Building a Hybrid Model for Food Recommendations

1st Adarsh Narasimha Murthy
dept. of Software Engineering
San Jose State University
San Jose, USA
adarsh.narasimhamurthy@sjsu.edu

2nd Alireza Mahin Parvar
dept. of Software Engineering
San Jose State University
San Jose, USA
alireza.mahinparvar@sjsu.edu

3rd Aomkar Mathakar
dept. of Software Engineering
San Jose State University
San Jose, USA
aomkar.mathakar@sjsu.edu

I. Introduction

Online Recommendation systems have become central to improving user experience, assisting users to make choices quickly and wisely. These systems have historically been concentrated on e-commerce and entertainment. Food recommendations have received relatively less attention compared to these fields. This is surprising, considering that humans spend a disproportionate amount of time thinking about food and eating it. Food has gone beyond just being a basic human need for sustenance, towards a symbol of culture, art and evolution. As humans have become more aware of the impact of food on health, they have developed regimens in the form of various diets for healthier living. This makes food recommendations challenging as users have developed complex, constrained food habits such as vegan or paleo diets and preferences based on ingredients or flavor. Another challenge is that recipes/ ingredients can have multiple names and different methods of preparation. Therefore, we need to develop a food recommendation system considering all these user quirks and challenges.

In this project, we will focus on two approaches to recommendation systems: content-based and collaborative filtering-based methods. Content-based methods rely on information about the user/items to recommend similar items to the user. J. Freyne and S. Berkovsky [1] made food recommendations based on ingredients that the user had liked in the past. For example, if bell peppers were present in a majority of the recipes that the user liked, the recommendation engine would recommend more recipes containing bell peppers. Although content-based models work well with new users/items solving the cold start problem, it suffers from a lack of diversity i.e., they can only make recommendations based on existing user interests.

Collaborative filtering methods find similar users/items and provide recommendations based on this. The method tries to find the underlying relationship between different users/items and uses this to make predictions. Collaborative filtering can be further classified into user-based, item-based, and matrix Factorization. User-based methods rely on the likes of similar users whereas item-based methods find the items similar to the ones user has liked in the past to make recommendations. Matrix Factorization decomposes the user and items into smaller dimensions and makes predictions based on it. Collaborative filtering methods offer relatively good diversity and accuracy but are unable to recommend items for new users (and in some cases, new items). To overcome the shortcomings of both models, in this project, we explore different combinations of content-based and collaborative models to develop a hybrid food recommendation system.

II. System Design and Implementation

A. System Design

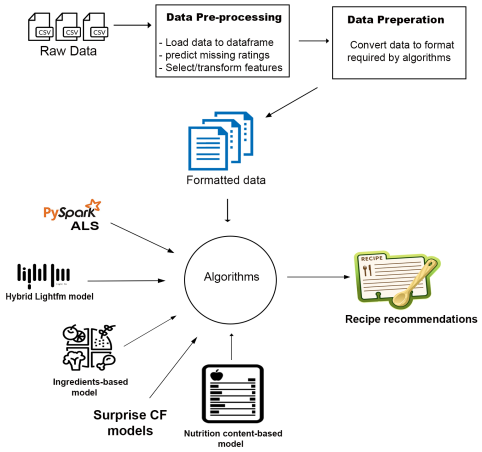

Fig. 1. An overview of the food recommendation process

B. Algorithms

a) Spark ALS: This is a Matrix Factorization collaborative filtering model provided by Apache Spark. It uses the Alternating Least squares (ALS) algorithm [2] to learn latent factors. ALS converts high-dimensional user-item matrices into low-dimensional user and item matrices. In the case of new users or items with no rating history (the cold-start problem), Spark assigns NaN predictions to such items.

b) Nutrition-based Item model: the nutrition content of recipes can provide valuable insight into the food habits of the user. For example, if the user is following a keto diet, he

will prefer only high-fat, low-carb food items. In this model, to determine the predicted rating for recipe X, we find the cosine of recipe X to all other recipes that are rated by the user. We then find the top-N recipes that are at the nearest distance to recipe X and find the weighted average of ratings of these recipes to compute the predicted rating for recipe X.

c) Tag/Ingredient Item model: We construct an item-based model based on ingredients in combination. In this model, we build an ingredient vector for each recipe and compute the cosine similarity of the recipe to all other recipes. To predict ratings for an unknown recipe, we use the cosine distance of recipe X to top-N recipes rated by the user. A similar strategy is followed for tags.

d) Surprise Collaborative Filtering models: We have experimented with user-based, item-based, and Matrix factorization collaborative filtering models. We have used the Python Surprise library to accomplish this.

e) LightFM hybrid model: LightFM provides a hybrid matrix factorization model that represents users and items as linear combinations of their latent factors. However, in the case of new users/items, it considers features similar to content-based models to make recommendations to the user. The model claims to outperform both collaborative and content-based models in cold-start scenarios and perform at least as well as a pure matrix factorization model where interaction data is abundant. [3]

