negative Matrix Factorization) as the baseline model. NMF is a way to reduce dimensionality used for collaborative filtering, which is one of the most popular and effective ways to predict rating given a user-item pair. The result that he gets using this model is very similar to ours since we use the same dataset, but the performance is not as good as expected since the MSE and RMSE are fairly high. We would still need to fine-tune the model by finding the best hyperparameters using GridSearchCV.

5.4 Machine Learning Based Food Recipe Recommendation System[4]

In this article, the authors present two approaches to recommend recipe based on users' past preferences given in the form of ratings. One approach they employed is item based, in which they implement Tanimoto Coefficient Similarity and Log Likelihood Similarity to calculate similarity between recipes. Another approach is user based, in which they use Euclidean Distance and Pearson Correlation. They also introduce fixed size neighborhood and threshold-based neighborhood. Similar to one of our approaches, this paper also implements method of collaborative filtering, but it shows that user-based recommendation is more appropriate, which may suggest that the characteristic of each user is more significant than the characteristics of the recipes. Therefore, when training our model, we could add users' reviews to recipes as one of the features to improve accuracy of the model, since it could better reflect the users' impact on the recommender system.

5.5 Food Recommendation by Sahil Rakholiya[5]

In Sahil's project, he performs exploratory data analysis (EDA) and builds a recommender system based on name or tags, description, and ingredients. Some of his EDA has influenced our analysis when exploring the dataset at an early stage. For example, we cleaned the nutrition column and draw plots to evaluate the relationship between different nutrition and rating. In addition, we have also observed the relationship between n_gredients/n_steps and rating, in which we have found no significant trends. When building the recommender system, Sahil uses TF-IDF to calculate the similarity and recommend recipes based on top similarities in name/tags, description, and ingredients.

6 Results

After fine-tuning and adding features to our proposed models, we are able to compare the performance of models. For the model that minimizes alpha, betaU, and betaI, the weighted average is 0.697, while is fairly good for the baseline model, with no features and no additional fine-tuning.

For logistic regression, the final weighted average on the test dataset is 0.46 as shown below, same as the baseline performance, with a 0.1 increase in accuracy.

Classification Report:

	precision	recall	f1-score	support
0.0	0.00	0.00	0.00	687
1.0	0.00	0.00	0.00	249
2.0	0.00	0.00	0.00	311
3.0	0.00	0.00	0.00	815
4.0	0.00	0.00	0.00	2807
5.0	0.61	1.00	0.76	7586
accuracy			0.61	12455
macro avg	0.10	0.17	0.13	12455
weighted avg	0.37	0.61	0.46	12455

For decision tree classifier, the final weighted average on the test dataset improves from 0.65 (baseline) to 0.85 as shown below.

```
]: print(classification_report(y_pred, y_test))
```

	precision	recall	f1-score	support
0.0	0.00	0.30	0.01	61
1.0	0.00	0.00	0.00	2
2.0	0.00	0.00	0.00	13
3.0	0.00	0.21	0.00	33
4.0	0.01	0.42	0.02	602
5.0	1.00	0.74	0.85	174015
accuracy			0.74	174726
macro avg	0.17	0.28	0.15	174726
weighted avg	0.99	0.74	0.85	174726

From the above reports, we can see that the models demonstrated varied performance in predicting ratings, with the decision tree classifier showing the most promising results. However, challenges with imbalanced data persisted across models, impacting their ability to generalize across all rating classes. The result is still significant in showing how the decision tree classifier is suitable for this kind of large dataset and multi-class prediction.

After evaluating the features, we come to a conclusion that adding the popularity_score, n_steps, n_ingredients, and minutes improve the model performance, especially the popularity_score feature. However, the description feature does not improve the model performance. When we train the model without this description feature, the weighted average does not change, but the macro average increases.

The decision tree classifier's parameter that we fine-tuned include max_depth, min_samples_split, and criterion. Max depth controls the maximum depth of the decision tree, so that we didn't set the upper threshold too high since we want to avoid overfitting the model. The min_samples_split defines the minimum number of samples required to split an internal node. It also helps control overfitting by setting a threshold for the number of samples required for a node to split. Criterion defines the function to measure the quality of a split. The two commonly used criteria are 'gini' for the Gini impurity and 'entropy' for information gain. These metrics help decide how the tree should split at each node, aiming for homogeneity within the nodes.

Despite its simplicity, the logistic regression model struggled to handle the imbalanced dataset, resulting in suboptimal performance on minority classes. NMF, often effective in recommendation systems, exhibited high Mean Squared Error (MSE) even after fine-tuning. The iterative approach to minimize biases showed relatively good performance, demonstrating improved prediction accuracy. However, its complexity might limit further feature addition or fine-tuning. The decision tree classifier proved robust in handling non-linear relationships within the dataset. This model showcased promising performance with a significant improvement in weighted average score.

7 References

[1]https://dl.acm.org/doi/10.5555/2887675.2887686#sec-ref

[2] https://aclanthology.org/D19-1613.pdf

[3] https://www.kaggle.com/code/earthearth/mealplanner

[4]https://www.researchgate.net/profile/Manju-N/publication/366962956_Machine_Learning_Based_Food_Recipe_Recommendation_System/links/63bb9a9e03aad5368e7665e6/Machine-Learning-Based-Food-Recipe-Recommendation-System.pdf

[5]https://www.kaggle.com/code/sahilr05/food-recommendation