AI4G22 - G1: Recommendation System Group

Xavier Castillo, Matt Clark, Anushruti Huilgol, Ishan Mathur

December 15 2022

1 Introduction

Poor lifestyle choices with regards to health and fitness are directly related with negative health outcomes such as cardiovascular disease and mortality [6]. A large proportion of adults across the globe make sub-optimal dietary lifestyle choices based on a number of factors, namely lack of education, lack of time, and high stress levels. Without discussing obesity rates, it is important to maintain a balanced diet to provide the body with essential vitamins and macronutrients to maintain healthy physical and cognitive performance. Because of daily responsibilities, many adults cannot afford to spend time planning and researching proper nutrition. This is a problem, as proper nutritional guidance could lead to better health outcomes. For this reason, many companies have invested in developing tools to track, monitor and improve health and lifestyle.

To address this challenge, there are two avenues of exploration. The first is the applications or final products and interfaces to help the user, and the second is the underlying principles behind the recommendation engines that operate under the hood in online shopping, health recommendations and beyond. There are many tools and applications that one can use to gain insight on their behaviors by tracking their consumption, recording and analyzing their physical activity, structuring and recommending exercise, and more. It has become easy for users to gain in-depth insight and recommendations to optimize their health lifestyle, with a wide variety of apps catering to different lifestyle goals [2]. This industry is growing rapidly, with a market cap of over 24 Billion dollars [2] and growth of over 65 percent in 2020. As health apps come into the mainstream, this will expand from its primary focus on fitness and exercise into diet and consumption. In fact, many apps already exist to recommend new recipes based on ingredient availability, and even to recommend and structure meals for the week to promote healthy eating.

More importantly, however, are the innovations in AI that enable such dietary and health intervention tools. Recommendation and next-point of interest algorithms have become commonplace in advertising, facilitated by the development of algorithms to identify items a user may like. In the case of dietary recommendation, randomly selecting from a database of recipes would not result in user adherence to recommendations as personal preference is an important

aspect in dietary decisions. This same principle is applied to online shopping, and point of interest applications (like Yelp) which will be discussed later. To identify items that a user may like and will likely adhere to, two main approaches are commonplace in E-commerce industry applications: content-based approaches and collaborative filtering [7]. In the space of dietary recommendation, AI methods have been employed for various different purposes, below a few of them will be discussed as found in Samad et al [19].

The first AI application prevalent in dietary recommendation and analysis is image analysis [9]. The first is food recognition, which uses images to classify the type of food the user is about to consume as well as estimate the caloric value and contents. This implementation relies on the bag of visual words (BoW) model to divide an image into a visual words distribution before applying and SVM classifier [19]. Other approaches employ deep learning and convolutional neural networks to identify meals and estimate their nutritional value. Several other works have been done with similar machine learning-based approaches for classifying and quantifying user consumption based on images.

The second common theme in AI-based dietary apps is food recommendation. Mokdara et al. [12] proposed a recommendation system employing deep learning focused on recommending different Thai foods to users. The system considers the user's preferences and health, as well as their historic consumption to extract a user preference profile, which is used with the DNN to recommend a series of possible next dishes. The systems efficacy is judged by whether or not the user selected the recipe, a metric called the "hit ratio". In experimental results, the model was able to predict the users preference profile with an accuracy of 90% and achieve a 89% hit ratio on recommended recipes. Another recommendation tool focuses on recommending alternative meals to diabetic users based on nutrition and food characteristics. This tool uses Self-Organizing Maps [22] and k-means clustering for clustering analysis based on the similarity of eight significant nutrients for diabetics [13]. This method was evaluated by nutritionists and "performed well".

It is increasingly clear that there is a strong market for tools aimed at improving and facilitating healthy lifestyle choices, and there is significant room for growth and improvement in this space. Physical fitness has been the main focus for these apps, with nutritional apps only account for a "measly" 630 million of the aforementioned health app market cap [3]. This suggests that there is a gap and large opportunity for tools aimed at nutritional recommendations and health tracking. This could be a large opportunity to alleviate the stress of nutritional planning for adults seeking to maintain or begin a healthy lifestyle. In the implementation of the existing tools, we see a common pattern of Deep Learning for dietary recommendations. While this is effective in the literature mentioned above, we believe that simpler approaches used in novel item recommendation common in E-commerce platforms (such as Amazon [7]) can also be employed in recipe recommendation.

Throughout this project, our team aims to develop an AI-driven dietary-recommendation tool to suggest healthy meals for adults and their families based on a number of variables. The tool aims to use ingredient availability,

personal preferences (both dietary and lifestyle), dietary restrictions and a number of other lifestyle factors to make daily recipe recommendations to remove the burden of healthy meal planning.

