Final Year Project

Health-Aware Food Recommender System

Sagar Mahajan

Student ID: 19204052

A thesis submitted in part fulfilment of the degree of

BSc. (Hons.) in Computer Science

Supervisor: Professor David Coyle

UCD School of Computer Science

University College Dublin

May 21, 2023

Table of Contents

Abstract

This report presents the implementation and evaluation of recommender system algorithms for a recipe recommendation application. The objective is to develop a reliable recommender system that suggests recipes to users based on their preferences and dietary requirements. The further goal is to modify the recommender system to recommend healthy recipes. The report uses the FoodRecSys-V1 dataset from Kaggle which includes ratings given by users to recipes along with the nutritional breakdown of the recipes. We construct three rating prediction algorithms; user-based, content-based and hybrid based. The algorithms are tuned and tested using the Root Mean Squared Error. The content-based algorithm turns out to be the best performing among the three. A health weightage is added to the content-based algorithm using the FSA score of the recipes. The resulting algorithm can be tuned by the user based on how healthy they want their food recommendations to be.

Chapter 1: Project Specification

These days, people are aspiring to have a healthier lifestyle. However, the majority of us struggle to take the first step towards it. According to WHO, worldwide obesity has nearly tripled since 1975. A great first step towards a healthy lifestyle is to keep track of ones eating habits. With the rapid development of mobile technology and the abundance of data, doing this is easier than ever using our phones. There are many apps and websites in the market which recommend to users the healthy dishes they would prefer. Many of them also keep track of their calorie and nutrient intake. But how do they work? How do they predict and recommend recipes according to users' preferences? The answer is Recommender systems.

In this project, I will be using a food recipe dataset to implement 3 types of recommender system algorithms. The systems will recommend food recipes according to the user's preferences. Post that, their accuracy will be tested and the most accurate one will be updated by adding the health weightage to the algorithm. The health weightage will make sure the recipes being recommended have a good nutritional balance.

The Core goals of my project are:

- Implementing 3 recommender systems and testing their accuracy.
- Taking the most accurate recommender system and adding health weightage to it.

The advanced goal of my project is:

- Run a user survey to see how the recommender system performs.

For my project, I will be looking for a detailed Dataset which includes food recipes, their nutrition values and how different people have rated that recipe. The data will then be divided into training and test sets in order to implement the selected recommender system algorithms.

Chapter 2: **Introduction**

In today's ever-changing food industry and fast-paced lifestyle, individuals are increasingly confronted with the challenge of making improved and healthier food choices . A common daily task involves meal preparation and cooking. When deciding what to cook, people often seek recipes that cater to their personal preferences. However, with the vast array of food options available, manually browsing through a complete recipe catalog can be time-consuming and discouraging. Hence, the demand for decision-support systems that recommend personalized food choices while considering user preferences and eating history has emerged[2].

Recommender systems are tools that filter information and narrow that information down based on the content of that information or based on a user's preferences or needs. In order to help a person grasp the topic of interest among the sea of information, recommender systems frequently take the opinions of user communities into account . A recommender system is basically a piece of software created to interact with huge, complicated information spaces and present the user with information or products that are relevant to them[18].

Food is one of the big and upcoming application of recommender systems. Based on bookmarks and ratings given to recipes by users, such recommenders retrieve recipes that contain, for example, the same ingredients as recipes liked previously[19]. However, majority of food recommendation systems make no mention of health or fitness. Up until now, the main emphasis has been on figuring out and forecasting what users will appreciate in terms of meals, which isn't always a guarantee of wholesome nourishment. In fact, in many instances, it will have the reverse effect; individuals who enjoy fatty or calorie-dense foods will be recommended foods that have these characteristics[3].

This report will target this issue by finding the best working food recommender system for the dataset FoodRecSys-V1 on Kaggle and modifying it by incorporate the health aspect in it. We will use the Food Standards Agency (FSA) score of the recipes to measure the healthiness of the recipes and change the recommendation according to it.

Chapter 3: Related Work and Ideas

I conducted thorough research in order to find more information about the field of food recommender systems. In this section, I will discuss some existing papers and the state of the art.

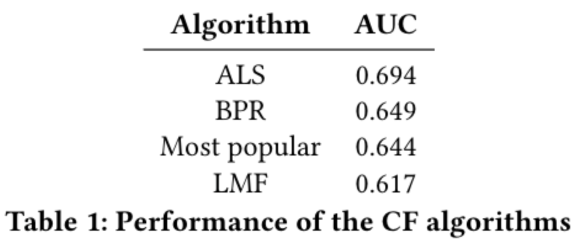
3.1 Recommender System Technologies

According to Burke et al. (2011) and Burke (2000), a recommender system can be defined as follows: "Any system that guides a user in a personalised way to interesting or useful objects in a large space of possible options or that produces such objects as output". It is used in a variety of fields these days including streaming websites like Netflix, e-commerce websites like Amazon and job search engines like LinkedIn. There are many different recommender system technologies. 2 of the main ones we will be focusing on are collaborative filtering and Content-based filtering.

Collaborative filtering

The underlying assumption of the collaborative filtering approach is that if a person A has the same opinion as a person B on an issue, A is more likely to have B's opinion on a different issue than that of a randomly chosen person. Therefore it looks at similar user's data and recommends according to that. This was the main reason I chose CF as one of the filtering systems. It does not only depend on ingredients or cuisines that they have already tried. It will help users explore recipes of different styles that they have never experienced.

There are many existing CF algorithms made for different fields. In a study [1] which focuses on making a healthy and personalised recommender system, They picked 3 popular CF algorithms and 1 most popular recommender system and decided to test their performance. The algorithms they chose were Alternating Least Squares (ALS), Bayesian Personalized Ranking (BPR) and Logistic Matrix Factorization (LMF). To conduct the test they decided to divide their dataset into training, cross-validation and test sets. They evaluated the test using the AUC score. Below are the final scores of the algorithms as shown in the paper.



Algorithm	AUC
ALS	0.694
BPR	0.649
Most popular	0.644
LMF	0.617

Table 1: Performance of the CF algorithms

Figure 3.1: performance of the CF algorithms

According to their test, ASL had the best AUC score. This score tells us that 69.4% of the algorithm's predictions when tested on their particular dataset were correct. This can help me further in the project to choose the best collaborative algorithm.

Rather than just using the user rating of the recipe, I can also use the tags. Tags can be, For instance, a user may use the tag 'spicy' to tell the system she prefers that type of recipe. Similarly, an item can be described with the tag 'tomato' if this is one of its ingredients. In paper [2] They explore the matrix factorisation and how user tags can be incorporated into it. In matrix factorization, ratings are estimated by computing the dot product of the vectors shown in figure 3.2:

$$\hat{r}_{um} = \mathbf{p}_u^T \mathbf{q}_m$$

Figure 3.2: matrix factorization dot product

they adopted the rating prediction model and introduced additional feature vectors for users' and recipe tags, respectively. They changed the algorithm to the equation shown in figure 3.3:

$$\hat{r}_{um} = \left(\mathbf{p}_u + \frac{1}{|T_u|}\sum_{t \in T_u} \mathbf{x}_t\right)^T \left(\mathbf{q}_m + \frac{1}{|T_m|}\sum_{s \in T_m} \mathbf{y}_s\right)$$

Figure 3.3: updated matrix factorization equation

In paper [4] They took a food recipe dataset and applied 5 different recommender system algorithms to it. One of the algorithms they used is N neighbours. N nearest neighbours is amongst the earliest algorithms developed for collaborative filtering. They are identified using Pearson's correlation algorithm shown in first equation in figure 3.4 and predictions for recipes not rated by a user are generated in the second equation of figure 3.4 .

$$sim(u_a, u_b) = \frac{\sum_{i=1}^{k}(u_{a_i} - \overline{u_a})(u_{b_i} - \overline{u_b})}{\sum_{i=1}^{k}(u_{a_i} - \overline{u_a})^2 \sum_{i=1}^{k}(u_{b_i} - \overline{u_b})^2}$$
$$pred(u_a, r_t) = \frac{\sum_{n \epsilon N} sim(u_a, u_n) rat(u_n, r_t)}{\sum_{n \epsilon N} sim(u_a, u_n)}$$

Figure 3.4: N Nearest neighbour Sim and pred equation

One of the biggest disadvantages of collaborative filtering models is the cold start problems. The prediction of the model for a given (user, item) pair is the dot product of the corresponding embeddings[9]. So, if an item is not seen during training, the system can't create an embedding for it and can't query the model with this item. To overcome this problem, people use content-based recommender systems.

