CSE258 Assignment 2 Report- Using Stacking to Enhance Model Performances on Rating Predictions for Food Recipe

Haotian Xu, Yi Jiang
University of California, San Diego

Dataset

The dataset selected is based on *Food.com Recipes and Interactions*[1]. The datasets are composed of two sections. The first section of the data is the interaction data, which is the users' ratings on food recipes. It contains the columns user_id, recipe_id, date, rating, and review. The other portion is the recipe data, which contains information about each recipe, including recipe name, recipe_id, minutes, contributor_id, submitted time, tags for category, nutrition, n_steps, the steps to cook the food, description, ingredients, and n_ingredients. Here are two example data points from Interaction Data and Recipe Data.

```
Interaction Data
{'user_id': 241882,
 'recipe_id': 116976,
 'date': '2010-04-12',
 'rating': 5,
 'review': 'Made these this weekend.
They were wonderful.....'}
```

```
Recipe Data
{'name': 'all in the kitchen  chili',
 'id': 112140,
 'minutes': 130,
 'contributor_id': 196586,
 'submitted': '2005-02-25',
 'tags': "['time-to-make',...]",
 'nutrition': '[269.8, 22.0, 32.0,
48.0, 39.0, 27.0, 5.0]',
 'n_steps': 6,
 'steps': "['brown ground beef in large
pot',...]",
 'description': "this…",
 'ingredients': "['ground beef',...]",
 'n_ingredients': 13}[1]
```

We have conducted an exploratory analysis of the dataset. There are 1132367 interaction data entries and 231637 recipe data entries, so the data is large enough for the purpose of this assignment. Among the interaction dataset, there are 226570 users and 231637 recipes. The global average rating of all user-to-recipe interactions is 4.41. The median rating of all user-to-recipe interaction is 5. We have also observed some trends of the dataset that help us to come up with the model.

Firstly, most of the ratings made by users are 5, which occur approximately 800,000 times. Then comes the second most

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

popular ratings around 4, for approximately 200,000 times. The rest of the ratings range from 0 to 3. Figure 1 vividly represents the rating distribution. Consequently, people tend to give relatively high ratings in the review of food recipes.

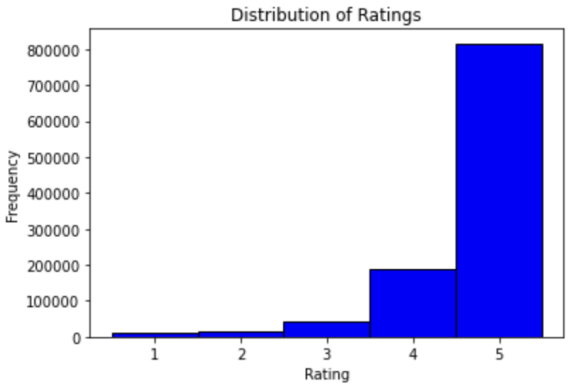

Figure 1: Distribution of Ratings

Additionally, we observed that users are prone to give higher ratings from 2002 to 2011 in comparison with other years. Users rate the recipes lower and lower from 2012 to 2016. There is a slight turnover starting from 2017. Thus, users gave higher ratings in earlier years. This provides the insights for us to use temporal models for the rating predictions. Figure 2 the average rating by year is presented below.

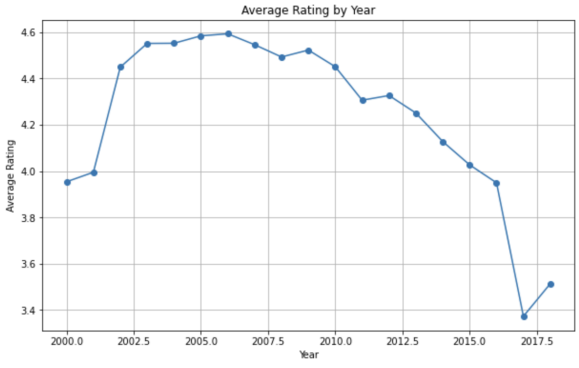

Figure 2: Average Rating by Year

We found that the peak time of the reviews are from 2007 to 2009, as revealed by Figure 3. There are from 140,000 to 160,000 interactions from 2007 to 2009. The years in the tail portion of the graph only have around 20,000 interactions.