Throughout this project, the team aims to develop an AI-driven dietary recommendation tool to suggest healthy recipes for adults based on their personal preferences, activity levels, and dietary restrictions. Rather than implementing deep learning or other neural network based AI methods, the team aims to employ traditional item recommendation algorithms to determine whether they are effective in recommending users recipes. Our solution will leverage collaborative filtering and content-based filtering to identify new recipes users may prefer based on their consumption history and their likeness to other users. This method is primarily used in item recommendation on E-commerce platforms, but it is believed that they will also be effective in dietary recommendation. In addition, we plan to have our recommendation tool routinely update and modify its recommendation patterns based on user adherence to recommendations as well as other preferences such as how often they are willing to repeat a recipe.

The efficacy of our recommendation tool will be evaluated based on user ratings. Ratings will capture the user's sentiment towards the recipes recommended as well as towards the system itself, including their adherence to recipes, their willingness (or likeliness) of continuing to use the tool, and their trust in the recommendations. Our system will be compared against a baseline recommendation system which has no recipe selection heuristics, only random recommendation. The goal of this work is to see significant preference from the user towards our AI-driven recommender compared to baseline. Due to time and monetary constraints, user health outcomes will not be considered in the evaluation of this tool.

Our evaluation provides results that indicate our decision making and communication agents work well, but that our learning agent could be improved. This implies our hybrid filter recommendation system approach is effective at producing dietary recommendations, but that our simple statistics-based approach to the learning agent must be improved. In all, our evaluation indicates our system accomplishes the desired goal.

2 Related Work

2.1 Recommendation Systems

2.1.1 Knowledge-Based Filter

A recommendation system is considered to be knowledge-based when it makes suggestions based on explicit knowledge such as user preferences. For example, suppose we are looking to book tickets online, we just provide the source, destination, and intended date of travel. Based on this, we get details of all the flights operating between that source and destination on that particular day. This is a basic filtering based on the explicit knowledge provided by the user.

To understand the evaluation of the BMR for the user, we went through the website BMR calculator to get the formula for evaluation.

We went through the paper [5] for understanding how knowledge-based filtering is used in their implementation and the most effective way which even we would apply in our project was that the knowledge-based was dependent only on the information user provided explicitly. The paper summarizes that in the process of interacting with the system, users elaborate on their information requirements, which makes them an essential component of the knowledge discovery process. Only an informal understanding of one's requirements and a general knowledge of the set of items are required; The system is aware of the domain's useful search strategies, category boundaries, and trade-offs. It does not consider user-ratings or any particular users. The authors have explained a technique of tweak application. In this, they used the filtering based on explicit knowledge present already about the candidates before passing it for further evaluation. The same idea has been employed in our implementation as well, where we plan to filter the recipes before passing them on to the next filter.

2.1.2 Collaborative Filter

Collaborative filters are a kind of recommendation system that make recommendations by comparing users [20]. At the most basic level, they use the logic of 'if user A has rated items similarly to user B in the past, then user B's rating for an unknown item should be similar to user A's predicted rating.' That is to say, if user A is similar to user B, and user B likes an item, then the collaborative filter will predict user A will also like the item.

Researchers have developed many approaches and applications for collaborative filtering. In [20], authors define three basic functions for collaborative filtering: *recommend items*, *predict a users score for an item*, and *recommend from a set of items*. Similarly, the authors define several tasks that can be accomplished with collaborative filtering, with basic uses such as recommending items to more complex uses like providing users with a list (or group) of similar users.

The authors then discuss basic implementations of collaborative filters. First, they discuss a user-based nearest neighbor approach. This approach is the most basic approach to collaborative filtering, and follows the approached discussed in the introductory paragraph for this section. Essentially, it takes all ratings users' provided, and scores the relationship between the users based on the similarity of responses. The more similar users are, the more likely it is one users rating of an unknown object will be representative of the others. This means, if the system has two similar users, and one user likes an item the other has not experienced, then it is likely the other user will like the item too. However, this approach can encounter problems due to sparse ratings. It is unlikely all users have rated all items, so in this scenario a collaborative approach may fail. Additionally, correlation methods such as Pearson's correlation coefficient (the one suggested by this reading) fail to consider universal agreement, such as almost all users enjoying a famous movie, such as *Star Wars*. As such, other, more

complex approaches have been developed.

More modern approaches are proposed elsewhere, such as [4] the authors discuss methods to enrich the neighborhood models. They begin by discussing a global neighborhood model which focuses on the similarity between different users (or items) to predict rating, rather than making assumptions using just one user at a time. This model allows for implicit feedback, and prevents over fitting. While this model performs well, it can still be improved using factorization, which can reduce the run-time and space complexity without impacting accuracy. They then improve this model even further, building off the factorization model to include temporal dynamics. This new model accounts for preferences changing over time. This lets the system better predict user ratings, as our biases frequently change over time. As such, this is the model we decided to use for our collaborative filter.