Content-based filtering

Content-Based filtering uses item features to recommend other items similar to what the user likes, based on their previous actions or explicit feedback [7]. In our case, Rather than focusing on the user's ratings of the recipes, We can also predict what a user would prefer using the ingredients they tend to like in their choice of recipes. One of the reasons I chose this filtering system was that it does not depend on other user's data. Therefore there are higher chances of the recommended recipe being something that the user will like and consider.

In paper [4] They used a similar technique to implement a content-based algorithm. They separated the recipes provided in their dataset into the ingredients used to make them and then divided the rating given to the recipe equally amongst the ingredients. After that they a content-based algorithm shown in Equation 4 to predict a score for the target recipe rt based on the average of all the scores provided by user ua on ingredients ingr1, ..., ingrj making up rt.

$$pred(u_a, r_t) = \frac{\sum_{j \epsilon r_t} score(u_a, ingr_j)}{j}$$

Figure 3.5: content-based prediction equation

Yang et al in paper [10] explores the unconventional side of content-based filtering. According to him, the current interfaces for food or restaurant preference elicitation rely extensively on text-based descriptions and rating methods, which can impose high cognitive load. Instead, he looked at the food image and built a model based on Convolutional Neural Networks (CNN) and called it PlateClick. It uses similarity distance between food images to decide which food items a user will prefer. A similar algorithm can be used in my project if my dataset has food images.

One major advantage of content-based filtering is that it does not suffer from the cold start problem. It can recommend recipes even if they are not present in the training dataset. It does not require other people's data when recommending one user as it only uses their feature preferences. However, one drawback of content-based filtering is that it is not very in-depth. In paper [4] They realised that a content-based algorithm only looks at the ingredients when recommending recipes and leaves the features like 'cooking duration', and 'complexity' whereas collaborative filtering considers all of this as it is based on the user's rating. These disadvantages we have encountered in both of the filtering techniques can be overcome by the use of a hybrid filtering algorithm.

Hybrid Filtering

A hybrid algorithm of collaborative filtering and content-based filtering can be highly effective as it can benefit from the advantages of both and negate each other's disadvantages. For example in our case when we decide to use a collaborative filtering algorithm we only look at the rating of the users and no other features. However, we can modify and add weightage for other features like the ingredients or cooking duration which might increase the overall accuracy of the model.

A similar thing has been done in a paper [11] by Harvey Morgan. He explores different biases that can affect the user's choice of food and use them to compose a hybrid recommendation model. The biases he explored are- user item bias, and nutritional info ratio. An example of bias can be that some users naturally rate items higher than others and some may naturally choose from a lower baseline score. He thought By calculating these biases as part of the model, he could effectively remove eccentricities from the ratings of recipes. He used a Singular Value Decomposition or SVD-based similarity measure and removed the biases from it. SVD is a matrix factorisation technique (a part of collaborative filtering), which reduces the number of features of a dataset by reducing the space dimension from N-dimension to K-dimension (where K<N). This model performed the best compared to 7 other models.

In paper [4] they made 2 hybrid strategies using the Collaborative filtering and content-based filtering discussed earlier. They both divide the recipes rated by the user into ingredients. Then rather than using recipe ratings in the equations shown in figure4 find the neighbours, they used the individual ingredient scores. The denser data obtained is then used in content-based prediction formulae shown in figure 5 to predict the ratings.

3.2 Testing techniques

One of the most recurring testing techniques for recommender systems I ended up noticing was the MAE score. MAE or Mean Absolute Error is measured as the average of the absolute error values. The Absolute is a mathematical function that makes a number positive. Therefore, the difference between an expected value and a predicted value can be positive or negative and will necessarily be positive when calculating the MAE [5]. A lower MAE means higher prediction accuracy. In paper [2] They used MAE and RMAE to test the accuracy of their offline recommender system. RMSE is the Root of MAE. figure 3.6 shows the formulas used in the paper:

$$MAE = \frac{1}{|T|} \sum_{(u,i) \in T} |\hat{r}_{ui} - r_{ui}|$$

$$RMSE = \sqrt{\frac{1}{|T|} \sum_{(u,i) \in T} (\hat{r}_{ui} - r_{ui})^2}$$

Figure 3.6: MAE and RMSE equation

where T is a test set of user and item pairs. ^rui is the model-predicted rating and rui is the true user ratings, which they obtained using a user study. To conduct the test they used the 5-fold cross-validation method. This validation procedure divides the full rating data set into five disjoint subsets of the same size. Then, the system prediction model is trained using four of these five subsets and the predictions for the ratings in the fifth subset are compared with the true ratings found in that subset (test set). The training and testing process is repeated five times with a different test set. And finally, the system error measured over five iterations is averaged to produce a final measure. This way of dividing the rating data and testing the models can be very useful for me when I'm testing the accuracy of my models.

Table in figure 3.7 shows the MAE of the rating prediction models matrix factorisation and content-based model. As discussed earlier in the collaborative filtering section, their matrix factorisation algorithm (MF-T) includes the user tags. The mean absolute error of the algorithm created by them is much lower than the content-based model (CBFB) and the standard matrix factorisation model (Standard MF) which does not include the tag weightage.

Table 1: MAE of the considered rating prediction algorithm

MAE	MF-T	CBFB	Standard MF
Fold 1	0.702	1.559	1.122
Fold 2	0.678	1.503	1.200
Fold 3	0.733	1.393	1.052
Fold 4	0.546	1.464	0.854
Fold 5	0.772	1.025	0.860
Mean	0.686	1.389	1.018

Figure 3.7: MAE of considered rating prediction algorithm

In paper [4] they evaluated the 5 recommender algorithms constructed in the paper using MAE. They used a bar graph instead of a table to show the difference in the mean absolute error. A bar graph makes it easier to understand the outcome. Fig2 shows the bar graph. According to this graph, the intelligent hybrid ingr is the best performing model with the lowest MAE.

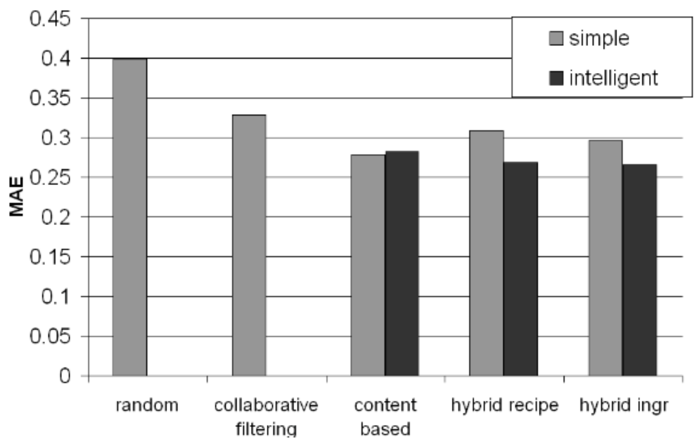


Figure 3.8: MAE values

There are multiple advantages to using MAE as a testing technique. First, the mechanics of the computation are simple and easy to understand. Second, mean absolute error has well-studied statistical properties that provide for testing the significance of a difference between the mean absolute errors of two systems[6]. However Mean absolute error may be less appropriate for tasks such as Find Good Items where a ranked result is returned to the user, who then only views items at the top of the ranking. For these tasks, users may only care about errors in items that are ranked high, or that should be ranked high. It may be unimportant how accurate predictions are for items that the system correctly knows the user will have no interest in. This can be the case with our recommender system as our main goal is to recommend food recipes according to user preference and the ratings given by them and therefore the main considerations of the user will be those recipes that have a high rating.