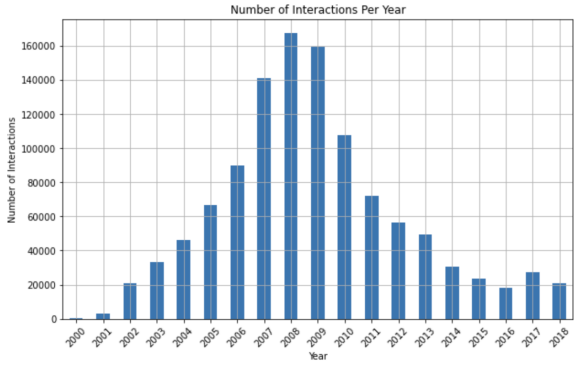

Figure 3: Number of Interactions Per Year

Predictive Task

The predictive task on this dataset is to predict the rating a user would give to a recipe. We will evaluate the model in the following approach. The dataset is splitted into a training set of 70% of the data for 800,000 entries, a validation set of 18% of the data, for 200,000 entries and a test set of the rest. We train the data using the training set, tune the parameters and come up with a meta model with the validation set. Finally, the model is evaluated based on the test set's MSE(Mean Square Error).

MSE is the Mean Square Error computed with the predicted ratings and the actual ratings as given below.

$$\frac{1}{N} ||y - X\theta||_2^2$$

Model's validity is assessed by comparing its MSE with the MSE of the baseline. The baseline predicts the global average rating for every pair of user and recipe in the test set. The model is defined

to be successful if its Mean Square Error is less than the baseline's Mean Square Error.

The features for the base model incorporated in the training process are as follows. For the users, the features are all of the recipes each user has rated and the average rating the user gives. For the recipe, the features are all of the users who review it, and the average rating each recipe has received. The interactive features are the ratings that users have given to the recipes. The latent factor models account for the latent features.

The data is read from CSVs into Pandas dataframes. Then it is converted to an array of objects, in which each object is either a recipe information or an interaction data point.

```
df =
pd.read_csv('./archive/RAW_interactions
.csv')
interactionData = []
for index, row in df.iterrows():

interactionData.append({'user_id':row['
user_id'],'recipe_id':row['recipe_id'],
'date':row['date'],'rating':row['rating
'],'review':row['review']})
```

Dictionaries like usersPerItem and itemsPerUser are used to store users' reviewed recipes and recipes' users who have rated this specific recipe. These data structures are used later on to calculate the similarity between two recipes. The user to recipe rating is stored in the python dictionary meanwhile.

```
ratingsPerUser = defaultdict(list)
ratingsPerItem = defaultdict(list)
usersPerItem = defaultdict(set)
itemsPerUser = defaultdict(set)
ratingsDict = {}
reviewsPerUser = defaultdict(list)
for d in dataTrain:
    r = d['rating']
    u = d['user_id']
    i = d['recipe_id']
    ratingsPerUser[u].append(r)
    ratingsPerItem[i].append(r)
    usersPerItem[i].add(u)
    itemsPerUser[u].add(i)
    ratingsDict[(u,i)] = r
    reviewsPerUser[u].append(d)
```

Average ratings per user and per recipe are calculated with python dictionaries and array aggregations.

```
#calculate rating averages per
user/item
userAverages = {}
itemAverages = {}

for u in itemsPerUser:
    rs = [ratingsDict[(u,i)] for i in
itemsPerUser[u]]
    userAverages[u] = sum(rs) / len(rs)

for i in usersPerItem:
    rs = [ratingsDict[(u,i)] for u in
usersPerItem[i]]
    itemAverages[i] = sum(rs) / len(rs)
```