2.1.3 Content-Based Filter

In recommendation systems, content-based filtering plays the vital role of filtering recommended items based on the consumption history of a user. Recommended items are selected if they share similar characteristics with past items the user has selected and (possibly) rated favorably.

Each user's preferences are encoded into a set of numeric, binary or nominal attributes, usually in the form of a vector [23]. Similarly, an item's attributes are broken down into a vector. These two "feature vectors" can be compared to calculate the correlation between an item's features and a user's preferences.

Content-based filtering is used in a number of different applications. One application of content-based filtering in the context of recommendations is Personalized Recommendation Systems (PRES) [16]. PRES recommend new documents/websites to users based on their historic literature consumption. Likeness between these two can be computed by comparing the presence of similar tags, keywords or titles. Another notable recommending agent that employs content-based filtering is *CinemaScreen* [18]. Cinema screen directs the user to previously unseen (to the user) film content by comparing the users' film preferences to the attributes of a new film. Some of the features that are quantified for comparison are: actors, directors, and genres.

2.2 Communication Agent

In a recommender system, a communication agent is the interface between the system and the user. The primary function of a communication agent is to provide users and the system a channel to provide output and receive input. This input from the user can be classified as explicit, meaning the user directly gives feedback to the system in the form of a rating or like/dislike, or implicit, where the users behavior and interaction with the system itself provides insight into their opinions. A notable application of implicit feedback can be seen in *MyBehavior*, an app built to provide users with automatic health feedback based on their behaviors and preferences [15]. The application provides both exercise

and dietary recommendations. What differentiates this work from others in the field of lifestyle recommendation is its heavy reliance on user behavior as a data point versus direct (explicit) feedback. Rather than asking the user which recommendation they preferred, *MyBehavior* distills information on the user's opinion of a recommendation by their adherence or lack thereof to a suggestion. For example, if the application suggests 15 minute walk during the user's work day, their lack of adherence to their recommendation provides the system information on the user's negative feelings towards this suggestion. Alternatively, by tracking the user's movements, the system can learn that the user walks to work at a certain time most days, and recommend a walk during that time period- increasing the likelihood the user will follow through with the recommendation and adhere to further recommendations. Implicit feedback provides a lot of information about the user's preferences, for this reason, our system will attempt to distill implicit feedback into information as well. Namely, the recipes the user routinely ignores can be recorded by the communication agent to provide some quantitative implicit feedback to the learning agent.

2.3 Learning Agent

Learning agents are AI tools that begin with basic knowledge and learn from their experiences over time [17]. These agents take implicit and explicit inputs to learn and then adapt and improve their performance. Learning agents are comprised of four parts: the learning element, the performance element, the critic, and the problem generator. The learning element improves the system, by determining what is important to consider and what it should mean. The performance element is responsible for using the information determined by the learning agent, and using it to make a decision. The critic analyzes this decision and sends a message to the learning element so the agent can learn how it did. Lastly, the problem generator forces the agent into new situations where it can learn.

Learning agents can be sorted into different types depending on how environmental states are structured [17]. In an atomic representation, states are highly simplified, and cannot be any further subdivided. In a factored representation, a state consists of multiple describing features, each of which contains values. Lastly, structured representations are similar to factored representation, but they also track how each attribute is related.

3 Methods

3.1 Prior Knowledge

Our system saves data responsible for several tasks, including: filtering recipes based on user preferences and dietary restrictions, storing recipes' nutritional value for model ingestion, calculating the user's caloric intake based on physical stats, and storing the meals already recommended to the user. To calculate

the users recommended caloric intake, as well as store any dietary preferences or restrictions, the communication agent will ask the user a series of questions during their on-boarding to the system. This "on-boarding questionnaire" will ask the user to specify their dietary restrictions, provide their physical dimensions, gender, and personal preferences, such as with what frequency they accept repeating a recipe they already have cooked. The data coming from the on-boarding questionnaire is essential in customizing the recommendation tool to the user.

Perhaps the most vital role of our system's prior knowledge is the presence of the recipes themselves. We use the Epicurious Recipe database, which includes over twenty-thousand recipes, each containing a user-rating, the macro nutrient content of the recipe, ingredients and other qualitative traits such as easy-to-make. It is worth noting, for use beyond a proof of concept, a more robust database should be used, because the Epicurious database does not actually provide the instructions to the recipe, meaning users would not know how to cook their recommended recipes. However, for a proof of concept implementation, we decided the Epicurious database should function well.

Finally, after the user chooses a recipe, the prior knowledge agent will store it along with other recipes recently used by the user to avoid frequent repetition. Storing used recipes and their respective ratings is also important in our collaborative and content-based filtering recommendation mechanisms. Both the recommendation mechanisms rely on the user's consumption patterns and ratings to recommend new recipes that the user is likely to rate favorably.