Another frequently used test for recommender systems is the Mean Average Precision or MAP. This test is a build-up on the traditional classification metric - Precision [8]. Precision by itself takes into account one recommended item, however, mean average precision averages the precision of the list of items or in our case recipes shown to the user. One can pick the number of top recommended instances MAP will consider. This number is usually denoted by 'K'. In paper [3], they used 5-fold cross-validation as the protocol for all the experiments and report the recommendation performance results employing MAP@5. thus they focus on a ranking task aiming to predict the 5 recipes users would rate highest. I can implement this test and adjust the 'K' so that it suits my recommender systems and gives out the most accurate answer.

3.3 Calculating health score

Post picking the most accurate recommender system, my plan is to add a health weightage to the algorithm. There are many existing health tests which look at the nutrient levels of the food ingredients and calculate whether a food is healthy or not. One of the most popular sources of food and its nutritional quality is Food Standard Agency (FSA). FSA is responsible for food safety and food hygiene in England, Wales and Northern Ireland. FSA uses 4 macro-nutrients (sugar, sodium, fat and saturated fat) to calculate the score. They show their scoring using a traffic light system as shown in figure 3.9.

Another very famous health test is provided by the World Health Organisation (WHO). They defined 15 ranges of macro-nutrients which should be considered in a daily meal plan. In the paper [3] one of their main research questions is "Can we improve standard recommender algorithms in

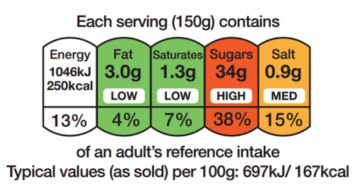

Each serving (150g) contains
Energy
1046kJ
250kcal
13%
Fat
3.0g
LOW
4%
Saturates
1.3g
LOW
7%
Sugars
34g
HIGH
38%
Salt
0.9g
MED
15%
of an adult's reference intake
Typical values (as sold) per 100g: 697kJ/ 167kcal

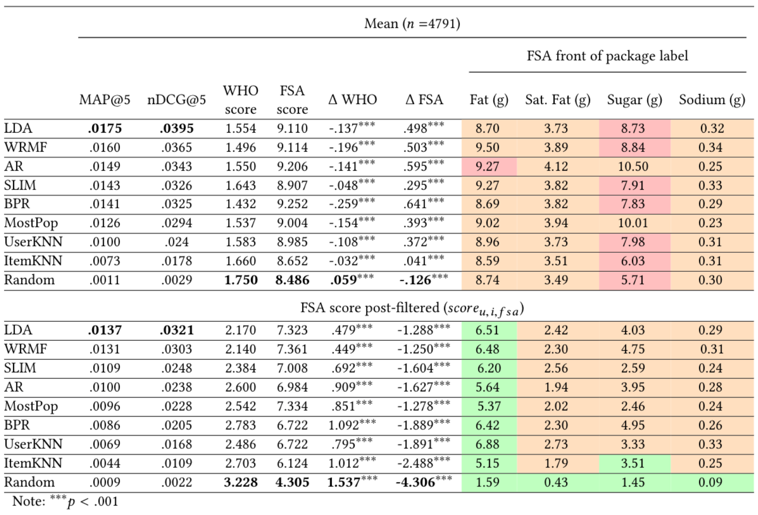
Figure 3.9: FSA traffic light system

terms of making the recommendations they offer more healthy? They did this by post-filtering and re-weighting the multiple recommender algorithms they decided to work with. they apply a simple linear scoring function which re-weights the scores of a recipe for a particular user based on the WHO or inverse FSA score of the recipe shown in figure 3.10 below

$$score_{u,i,who} = score_{u,i} \cdot (who_i + 1)$$
$$score_{u,i,fsa} = score_{u,i} \cdot (16 - fsa_i - 4 + 1)$$

Figure 3.10: FSA and WHO Score

The addition of the weight increased the FSA score and the WHO score of the algorithms however, it drastically decreased their accuracy as well. Figure 3.11 shows the mean FSA and WHO score of the top 5 recommended recipes from their dataset and how it changes after the filtering. It is clearly visible by the colour of the nutrients that the addition of the weight increased the healthiness of the recipes being recommended by the recommender systems.



	Mean (n =4791)									
							FSA front of package label			
	MAP@5	nDCG@5	WHO score	FSA score	Δ WHO	Δ FSA	Fat (g)	Sat. Fat (g)	Sugar (g)	Sodium (g)
LDA	**.0175**	**.0395**	1.554	9.110	-.137***	.498***	8.70	3.73	8.73	0.32
WRMF	.0160	.0365	1.496	9.114	-.196***	.503***	9.50	3.89	8.84	0.34
AR	.0149	.0343	1.550	9.206	-.141***	.595***	9.27	4.12	10.50	0.25
SLIM	.0143	.0326	1.643	8.907	-.048***	.295***	9.27	3.82	7.91	0.33
BPR	.0141	.0325	1.432	9.252	-.259***	.641***	8.69	3.82	7.83	0.29
MostPop	.0126	.0294	1.537	9.004	-.154***	.393***	9.02	3.94	10.01	0.23
UserKNN	.0100	.024	1.583	8.985	-.108***	.372***	8.96	3.73	7.98	0.31
ItemKNN	.0073	.0178	1.660	8.652	-.032***	.041***	8.59	3.51	6.03	0.31
Random	.0011	.0029	**1.750**	**8.486**	**.059*****	**-.126*****	8.74	3.49	5.71	0.30
	FSA score post-filtered ($score_{u,i,fsa}$)									
LDA	**.0137**	**.0321**	2.170	7.323	.479***	-1.288***	6.51	2.42	4.03	0.29
WRMF	.0131	.0303	2.140	7.361	.449***	-1.250***	6.48	2.30	4.75	0.31
SLIM	.0109	.0248	2.384	7.008	.692***	-1.604***	6.20	2.56	2.59	0.24
AR	.0100	.0238	2.600	6.984	.909***	-1.627***	5.64	1.94	3.95	0.28
MostPop	.0096	.0228	2.542	7.334	.851***	-1.278***	5.37	2.02	2.46	0.24
BPR	.0086	.0205	2.783	6.722	1.092***	-1.889***	6.42	2.30	4.95	0.26
UserKNN	.0069	.0168	2.486	6.722	.795***	-1.891***	6.88	2.73	3.33	0.33
ItemKNN	.0044	.0109	2.703	6.124	1.012***	-2.488***	5.15	1.79	3.51	0.25
Random	.0009	.0022	**3.228**	**4.305**	**1.537*****	**-4.306*****	1.59	0.43	1.45	0.09

Note: $^{***}p < .001$

Figure 3.11: recommender system performance pre and post health score

Chapter 4: **Project Workplan**

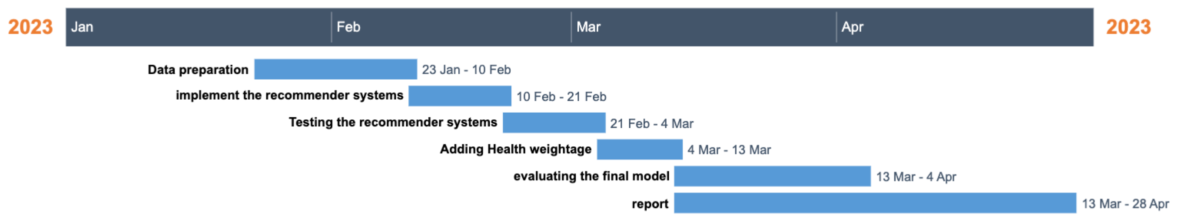


Figure 4.1: Workplan Gantt Chart

I will be working on the Jupiter notebook for the majority of the semester. The data preparation stage and the evaluation stage will take the most amount of time as shown in the Gantt chart. In the last 2 weeks, I will focus completely on the report and finalise it. Here is a breakdown of the targets I plan to achieve in each of the phases shown in the Gantt chart.

Data preparation

- clean the data so there are no missing values, outliers or any other noise that can cause a hindrance in the accuracy of the recommender systems.
- Refine and filter out the requires nutritional data.

Implement the recommender systems

- Decide which 3 algorithms I will be going forward with.
- Implement the 3 types of algorithms and train them.