C. Technologies and Tools

a) Programming Language: Python: The easy syntax of python and its rich data science-related libraries makes python an ideal choice for our project. Python 3.6 was used for our development. Numpy, pandas, scipy, sklearn, surprise, networkx are a few of the python libraries used in our project.

b) Notebook: Jupyter Notebook: Jupyter is a free, open-source, web-based interactive development environment for notebooks, code, and data. Its flexible interface allows us to configure and arrange workflows, making it easier for the user to understand the code. Jupyter Notebook is compatible with common programming languages such as Python, Ruby, and R. It provides easy access to the outcome of the code and also assists users to tweak and edit the code before execution.

c) Distributed Computing Framework: Apache Spark: Apache Spark is a large-scale data processing and analytics framework that can distribute data processing across multiple nodes to crunch through large amounts of data. Spark supports Java, Scala, Python, and R programming languages. We have used pyspark, which is an interface for Spark in Python. With the right distribution of nodes, Spark can train ML models 100x faster compared to normal execution.

The Databricks platform was used to run the code. Databricks community edition is a free version of their cloud-based big data platform, provides access to free 15GB clusters, and a jupyter-like notebook environment to run the code.

d) High-Performance Computing (HPC): San Jose State University provides a high-performance computing system with multi-core and multi-socket servers, high storage, and GPUs tied together by an ultra-fast inter-connection network. We used HPC with 50GB of RAM memory with four threads and four nodes.

e) PuTTY: PuTTY is an open source SSH and telnet client developed for the Windows platform. Putty was used to connect to SJSU HPC cluster.

III. Experiments

A. Dataset

The Recipe dataset has multiple files: recipe, users, and interactions. The recipe file contains information about the recipe ingredients, nutrition, and steps. The interactions file has the reviews and ratings of users for recipes. The user file has user details. The statistics of these files is shown in "Fig. 2"

Dataset	Rows	Features	Size
Users	25076	1	13 MB
Recipes	231637	12	280 MB
Interactions	711,356	6	333 MB

Fig. 2. Data statistics

B. Data Preprocessing

As shown in "Fig. 3", the ratings range from 0-5. An examination of ratings with the value '0' revealed that these were reviews with missing ratings. This was significant since there were 17644 reviews with 0 ratings. Therefore, we decide to impute these missing ratings using Sentiment Analysis.

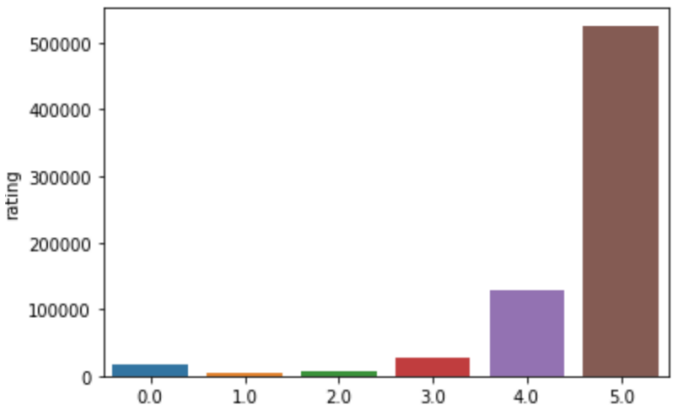

Fig. 3. Ratings range

The training set for Sentiment analysis was the reviews with ratings other than 0, and the test set was reviews with 0 ratings. Before the review data could be fed to ML algorithms, we had to perform data-cleaning operations on the review text. As shown in "Fig. 4", stop words, punctuation, HTML tags, words less than two characters, digits were removed, and Lemmatization was performed to format the data.

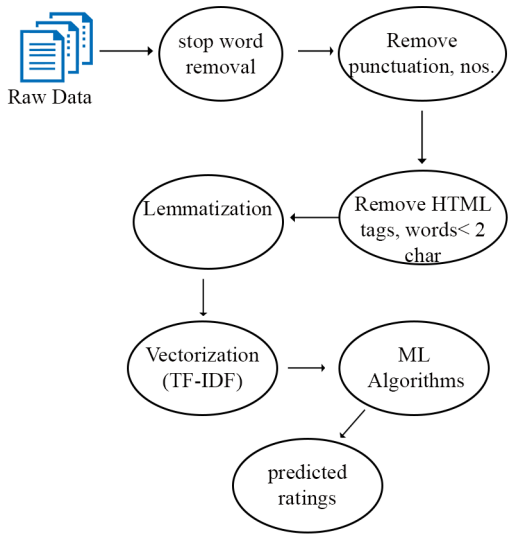

Fig. 4. An overview of the Sentiment Analysis process

This cleaned data was vectorized using TF-IDF (term frequency-inverse document frequency) and fed to various machine learning algorithms provided by sklearn such as Logistic Regression, Multinomial Naive Bayes, Random Forest, and k-means. The best accuracy of 75.7% was achieved using Multinomial Naive Bayes."Fig. 5" provides a comparison of different algorithms.