Furthermore, we adopted the stacking ensemble learning algorithm so there are features for the meta model. The features selected are predicted ratings with Jaccard collaborative filtering, Cosine collaborative filtering, Pearson collaborative filtering, and Surprise latent factor model. To illustrate the process to gather the feature in a clear way, the simplified code section is below.

```
def predictRating(user,item):
    ratings = []
    jaccard = []
    cosine = []
```

```
    pearson = []
    ratings = []
    for d in reviewsPerUser[user]:
        i2 = d['recipe_id']
        if i2 == item: continue
        ratings.append(d['rating'] -
itemAverages[i2])
        similarity = feature(item,i2)
        jaccard.append(similarity[0])
        cosine.append(similarity[1])
        pearson.append(similarity[2])
    predictRatings = [1]
    #jacard
    if (sum(jaccard) > 0):
        weightedRatings = [(x*y) for x,y
in zip(ratings,jaccard)]

predictRatings.append(itemAverages[item
] + sum(weightedRatings) /
sum(jaccard))
    elif item in itemAverages:
predictRatings.append(itemAverages[item
])
    else:
predictRatings.append(globalRating)
    #cosine
    if (sum(cosine) > 0):
        weightedRatings = [(x*y) for x,y
in zip(ratings,cosine)]

predictRatings.append(itemAverages[item
] + sum(weightedRatings) / sum(cosine))
    elif item in itemAverages:
predictRatings.append(itemAverages[item
])
    else:
predictRatings.append(globalRating)
    #pearson
    if (sum(pearson) > 0):
        weightedRatings = [(x*y) for x,y
in zip(ratings,pearson)]

predictRatings.append(itemAverages[item
] + sum(weightedRatings) /
sum(pearson))
    elif item in itemAverages:
predictRatings.append(itemAverages[item
])
    else:
predictRatings.append(globalRating)
    prediction2 =
algo2.predict(user,item)

predictRatings.append(prediction2.est)
    return predictRatings
```

Model

The model selected is an ensemble model of latent factor model and several collaborative filtering models using stacking. Stacking is detailed in *A Stacking Ensemble Model of Various Machine Learning Models for Daily Runoff Forecasting*. It has a “two layer structure” with the base model and the meta model. The meta model will combine the outputs from the base models to make the final predictions.[2]

In this assignment, linear regression is the meta model that combines several base models - latent factor model by Surprise Library, collaborative filtering model with Jaccard Similarity, collaborative filtering model with Cosine Similarity, and collaborative filtering model with Pearson Similarity. The equation of linear regression is

[2] A Stacking Ensemble Model of Various Machine Learning Models for Daily Runoff Forecasting, *Water* 15, no. 7: 1265. Lu, Mingshen, Qinyao Hou, Shujing Qin, Lihao Zhou, Dong Hua, Xiaoxia Wang, and Lei Cheng. 2023.

$$f = \theta_0 + \theta_1 r_{Latent} + \theta_2 r_{Jaccard} + \theta_3 r_{Cosing} + \theta_4 r_{Pearson}$$

where r is the rating prediction from base models and θ is the learnable weight.

The training set trains the base models like the collaborative filtering models. Then, each model predicts the output for the hold-out validation set. These outputs are gathered as features for each data point in the validation set. In the next step, these features are fed into the linear regression model to produce the meta model. Finally, the meta model predicts the outputs on the test set. Figure 3 below illustrates the process accurately.

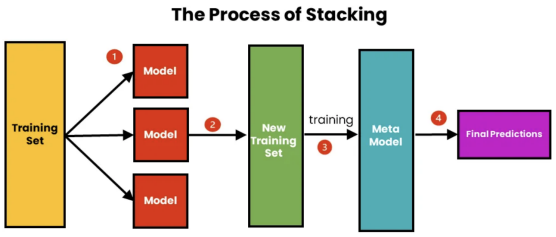

Figure 4[3]

This meta model is selected because ensemble learning can effectively reduce the generalization error of a single model. Since the prediction is numeric in value, linear regression is more suitable as the meta model in comparison to linear regression. Latent factor models and collaborative filterings are adopted since they are effective models covered in courses to predict ratings. To optimize it in the future, we could ensemble more models like BPR with more powerful machines. For the current models, the training, validating, and test time already took over 40 minutes. Additionally, we could refactor the code to enhance the performance. Moreover, we should incorporate temporal heuristics into the model. As we can observe from the trends, time is a large factor for rating predictions.