Testing the Recommender systems

- Use the MAE and MAP@K tests to find out which algorithm performs the best for the dataset.

Adding health weightage

- Use the FSA and WHO nutritional score to create a weightage and re-write the

Evaluating the final model

- conduct an online user study to find out how effective the recommender system.

Although I will leave the last 2 weeks completely for the report, I will start the work on my report during the study week which starts on the 13th of March and keep on updating the report post that. Along with writing the report I will conduct the user study and evaluate my final model as I will have more free time.

Chapter 5: **Implementation**

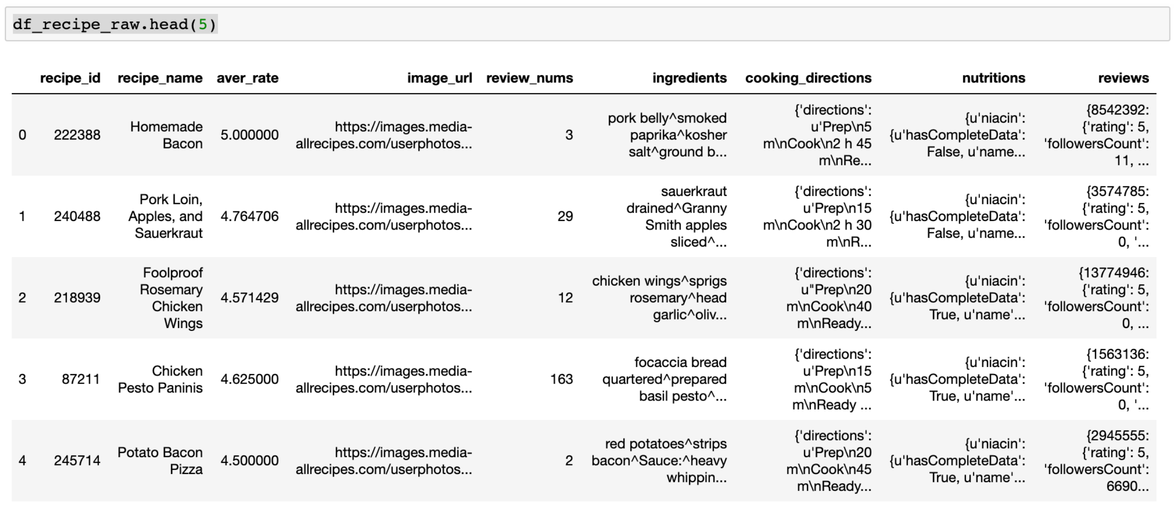
In this part of the report, I will provide a detailed discussion of how I used the information gathered from the literature review to implement my project. The main language I decided to conduct my project on was Python.

5.1 Data Collection

The initial step to set the environment for my project was to find a dataset to work on. The main criteria I wanted my dataset to satisfy were:

- Has enough recipes and users to build well-functioning recommender systems and is divisible into test and training sets.
- Must have some kind of ratings given to the recipes by the users for the recommender system algorithms to function.
- Should have a breakdown of ingredients for the recipes to successfully implement a content-based recommender system.
- Must have a detailed nutritional breakdown for all the recipes to form a health weightage for the algorithms.

Considering these criteria I looked for the dataset and found one on Kaggle called foodRecSys-V1. The file includes a total of 49698 recipes, 1,160,267 users and 3,794,003 interactions. The file also includes a training set and a test set to execute the recommender system algorithms. Figure 5.1 shows the first 5 entries of the raw recipe dataset.



```
df_recipe_raw.head(5)
```

	recipe_id	recipe_name	aver_rate	image_url	review_nums	ingredients	cooking_directions	nutritions	reviews
0	222388	Homemade Bacon	5.000000	https://images.media-allrecipes.com/userphotos...	3	pork belly^smoked paprika^kosher salt^ground b...	{'directions': u'Prep\n5 m\nCook\n2 h 45 m\nRe...	{u'niacin': {u'hasCompleteData': False, u'name...	{8542392: {'rating': 5, 'followersCount': 11, ...
1	240488	Pork Loin, Apples, and Sauerkraut	4.764706	https://images.media-allrecipes.com/userphotos...	29	sauerkraut drained^Granny Smith apples sliced^...	{'directions': u'Prep\n15 m\nCook\n2 h 30 m\nR...	{u'niacin': {u'hasCompleteData': False, u'name...	{3574785: {'rating': 5, 'followersCount': 0, '...
2	218939	Foolproof Rosemary Chicken Wings	4.571429	https://images.media-allrecipes.com/userphotos...	12	chicken wings^sprigs rosemary^head garlic^oliv...	{'directions': u"Prep\n20 m\nCook\n40 m\nReady...	{u'niacin': {u'hasCompleteData': True, u'name'...	{13774946: {'rating': 5, 'followersCount': 0, ...
3	87211	Chicken Pesto Paninis	4.625000	https://images.media-allrecipes.com/userphotos...	163	focaccia bread quartered^prepared basil pesto^...	{'directions': u'Prep\n15 m\nCook\n5 m\nReady ...	{u'niacin': {u'hasCompleteData': True, u'name'...	{1563136: {'rating': 5, 'followersCount': 0, '...
4	245714	Potato Bacon Pizza	4.500000	https://images.media-allrecipes.com/userphotos...	2	red potatoes^strips bacon^Sauce:^heavy whippin...	{'directions': u'Prep\n20 m\nCook\n45 m\nReady...	{u'niacin': {u'hasCompleteData': True, u'name'...	{2945555: {'rating': 5, 'followersCount': 6690...

Figure 5.1: Raw FoodRecSys dataset

The user ratings range from 0 to 5. The users have also given a written review of the recipes they rated. The dataset also includes a URL to the food's image. All this information will be beneficial to analyse the data and create the most accurate recommender system.

5.2 Data preparation

As I continued with my implementation of the algorithms, I realised that although the total recipes and the users were adequate amount, the number of interactions in the pre-divided datasets were too big. Even though the algorithms were functioning, the calculation process was taking a lot of time. My computer's RAM could not support the amount of data being provided and looped through. Therefore I decided to not use all of the data.

I decided to cut short the data by only using 30000 interactions from the training dataset provided. Then I divided the selected interactions into training and testing by the ratio of 80:20.

5.2.1 Nutritional Dataset

Nutrition data is in the form of dictionaries. There are 20 different nutrients given in the dataset for each recipe. Each nutrient dictionary includes a boolean data called HasCompleteData, its name, the amount, percentage daily value, display value, and the units. Even though we are not provided with any description about the nutrient variables, I assumed that the variable 'amount' tells us the amount of that nutrient present in the recipe per 100 grams. By looking at the values, I could infer that the 'displayValue' holds the rounded-off value of the 'amount'.

As I decided to use FSA score to add the health weightage, I decided to extract only the nutrients I needed to calculate it from the data. The required nutrients were calories, fat, saturated fat, sugar, and sodium. These can be seen in the traffic light system talked about in the literature review. However I faced a huge problem when my dataset did not include the amount of salts in the recipes. Therefore it was challenging for me to calculate an accurate FSA score. I saw multiple papers, one particular example being paper[17], were they replaced the salt nutrient with sodium. I therefore decided to replace it for my health weightage and the calculation of the FSA score. I extracted just the 'dailyValue' variable for each nutrient for it to be used in the formulae. The values were turned into integers and all the rows with the values not available were dropped.

5.2.2 Ingredients

I decided to use ingredients as the feature in the content-based algorithm. The ingredients were given as a string in the dataset separated with the character "ˆ. A lot of pre-processing needed to be done on the ingredient for it to be accurate. Below is an example of an ingredients.

['green bell peppers','salt to taste','ground beef','chopped onion','salt and pepper to taste', 'peeled tomatoes chopped', 'Worcestershire sauce', 'uncooked rice', 'water', 'shredded Cheddar cheese', 'condensed tomato soup', 'water as needed']

The initial pre-processing steps I implemented were turning the words into lowercase, and removing numbers and any other non-alphabetic characters. Next, I removed any stop-word for the lists. Post this I had to deal with words like “chopped”, “to taste” and “shredded” which are not ingredients but rather adjectives and adverbs describing the ingredients. Looking at a paper [12], I decided to tackle this using the Parts of Speech library in NLTK.