	Name	Accuracy	Precision
0	Random Forest	75.56	32.37
1	KNN	74.93	42.02
2	Logistic Regression	73.38	47.68
3	Naive bayes	76.69	75.06

Fig. 5. An evaluation of different ML algorithms for review text

C. Algorithms implementations

a) Spark ALS: We first need to load the data into Spark RDD for processing. After loading the data into RDD, we split the data into training and holdout sets to determine the best parameters for ALS algorithm using Spark itertools. It was found that rank= 20, numIterations = 10 and alpha=0.03 were the best parameters that yielded an RMSE of 4.73.

b) Nutrition-based item model: This model considers only the nutrition values of the recipes to make recommendations. Each recipe has nutrition labels in the format [51.5, 0.0, 13.0, 0.0, 2.0, 0.0, 4.0]. The first step in the process is separating this list of values into separate columns of calories, total fat, sugar, sodium, protein, saturated fat, and carbohydrates. We next normalize the data using sklearn.preprocessing. To predict the rating for a recipe by a user, we determine the cosine distance of the recipe to other recipes in the normalized matrix and compute the predicted rating as the weighted average of the top-N nearest recipes. The best RMSE achieved through this method was 4.2.

c) Tag/ingredient-based item model: For this experiment, the following steps were followed to arrive at predicted ratings

- Determine the ingredients for each recipe and create a separate list for each ingredient
- Create sparse matrices for each recipe using the BERT Sentence transformer
- Find the cosine similarity between all sparse matrices of ingredients using scipy
- For a given recipe X, find the top-similar recipes based on ingredients
- The predicted rating for recipe X will be the same as the top-similar recipe. The RMSE of this method was 8.9.

d) Surprise collaborative filtering algorithms: Surprise is an easy-to-use python library that provides standard implementations of many of the popular recommendation algorithms. We used Surprise's user-based, item-based, and matrix factorization algorithms to make predictions. We first selected a sample set of 30k reviews to determine the best algorithm among the three, and also arrive at the best parameters using GridSearchCV. A cv=5 was used to perform the experiments, and the same steps were repeated for reviews containing '0' ratings and without '0' ratings. "Fig. 6" provides a comparison of the algorithms. From the experiment, it is clear the data preprocessing step of removing 0 ratings has led to significant improvements in RMSE, and matrix factorization performs marginally better compared to its counterparts.

Based on the experiments on the sample dataset, we selected the best hyperparameters for matrix factorization and applied the algorithm to the complete dataset resulting in an RMSE of 0.88.

e) LightFM hybrid algorithms: The following steps were followed to implement LightFM:

- Each recipe had comma-separated nutrition labels. There were separated into columns of calories, total fat, sugar, sodium, protein, saturated fat, and carbohydrates.
- One-hot encoding was performed on the tags present in each recipe.
- The nutrition labels, tags, n steps, minutes, and n ingredients were parsed to generate data in the format [('Item1', ['f1:1', 'f2:1', 'f3:0'])] for each recipe.
- The formatted item data was passed to buildItemFeatures to generate item embeddings.
- The unique set of item and recipe ID's are extracted from the interactions file and passed to buildInteractions to generate user-item and weights sparse matrices.
- The user-item matrix, item embeddings, and sample weights are all passed to the lightFM model for training.
- LightFM generates implicit ratings and does not provide exact ratings on a scale of (1,5). Therefore, we cannot

Rating Scale	Algorithm	Best Parameters	RMSE
0-5	User based CF	{'k': 5, 'sim_options': {'name': 'pearson', 'min_support': 1, 'user_based': True}}	0.97
	Item based CF	{'k': 3, 'sim_options': {'name': 'pearson', 'min_support': 1, 'user_based': False}}	0.97
	Matrix Factorization	{'n_factors': 50, 'reg_all': 0.1, 'n_epochs': 20}	0.96
1-5	User based CF	{'k': 5, 'sim_options': {'name': 'pearson', 'min_support': 1, 'user_based': True}}	0.64
	Item based CF	{'k': 3, 'sim_options': {'name': 'pearson', 'min_support': 1, 'user_based': False}}	0.64
	Matrix Factorization	{'n_factors': 50, 'reg_all': 0.1, 'n_epochs': 20}	0.62