Some issues encountered are that the training set is large, so it takes 30 minutes to generate the features for the meta model. In this case, it takes too much time to compare different ensemble models and tune the parameters. Also, it is time consuming to predict ratings for a new test set. Thus, it might not scale well. Since we use validation sets to tune the parameters, it does not overfit that much. Though more folds of cross-validation can be performed in the future to mitigate overfitting better.

We have considered multiple models. Firstly, we tried a simple baseline model using the global average rating. We continued the average rating models by grouping the average rating by time and giving predictions only based on the time. We tried 90-day and 1-year time frames and got similar results. This model is the simplest to implement and takes the least time to train. However, we still thought the performance can be improved. Then, we tried the single collaborative filtering model with different similarity heuristics. We attempted to incorporate temporal heuristic into the collaborative filtering model, which took too long to complete, like over an hour. Temporal collaborative filtering models should reduce the MSE, yet it is time consuming to train and validate. Additionally, we used the latent factor models. We applied the latent factor models using SVD, SVDpp, and BaselineOnly from the surprise package, which has sample codes from the textbook. We compared

[3] Stacking to Improve Model Performance: A Comprehensive Guide on Ensemble Learning in Python. Medium. Soni, Brijesh. May 1, 2023.

models and parameters using grid search and cross validation and concluded that the BaselineOnly performed the best and the best parameters are {'bsl_options': {'method': 'als', 'n_epochs': 30, 'reg_u': 5, 'reg_i': 13}}. We selected this model as one of the base models for our meta model. The BaselineOnly method is the simplest latent model without latent factors (Gamma)[4]. The equation is

$$f(u,i) = \alpha + \beta_u + \beta_i$$

We attempted some advanced models as well, such as the temporal latent factor model but did not succeed. We built the temporal latent factor model based on TensorFlow with the starting code from the textbook. However, our machine did not support the memory requirement for such a large dataset, which would result in high dimensions in parameters.

Literature

Our dataset is sourced from the influential work presented in *Generating Personalized Recipes from Historical User Preferences*[1], a pioneering study in the field of Natural Language Processing (NLP) and personalized recommendation systems. This dataset, curated from Food.com, encompasses an extensive collection of over 180,000 recipes and 700,000 user interactions, providing a rich foundation for our research. The original study's innovative approach, combining an encoder-decoder framework with attention mechanisms and user preference modeling, offers a novel perspective on recipe generation. However, our work focuses on predicting ratings for recipes given by users, which does not involve personal recommendation and text mining.

Many other research works[5] [6] on similar datasets of food recipes with a similar focus on recommendations to the last paper. For example, the study *Recipe Recommendation With Hierarchical Graph Attention Network*[5] introduces an advanced hierarchical graph attention network (HGAT) for recipe recommendation, leveraging relational data among users, recipes, and food items to optimize recommendations. This contrasts with traditional methods by using neural network modules for a more nuanced analysis. Again, since our focus is on a different area, we do not have a conflict with their conclusions.

There are also studies[7] [8] that focus on recipe rating, similar to our work. The study *An Intelligent Approach for Food Recipe Rating Prediction Using Machine Learning*[7] explores some common models such as decision trees, SVM, and KNN in predicting recipe ratings. They also applied the ensemble model by combining several

[4] "Basic Algorithms." Basic algorithms - Surprise 1 documentation. Accessed December 3, 2023. https://surprise.readthedocs.io/en/stable/basic_algorithms.html.

[5] Tian, Yijun, Chuxu Zhang, Ronald Metoyer, and Nitesh V. Chawla. "Recipe recommendation with hierarchical graph attention network." Frontiers in big Data 4 (2022): 778417.

[6] Forbes, Peter, and Mu Zhu. "Content-boosted matrix factorization for recommender systems: experiments with recipe recommendation." In Proceedings of the fifth ACM conference on Recommender systems, pp. 261-264. 2011.

[7] Khan, Ismam Hussain, Md Habib Ullah Khan, and Md Mamun Howlader. "An Intelligent Approach for Food Recipe Rating Prediction Using Machine Learning." In 2021 1st International Conference on Artificial Intelligence and Data Analytics (CAIDA), pp. 281-283. IEEE, 2021.