In NLTK (Natural Language Toolkit), which is a popular Python package for natural language processing (NLP), part-of-speech (POS) tagging is the process of automatically assigning the appropriate parts of speech to words in a given text or sentence. The NLTK package provides different algorithms and tools for POS tagging, such as the default POS tagger and the Punkt tokenizer [13]. The tags we are interested in nouns as we only need the ingredient names in our dataset. The code I used to implement this is given below.

word = [w.lower() for w, tag in tagged_word if tag.startswith('N')]

The “NN” tag means that the word is a noun. Therefore the code only keeps those words whose tag starts with the letter “N”. Below is the resultant ingredients list of the example given above.

['bell peppers', 'salt','ground beef','onion','salt pepper','tomatoes','worcestershire sauce','rice','water','cheddar cheese','tomato soup','water']

I also performed lemmatisation on the ingredients. Lemmatization is the process of reducing a word to its base or dictionary form, known as the “lemma". An example of this given below where the words tomatoes, carrots and potatoes to its non-plural form.

pre : ['pork chops', 'carrots', 'celery', 'tomatoes', 'onion', 'kidney beans', 'potatoes', 'beef bouillon cube', 'salt']

post : ['pork chops', 'carrot', 'celery', 'tomato', 'onion', 'kidney beans', 'potato', 'beef bouillon cube', 'salt']

5.3 Algorithms

Prior to my implementation, I had thought of making recommender system algorithms that recommend top k recipes to the target user. I later scrapped this idea as there were very limited ways of testing the results from them. Instead, I decided to make rating prediction recommender systems that predict the rating for a target recipe for a specific user. This is much easier to evaluate as the actual and the predicted rating can be assessed.

I picked 3 algorithms to work on, user-based, content-based and a hybrid of both. I created a separate Python class for each of the algorithms. All of these algorithms include a function called predict which calculates the predicted rating for a specific recipe by a specific user. The code for the predictor will be different for each algorithm. The accuracy of the predicted rating will be calculated using the RMSE which is the Reduced Mean Squared Error. The RMSE results will also help us perform tuning on the algorithms.

5.3.1 User-Based Algorithm

The user based algorithm I decided to go forward with was k nearest neighbours. K nearest neighbours. The k-nearest neighbours algorithm, or KNN for short, is a supervised learning classifier used in machine learning to make predictions or classifications based on proximity[14]. The way we implement it in our algorithm is by trying to find the k-nearest neighbours of the user based on the user-rating similarity and then predicting the rating by taking the weighted average of them.

The first step is to create a matrix where the rows correspond to recipes and the columns correspond to users, with the matrix elements being the rating that each user gave to each recipe. This matrix is used to calculate the similarity between users, using the nearest neighbour algorithm from sklearn.neighbours package, which returns the k nearest neighbours to a given user based on a chosen similarity metric. Then it takes a user ID and a recipe ID as input and returns the predicted rating that the user would give to the recipe, based on the ratings of the k-nearest neighbours to that user who have also rated the recipe.

I tuned the algorithm by testing different similarity metrics for the algorithm. The chosen metrics for testing were cosine, euclidean and Manhattan metrics. I decided to go forward with these metrics as they are the most commonly used among the research papers I have gone through. Below is an explanation for them.

Cosine Similarity: Measures the cosine of the angle between the two vectors, which represents how similar they are in the direction. The range of cosine similarity is between -1 and 1, where a value of 1 represents two identical vectors, 0 indicates that the vectors are orthogonal or dissimilar, and -1 indicates that the vectors are diametrically opposed. [15]

Euclidean distance: Measures the distance between two points in n-dimensional space. It is calculated as the square root of the sum of the squared differences between the corresponding elements of two vectors.[15]

Manhattan: Measures the distance by summing the absolute differences of the corresponding elements of the two vectors.[15]. The formula to calculate the Manhattan distance is as follows:

$$d = |x1 - x2| + |y1 - y2| + ... + |n1 - n2|$$

where d is the distance between the two points and (x1, y1, ..., n1) and (x2, y2, ..., n2) are the coordinates of the two points in n-dimensional space.

Choosing a low value of K leads to unstable decision boundaries, and a very high value of K may result in poor classification accuracy. Therefore, it is important to choose a reasonable range of K values to achieve a balance between underfitting and overfitting. I decided to test the algorithm for values 5, 10 and 15.

Figure 5.2 shows a heat map showing the results for cross-validation used to perform tuning and deciding what number of k and which metric works the best for our dataset. We can clearly see that k = 5 with cosine similarity has the lowest RMSE. It's surprising that as the number of neibours is increased, the RMSE value for all the similarity metrics also increases. One of the possible reasons for this can be due to overfitting. As we increase the number of nearest neighbors, the model becomes more complex and captures more intricate relationships between users and items. However, the increase in complexity may cause the model to overfit the training data, leading to poor performance on new, unseen data. This is because the model may start to rely too heavily on the local patterns in the training data, which may not generalize well to new data.

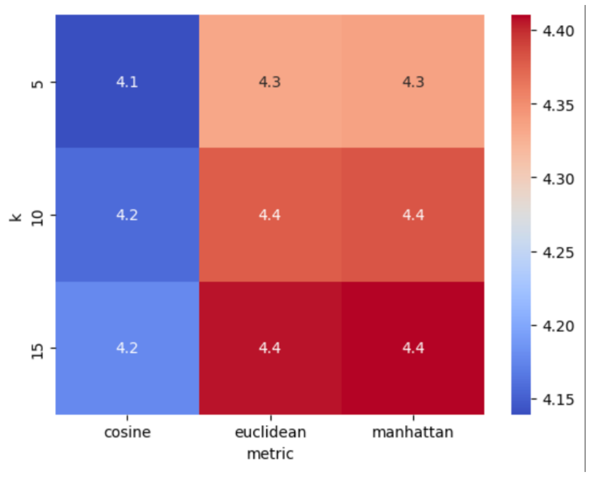


Figure 5.2: Heatmap showing tuning result of K an similarity metric on rmse of User-based algorithm

The number of nearest neighbours and the similarity metric our algorithm works best for is 5 and cosine respectively and had the RMSE value of 4.1. These parameters will therefore be used when comparing the algorithms on the basis of the test data results.

One of the reasons cosine similarity works the best for this algorithm compared to the rest could be due to how it handles sparsity. Sparsity in our case means that a lot of users have not rated many recipes and there are not enough users who have rated the same set of recipes. This leads to missing values as the rating would be zero. Cosine similarity handles sparsity well because it focuses on the non-zero elements (rated recipes) and ignores the missing values. In contrast, Euclidean and Manhattan distances consider all elements, including missing values, which can lead to less reliable similarity measures when dealing with sparse data.

Overall the algorithm did not perform that well. One reason for this can be insufficient data. The algorithm heavily relies on user-item interactions to make predictions. The algorithm may have trouble identifying significant patterns and producing reliable predictions if the training and testing datasets are limited or lack diversity. Having a larger dataset would have made the performance of this algorithm much better.

Another reason for the bad performance of the algorithm could have been due to sparsity in the data as discussed above. The dataset is too small that it could not find users that have rated similar recipes and therefore the majority of the calculations are done where the similar user's ratings are zero. Although the use of cosine similarity reduces the effect of this, it makes the predicted rating inaccurate.

To improve the performance I could have increased the dataset and incorporated dimensionality reduction to make the algorithm more scalable. Dimensionality reduction is the process of reducing the number of features (or dimensions) in a dataset while retaining as much information as possible.

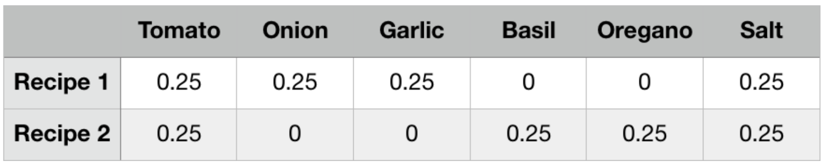
5.3.2 Content-Based Algorithm

Content-based algorithms use features of items, in our case the recipes, to recommend similar items to users. I decided to go forward with ingredients as the feature I base my algorithm on. This is similar to the way they did it on paper [4]. The algorithm uses the pre-processed ingredients lists of each recipe and vectorises it. Vectorization in our case is the process of converting text data into numerical vectors. The vectorization is done using the TfidfVectorizer from scikit-learn library.