Fig. 6. A comparison of Surprise algorithms on a sample set of 30k records

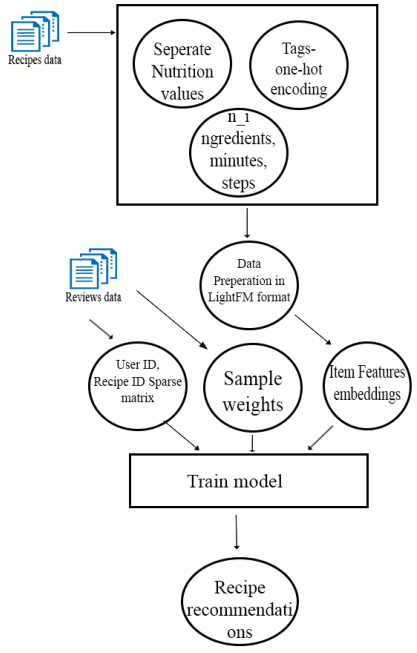

Fig. 7. An overview of LightFM process

calculate RMSE in this case. But LightFM provides a utility to calculate AUC and it was found to be 0.838878 for the complete dataset.

To provide recipe recommendations to a user, we will pass in all the available recipe IDs and the user id as input to the LightFM model. LightFM ranks the recipes on its own local scale and returns the scores. We will filter out the recipes that the user has already rated from these, sort them according to their ranks, and return the top-N ranked recipes as recommendations.

IV. Discussion and Conclusion

A. Things that worked

- Pre-processing the data by imputing '0' ratings improved RMSE.
- the Hybrid recommendation model using LightMF and the Collaborative filtering model are the types of algorithm models that worked and executed in good form.
- Surprise matrix factorization algorithms and Spark ALS took the least training time, executing in less then 5 mins.

B. Things that did not work well

Algorithm	Training Time
Matrix Factorization	5 mins
Spark ALS	5 mins
Nutrition-based model	>200 hrs
Ingredients/tags model	>200 hrs
LightFM	2 hrs

Fig. 8. Training time for complete dataset on HPC

- Issues with the large size of data
- Nutrition-based and Tags/Ingredient-based models could not be scaled for the complete dataset. For computing the cosine similarities between each ingredient list/tags list required memory was more than 400G and the estimated time for training was about 200 hrs as shown in "Fig. 8"
- Spark ALS did not yield good RMSE

C. conclusion

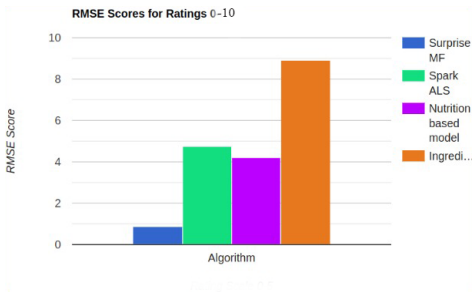

Fig. 9. A comparison RMSE of all algorithms

We have tried various methods of implementing recommendations for the recipe dataset. From our analysis, it was

found that the LightFM hybrid model that combines matrix factorization with item features such as nutrition, tags, and steps yields the best results. It has an AUC of 0.84. AUC has a range from 0-1, and AUC above 0.8 is considered to be a good recommendation model. The Surprise implementation of Matrix factorization gave the next best results with an RMSE of 0.88. It also took less training time compared to LightFM.

V. Task Distribution

A. Adarsh Narasimha Murthy

- Review Sentiment Analysis using data mining techniques
- Nutrition Based Item model
- LightFM Hybrid model
- Spark ALS implementation on Databricks community platform

B. Alireza Mahin Parvar

- Surprise Collaborative Filtering methods: User-based, Item-based, and Matrix Factorization
- Ingredient/Tags based Item model
- Visualization of recipes based on tags in the form of networks using networkx
- Figuring out to connect Jupyter notebook to HPC using tunneling.

C. Aomkar Mathakar

- Deep learning model for Review sentiment analysis
- Documentation generation in the form of reports and slides
- Helped in running Surprise Collaborative Filtering methods.

VI. Appendix

A. Github

https://github.com/Adarsh3thy/RecipeRecommender

B. Link to Dataset

https://www.kaggle.com/datasets/shuyangli94/food-com-recipes-and-user-interactions

References

[1] J. Freyne, and S. Berkovsky, "Intelligent food planning: personalized recipe recommendation," Proc. of the 15th international conference on Intelligent user interfaces, pp. 321-324, Feb 2010.

[2] Y. Koren, R. Bell, and C. Volinsky, "Matrix factorization techniques for Recommender Systems," Computer, IEEE, vol. 42, pp 30–37

[3] M . Kula, "Metadata Embeddings for User and Item Cold-start Recommendations," in CBRecSys, Vienna, Austria, Sep., 2015

[4] V. Sher, "How I would explain building LightFM Hybrid Recommenders to a 5-year old," Online: https://towardsdatascience.com/how-i-would-explain-building-lightfm-hybrid-recommenders-to-a-5-year-old-b6ee18571309