[8] Harvey, Morgan, Bernd Ludwig, and David Elsweiler. "You are what you eat: Learning user tastes for rating prediction." In String Processing and Information Retrieval: 20th International Symposium, SPIRE 2013, Jerusalem, Israel, October 7-9, 2013, Proceedings 20, pp. 153-164. Springer International Publishing, 2013.

models. However, their models use features such as the number of ingredients and number of instructions, while our models primarily rely on user-item interactions and only need the user and the recipe to predict. Another major difference between them and us is that they treat recipe ratings prediction as a classification problem, while we take it as a regression problem. Both ideas are reasonable since ratings have a limited range and if we round our prediction outputs, our model can also become a classifier.

Results

We list our results from various models below. We use MSE as our metrics. The first table shows models' performance on the validation set and the second one shows that on the test set.

Model Name	Performance (MSE)
Jaccard Collaborative Filtering	1.873
Cosine Collaborative Filtering	1.871
Pearson Collaborative Filtering	1.791
Global average baseline	1.600
Average rating by year	1.536
Average rating by 90-day	1.533
SVDpp (default parameters)	1.506
SVD (default parameters)	1.492
BaselineOnly (default parameters)	1.475
BaselineOnly (optimized parameters)	1.467
Meta Model	1.458

Table 1: Models' Performance on Validation Set

Model Name	Performance (MSE)
BaselineOnly (optimized parameters)	1.470
Meta Model	1.460

Table 2: Models' Performance on Test Set

From these tables, we can find that our meta model performs the best among all other models. From the performance of average rating by time models and the trend in Figure 2, we can find that the time feature can play a role in predicting ratings. However, the performance of the simple latent model BaselineOnly also shows that we can reach good results without the time features, as time will increase model complexity and computation requirements a lot. Our final meta model only takes user-item interactions as input and does not require other features. We think it is an advantage of our model because the real-world data may not provide a complete feature list for those feature-based models and the missing features may influence their performance a lot.

In the future, we may improve our model by improving the latent factor model to the temporal latent factor model, which may require us to upgrade our machine or use cloud computing.

Citations

Lu, Mingshen, Qinyao Hou, Shujing Qin, Lihao Zhou, Dong Hua, Xiaoxia Wang, and Lei Cheng. 2023. "A Stacking Ensemble Model of Various Machine Learning Models for Daily Runoff Forecasting" *Water* 15, no. 7: 1265. https://doi.org/10.3390/w15071265

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

Soni, Brijesh. “Stacking to Improve Model Performance: A Comprehensive Guide on Ensemble Learning in Python.” Medium, May 1, 2023. https://medium.com/@brijesh_soni/stacking-to-improve-model-performance-a-comprehensive-guide-on-ensemble-learning-in-python-9ed53c93ce28#:~:text=Stacking%20is%20a%20strong%20ensemble,stacked%20ensembles%20or%20stacked%20generalization.

“Basic Algorithms.” Basic algorithms - Surprise 1 documentation. Accessed December 3, 2023. https://surprise.readthedocs.io/en/stable/basic_algorithms.html.

Tian, Yijun, Chuxu Zhang, Ronald Metoyer, and Nitesh V. Chawla. "Recipe recommendation with hierarchical graph attention network." Frontiers in big Data 4 (2022): 778417.

Forbes, Peter, and Mu Zhu. "Content-boosted matrix factorization for recommender systems: experiments with recipe recommendation." In Proceedings of the fifth ACM conference on Recommender systems, pp. 261-264. 2011.

Khan, Ismam Hussain, Md Habib Ullah Khan, and Md Mamun Howlader. "An Intelligent Approach for Food Recipe Rating Prediction Using Machine Learning." In 2021 1st International Conference on Artificial Intelligence and Data Analytics (CAIDA), pp. 281-283. IEEE, 2021.

Harvey, Morgan, Bernd Ludwig, and David Elsweiler. "You are what you eat: Learning user tastes for rating prediction." In String Processing and Information Retrieval: 20th International Symposium, SPIRE 2013, Jerusalem, Israel, October 7-9, 2013, Proceedings 20, pp. 153-164. Springer International Publishing, 2013.