TF-IDF stands for Term Frequency Inverse Document Frequency. It calculates a weight for each term (ingredient our case) in a document (recipe) based on its frequency within the document and across the entire dataset. For example let's take 2 recipes with the following ingredients.

Recipe 1: ["tomato", "onion", "garlic", "salt"]
Recipe 2: ["tomato", "basil", "oregano", "salt"]

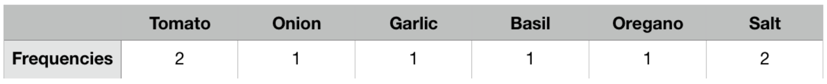
To calculate the TF-IDF matrix for this example tfidfVectorizer first creates a matrix with the term Frequancy. This is the number of times a word appears in the recipe. Figure 5.3 shows the resultant term Frequency table for the two recipes.



	Tomato	Onion	Garlic	Basil	Oregano	Salt
Recipe 1	0.25	0.25	0.25	0	0	0.25
Recipe 2	0.25	0	0	0.25	0.25	0.25

Figure 5.3: Term Frequency

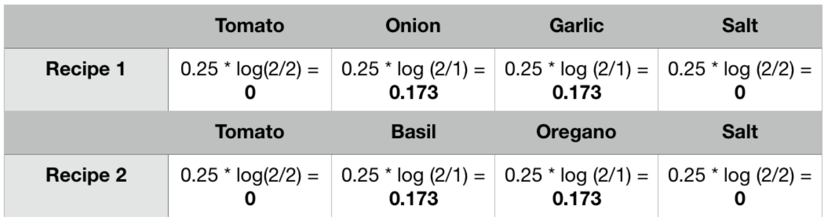
The Inverse Document Frequency (IDF) is the importance of a term across all documents (recipes) in the dataset. It helps to give more weight to terms that are less frequent across the entire dataset. To calculate IDF, we first need to determine the document frequency (DF), which is the number of documents containing a particular term.



	Tomato	Onion	Garlic	Basil	Oregano	Salt
Frequencies	2	1	1	1	1	2

Figure 5.4: Document Frequency

IDF is calculated using the log function. The mule for it is log(N/DF) where N is the total number of documents [21]. After this calculation, the TF and the IDF are multiplied to get the to form the final TF-IDF matrix. Figure 5.5 shows the final TF-IDF for the above example.



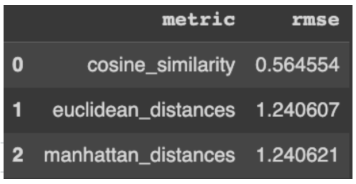
	Tomato	Onion	Garlic	Salt
Recipe 1	0.25 * log(2/2) = **0**	0.25 * log (2/1) = **0.173**	0.25 * log (2/1) = **0.173**	0.25 * log (2/2) = **0**
	Tomato	**Basil**	**Oregano**	**Salt**
Recipe 2	0.25 * log(2/2) = **0**	0.25 * log (2/1) = **0.173**	0.25 * log (2/1) = **0.173**	0.25 * log (2/2) = **0**

Figure 5.5: TF-IDF

This technique is used to convert all the recipe's ingredient list into a vector of numerical value. The resulting TF-IDF feature matrix is used to calculate the similarity between each pair of recipes in the dataset. We, therefore, end up with a similarity matrix for all the recipes in the dataset.

The predict function in this algorithm first finds all the recipes that the user has rated in the past. It then calculates the cosine similarity between the target recipe and all the recipes the user has rated. It then uses the similarity scores and the user's ratings to calculate a weighted average, where the similarity scores are the weights. The predicted rating is the weighted average of the user's ratings.

As I did not use the k-nearest neighbour for this algorithm I decided to tune the algorithm based just on the similarity metric used. I used cosine similarity, euclidean distance and Manhattan distance as the set of metrics. I again used the rmse value generated by each tuning of the algorithm to decide which metric I will choose for my final algorithm. Figure 5.6 shows the results of the tuning. Cosine similarity had the lowest RMSE therefore I will use that as the metric.



	metric	rmse
0	cosine_similarity	0.564554
1	euclidean_distances	1.240607
2	manhattan_distances	1.240621

Figure 5.6: similarity metric and its resultant RMSE

5.3.3 Hybrid Algorithm

Content-based filtering techniques do not involve the opinions of all users when recommending items and are consequently limited to making recommendations that are in the range of a user's tastes, as previously discussed. On the other hand, collaborative filtering cannot provide predictions for items that have not yet been rated, commonly referred to as the cold start problem. Therefore, hybrid filtering techniques overcome these limitations and use a combination of techniques to improve performance [16].

In weighted hybrid models, the results of multiple algorithms are combined to generate predictions by integrating the scores of each algorithm used.[12] In our case as we have worked on the user-based collaboritive filtering and content-based algorithms, I decided to combine the predictions of both of them and generate a new rating. A weightage was given to both the predictions as to which one will be given more importance. Below is the code I used to incorporate both the algorithm's predicted rating and come up with a new rating.

*hybrid_rating = UB_weightage * user_based_rating + (1-UB_weightage) * content_based_rating*

UB_weightage is the variable that holds the weightage size given to the user-based rating. The first part of the code calculates the weighted rating based on the user-based algorithm approach and the second part calculates the weighted rating based on the content-based algorithm. The weightage was tuned to see which ratio resulted in the most accurate predictions with the lowest RMSE.

While tuning the algorithm I faced a problem. The computation time and the RAM being used while executing the algorithm. This was due to my code being inefficient and the amount of data being parsed. To overcome this I decided to change some parts of my algorithms and make it more efficient. One particular example was in the code line below.

rated_indices = [self.df[self.df['recipe_id'] == x].index[0] for x in self.user_ratings['recipe_id']]

This section of code from the content-based prediction algorithm was computationally heavy. It creates a list of the indices of the rows in the similarity matrix corresponding to the recipes that the user has rated. It does this by iterating over the recipe ids in the user_ratings DataFrame and finding the index of the first row in the DataFrame where the recipe_id matches the current recipe id. This was taking a lot of time to be computed for user_id, which had rated a lot of recipes.

I updated the code by using hashing. The updated code given below creates a dictionary by zipping together the recipe_id column and a range object that generates a sequence of integers from 0 to len(self.df)-1. It then uses a list comprehension to look up the index of each rated recipe in the dictionary. After making this change the code was much faster and used less RAM.

# Create a dictionary mapping recipe ids to their indices in the DataFrame recipe_indices = dict(zip(self.df['recipe_id'], range(len(self.df))))

# Find the indices of the rows corresponding to the rated recipes using the dictionary rated_indices = [recipe_indices[x] for x in self.user_ratings['recipe_id']

After changing the prediction algorithm I performed the tuning on the weightage given to each algorithm. I tested for all possible ratios and the result is shown in the line graph in figure 5.7.

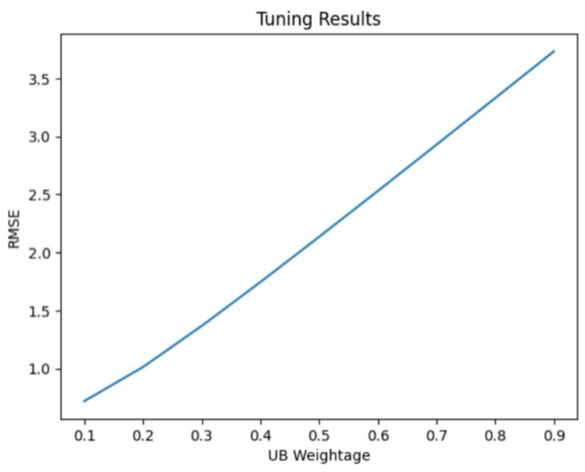


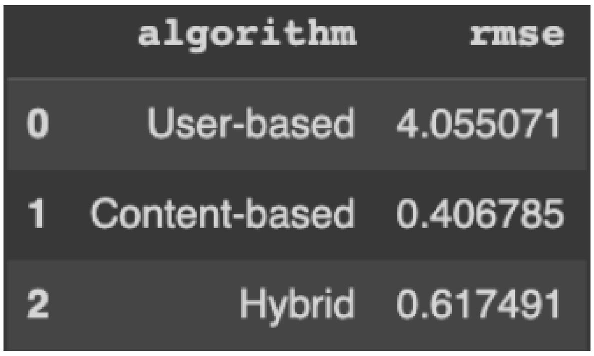
Figure 5.7: change in RMSE as the weitage of User-Based prediction increases in the Hybrid algorithm

UB_Weightage shows the amount of weightage given to the user-based algorithm. The line shows the change in the RMSE value of the hybrid algorithm as the user-based weightage is increased. It is visible that as the weightage increases, the RMSE value of the predictions also increases. Therefore the accuracy of the hybrid algorithm is the best when the weightage is of ratio 1:9 where 1 is for the user-based and 9 is for the content-based.

The performance of hybrid algorithm was highly influenced by the weak RMSE of the user-based algorithm. As discussed above due to reasons like sparsity and insufficient data, the user-based algorithm did not perform up to the mark. Therefore as we give that algorithm more weightage in our hybrid algorithm, The RMSE increases and leads to inaccurate rating predictions. Improving the user-based algorithm will in turn improve the performance of this algorithm.

5.3.4 Selected Algorithm

After the creation and tuning of all 3 algorithms, I compared the RMSE results of them based on the test sets. The RMSE results in figure 5.8 show that the content-based algorithm had the best RMSE and therefore was the most accurate in predicting the ratings.



	algorithm	rmse
0	User-based	4.055071
1	Content-based	0.406785
2	Hybrid	0.617491

Figure 5.8: RMSE of the algorithms

5.4 Health Weightage

5.4.1 FSA Score Evaluation

The healthiness of recipes could be as- sessed using various metrics (e.g., WHO, HCTS). In our study, we adopted the most commonly validated measure for food healthiness, the FSA score, which was issued by the British Food Standards Agency [18].

The FSA score was composed of four different nutrients: fat, sat- urates, sugar, and sodium. For each nutrient, it discerned between low, medium, or high content within a recipe. One point is assigned for each level (low, medium, high) per nutrient, leading to a scored scale that ran from 4 (healthiest) to 12 (least healthy). For example, fat content was designated as low if it fell below 3g per 100g served, while a medium range for saturated fat fell between 3g/100g and 17g/100g served. High recipe content is not only considered the per 100g content, but also the total weight in g per serving [19]. All computational details about the FSA score were reported in paper [20]. Figure below shows the distribution of the recipes in our entire dataset.

The figure 5.9 shows that the FSA score of 9 was the highest in the recipes.

5.4.2 Weightage Calculation

The goal of incorporating the health weightage into the algorithm was to tune the rating prediction such that the healthier recipes are given a higher predicted rating. This way, the algorithm will recommend healthier recipes to the user. Initially, I added the weightage in a way that it was all the same for all the recipes, whether the FSA was high or low. The resultant predicted ratings favouring healthy recipes but there was no distinct difference between the healthy and unhealthy recipes. To overcome this I decided to set a point in the FSA score which would be the threshold of healthy and unhealthy. I decided that any recipe which has FSA > 9 will be considered unhealthy and any recipe with a FSA score $<= 8$ will be considered healthy.

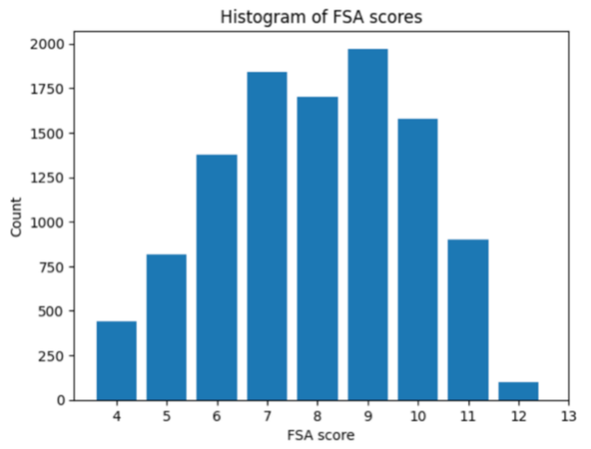


Figure 5.9: Distribution of FSA score

After this distinction, I changed the weightage given to the health according to it. If a recipe has an FSA score <= 9, only then will the health weightage be added. Below is the code that adds the health weightage to the predicted ratings.

```
1: if healthy is true then
2:     $fsa\_score \leftarrow$ calculate_FSA_score($recipe\_id$)
3:     $normalized\_fsa\_score \leftarrow (fsa\_score - 4)/8$          ▷ Adjust the range to 0-1
4:     if $fsa\_score \leq 9$ then
5:         $health\_weightage \leftarrow (1 - normalized\_fsa\_score) \times$ health_weightage
6:         $predicted\_rating \leftarrow predicted\_rating \times (1 - health\_weightage) + health\_weightage \times 5$
7:     end if
8: end if
```

Healthy boolean decides wether the health weightage will be added or not. If the user want the health weightage to be added, self.FSA_score calculates the FSA score of the recipe. Before adding the FSA score weightage, it needs to be normalised so that its value is change from the range 4-12 to 0-1. This is done by subtracting 4 from the score which is the lowest value of any FSA score and then dividing it by the range of values. The range of values is calculated by subtracting the maximum possible score from the lowest possible score which turns out to be 12 - 4 = 8. Post normalisation a health weightage is calculated only if the FSA score is healthy or <= 9. To calculate the health_weightage, We subtract the normalised score from 1 and then multiply it by the value of weightage decided by the user. This is how much the health weightage have an effect on the predicted rating of the recipe. The normalised score is subtracted from 1 so that the weightage given to the recipes with lower FSA scores is higher than the ones closer to the score of 9. After this the final prediction. Multiplying predicted_rating by (1 - health_weightage) before adding the health weightage component reduces the influence of the original rating and increases the the influence of healthiness.

5.4.3 Weightage Evaluation

To show the effect of adding a health weightage I plotted a bar graph showing the RMSE before and after the addition of the health weightage. I set the values of health weightage as 0.3,0.6 and 0.9 to see how the increase in the health weightage affects the RMSE value. It is visible in the figure 5.10 that as the health weightage is added, the RMSE of the predicted rating increase. Moreover, the RMSE tends to increase as the magnitude of health weightage is increased. This is due to the fact that the healthiness of the food is more important in this case than the user's preferences.

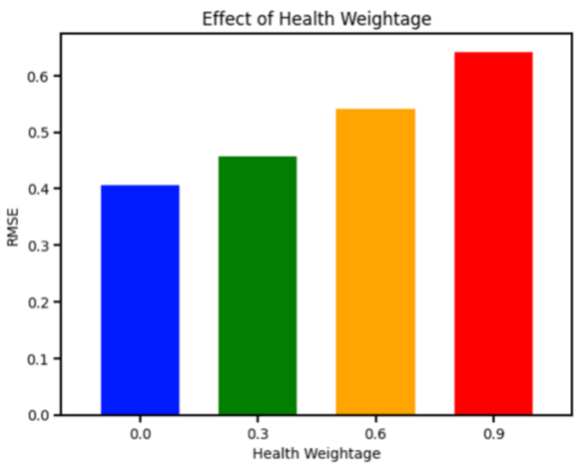
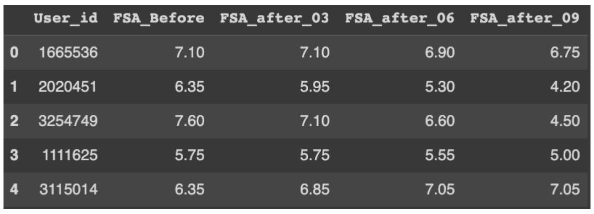


Figure 5.10: Change in RMSE as the Health Weightage is added and increased

Although there is an increase in the RMSE, the healthiness of the recipes with higher rating post the addition of the weightage are more healthy. To show this change I decided to calculate the mean FSA score of the top-20 highest-rated recipes of the users. My initial plan was to calculate the mean rating FSA score for all the users however that was turning out to be computationally very expensive. Therefore I decided to take 5 users at random from the data and show the change in Mean FSA for them. The resulting table in figure 5.11 shows that the mean FSA score decreases for all the users as the health weightage is added and increased. Therefore, the addition of health weightage is changing the recipes being recommended to be more healthy.



	User_id	FSA_Before	FSA_after_03	FSA_after_06	FSA_after_09
0	1665536	7.10	7.10	6.90	6.75
1	2020451	6.35	5.95	5.30	4.20
2	3254749	7.60	7.10	6.60	4.50
3	1111625	5.75	5.75	5.55	5.00
4	3115014	6.35	6.85	7.05	7.05

Figure 5.11: Change in average FSA score of top-20 recommended recipe as the Health Weightage is added and increased

The health weightage that would be selected for the algorithm will be subjective to the user's preferences. A higher health weightage would mean that the recipes being recommended and predicted to have a higher rating won't be much dependent on the user's taste palette and ingredients preference, but rather be more dependent on how healthy the recipe is. Users of the recommender system can decide how much importance they want to give to the healthiness of the recipe.

Chapter 6: Conclusion And Future Work

In this paper, we created 3 rating prediction algorithms and perform tuning on them. For the algorithms, I chose to work on rating prediction recommender systems rather than top-k recommendation systems, as it allows for easier evaluation. implemented three algorithms: user-based, content-based, and a hybrid of both. The user-based algorithm utilizes the k-nearest neighbours (KNN) approach to find the k-nearest neighbours of a user based on user-rating similarity and predicts the recipe rating by taking the weighted average of these neighbours. Cosine similarity was chosen as the similarity metric along with 5 nearest neighbours after the tuning stage. The content-based algorithm uses ingredients as the feature and vectorizes the pre-processed ingredient lists using TF-IDF. It calculates the similarity between each pair of recipes and predicts the rating based on the similarity scores and user ratings. Cosine similarity was again selected for the content-based algorithm as it performed the best. The hybrid combines the predictions from both user-based and content-based algorithms. tuned the weightage given to each algorithm and evaluated the results based on RMSE.

After the comparison of the algorithms, the Content-based algorithm was the best performing. The next goal of this paper was to add a health weightage to the best-performing algorithm. We introduced a threshold for distinguishing between healthy and unhealthy recipes based on the FSA (Food Standards Agency) score. We use the threshold to increase the predicted rating of the healthy recipes. The magnitude of the health weightage can be adjusted in the model. The weightage causes the RMSE to worsen as the predicted rating is now not solely dependent on the user's preferences however the top recipes that are being recommended post the addition of health weightage have a lower mean FSA score. This shows that the addition of health weightage was successful and the recipe recommendation is now adjustable by the user to be as healthy as they want.

My advanced goal was to implement user research where I would test my final model to see how it performs in the real world. I did not have enough time to conduct this research. In future, I would like to build a dataset from surveying people to give ratings to a set of recipes. Using that dataset I would recommend the recipes and ask the users how accurate they think the recommendation is according to their preference. I would also ask the users to how much importance they give to healthy food. Depending on the results of that survey I would try to predefine a health weightage that the user can just select or deselect while using the recommender system.

I would also like to increase the size of the dataset used to train the algorithms to get more accurate ratings. My system could not support the full FoodRecSys-V1 dataset that I gathered from Kaggle. Having a short dataset could have been one of the reasons that the user-based rating prediction algorithm did not perform up to the mark and had a high RMSE. It could have been that the data vas very sparse the algorithm could not find enough similar users

I would like to build a functioning app/website in the future that people can use to get food recommendations using the algorithm I built. The app would include pictures of the recipes and would show the recommended recipes in a visually appealing and effective manner.

Bibliography

[1] Pecune, F., Callebert, L., & Marsella, S. (2020). A Recommender System for Healthy and Personalized Recipe Recommendations. http://ceur-ws.org/Vol-2684/3-paginated.pdf

[3] Lopez, T., Tun, T. T., Bandara, A., Levine, M., Nuseibeh, B., & Sharp, H. (2018). An investigation of security conversations in stack overflow: Perceptions of security and community involvement. Proceedings - International Conference on Software Engineering, 26–32. https://doi.org/10.475/123_4

[4] Freyne, J., & Berkovsky, S. (n.d.). Recommending Food: Reasoning on Recipes and Ingredients. www.mturk.com

[5] "Mean Absolute Error." Mean Absolute Error - an Overview | ScienceDirect Topics, https://www.sciencedirect.co
absolute-error#: :text=The%20MAE%20score%20is%20measured,positive%20when%20calculating%20the%20M

[6] Herlocker, J. L., Konstan, J. A., Terveen, L. G., & Riedl, J. T. (n.d.). Evaluating Collaborative Filtering Recommender Systems.

[7] "Content-based Filtering | Machine Learning |." Google Developers, developers.google.com/machine-learning/recommendation/content-based/basics.

[8] MLNerds. "MAP at K: An Evaluation Metric for Ranking." Machine Learning Interviews, 3 Nov. 2021, machinelearninginterview.com/topics/machine-learning/mapatk_evaluation_metric_for_ranking.

[9] "Collaborative Filtering Advantages and Disadvantages | Machine Learning |." Google Developers, developers.google.com/machine-learning/recommendation/collaborative/summary.

[10] Yang, L., Cui, Y., Zhang, F., Pollak, J. P., Belongie, S., & Estrin, D. (2015). PlateClick: Bootstrapping food preferences through an adaptive visual interface. International Conference on Information and Knowledge Management, Proceedings, 19-23-Oct-2015, 183–192. https://doi.org/10.1145/2806416.2806544

[11] Harvey, M., Ludwig, B., & Elsweiler, D. (2013). You are what you eat: Learning user tastes for rating prediction. Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 8214 LNCS, 153–164. https://doi.org/10.1007/978-3-319-02432-5_19

[13] www.nltk.org. (n.d.). 5. Categorizing and Tagging Words. [online] Available at: https://www.nltk.org/book/c

[14] IBM (n.d.). What is the k-nearest neighbors algorithm? | IBM. [online] www.ibm.com. Available at: https://www.ibm.com/topics/knn.

[15] Briggs, J. (2021). Similarity Metrics in NLP. [online] Medium. Available at: https://towardsdatascience.com/si
metrics-in-nlp-acc0777e234c [Accessed 19 May 2023].

[16] Adomavicius, G. and Zhang, J. (2012). "Impact of data characteristics on recommender systems performance," ACM Transactions on Management Information Systems, 3(1), pp. 1-17.

[17] Elsweiler, D., Trattner, C. and Harvey, M. (2017). Exploiting Food Choice Biases for Healthier Recipe Recommendation. Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval - SIGIR '17. [online] doi:https://doi.org/10.1145/3077136.308082

[18] Department of Health and Social Care UK. 2016. Front of Pack nutrition la- belling guidance. https://www.gov.uk/government/publications/front-of-pack- nutrition- labelling- guidance

[19] Ayoub El Majjodi, Alain Starke, and Christoph Trattner. 2022. Nudging Towards Health? Examining the Merits of Nutrition Labels and Person- alization in a Recipe Recommender System. In Proceedings of the 30th ACM Conference on User Modeling, Adaptation and Personalization (UMAP '22), July 4–7, 2022, Barcelona, Spain. ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3503252.3531312

[20]Alain D Starke, Martijn C Willemsen, and Christoph Trattner. 2021. Nudging healthy choices in food search through visual attractiveness. Frontiers in Artificial Intelligence 4 (2021), 20.

[21] Chaudhary, M. (2020). TF-IDF Vectorizer scikit-learn. [online] Medium. Available at: https://medium.com/@cmukesh8688/tf-idf-vectorizer-scikit-learn-dbc0244a911a.