3.2 Decision Making Agent

The decision making agent of our system is a hybrid recommendation system, comprised of three filters: the knowledge-based, collaborative and content-based filters. All three filters work by determining a score for each recipe, and then sending the best recipes to the next part of the system.

3.2.1 Knowledge-Based Filter

The knowledge-based filter is the first level of filtering in our system. It removes recipes the user cannot eat, and determines the best based on nutritional value. To start, the filter uses the allergy information of users and filters out all the recipes that could be harmful to the user. To do this, the system checks if that recipe has that particular ingredient that the user is allergic to and assigns that recipe a flag of -1. Such recipes will not be considered for recommendations. If the recipe passes all the allergy checks, it is assigned a score of 60.

The second part of knowledge-based filtering is calorie-based filtering. The knowledge filter wants to ensure that the recommendations will get the user to their suggested caloric intake. To determine all recipes that sufficiently fulfill the dietary requirements, we evaluate the BMR of the user based on his personal information in order to get the ideal calorie intake for the user. We compare the calorie intake value with the calorie tag in the database for filtering the recipes

that do not fall within the range of 200 plus or minus the calorie intake required for the user. We have used scoring up to 40 based on how close it is to the recommended intake value.

In the final part, we filter the recipes that were recently suggested. In the end, we pull out the top 1000 recipes to be pushed to the collaborative filter.

3.2.2 Collaborative Filter

Our collaborative filter is heavily inspired by our group's first reading [4]. At the most basic level, it will compare the active user to all other users to determine their similarity, and then combine a numerical representation of their similarity to how those other users rated potential recipes. Naturally, the challenge with this approach arises with the question: "How do we define how similar users are to one another?" Our system will deploy a statistics approach to this: cossine similarity. We will describe how this variable is calculated in later sections, but will give the general concept here. Cossine similarity looks at all ratings the user has made, and will compare these ratings to one another. In the end, it provides a single value, ranging from 90 (strong disagreement) to 0 (strong agreement). This similarity representation will then be multiplied by the rating the other user gave of that recipe to create a score for how the current user will likely respond to the current recipe, based on what another user thought of the recipe. Will calculate this for all other relevant users, add all the scores together, and then standardize them.

This approach is not without its flaws. The largest limitations of this approach are: lack of mutually rated items, and lack of detection for universally agreed upon items. We plan to address this first problem by having all users rate the same 25 recipes when they first create an account. This will ensure we have basic information to compare users. The second limitation creates bias towards popular recipes, which will make the collaborative filter recommend them too frequently. We address this in two ways. First, our system will log all recent ratings, and prevent the same recipe from being rated to the same user within a certain period of time. Second, using the collaborative filter in addition to knowledge- and content-based approaches will dilute the liklihood of universally beloved recipes being suggested.

3.2.3 Content Filter

Before learning about the content-filter, it is important to understand how one of our databases, the item-item similarity database, was generated. Our system required us to have a database which knows all the recipes containing the most similar ingredients to a given recipe. Or in other words, the content-filter needs to know how similar recipes are to one another. To generate this, we must use the data provided by the Epicurious database, which has 20,000 rows representing the recipes and 700+ columns containing tags, mostly ingredients.

At first, we attempted to calculate the similarity between every pair of items. This was a naive approach to loop the entire database and match each recipe

row with another one. In other words, it would take about $20,000! * 700$ calculations to complete, which would greatly increase the memory requirements of our system, as well as potentially take long time to calculate. Rather than iterating through the entire database, we realized this problem could be resolved using linear algebra. We multiplied the Epicurious database with its transpose, which provided a similarity between every set of recipes while this resolved the time complexity problem, it did not resolve the storage requirements, as the data was still a 20,000 x 20,000 matrix. Rather than storing how every pair of recipes relate, we stored the top 5 most similar items for each recipe. It made it easier for us to process multiple recipes very smoothly for each request. Now that we have discussed how the similarity between items was calculated, we will discuss the content-based filter.

Content-based filter is the last part of the decision-making agent, and is responsible for taking user preferences into account. Immediately after receiving the recipe list from the collaborative filter, the content-based filter loads the user's average rating for each recipe that the collaborative filter provided. The system then selects the top 5 rated elements from the list. However, there are cases where very few suggested recipes had been previously rated, in such scenarios, we considered the recipes and score from collaborative to be the base, and select the top 5 recommendations from the collaborative filter, rather than from the rating history. Once the top 5 recipes are selected, we access find the 5 items most similar to each, using our saved item similarity list (described above). We then standardize this new list, and the recommendations from collaborative filter to determine the best 5, which are then passed to the communication agent.

3.3 Communication Agent

Our communication agent accomplishes two main goals: resolving the cold-start problem, and recommending recipes.

To resolve the cold-start problem, it blatantly asks the user for information about themselves, such as dietary restrictions, physiological data, etc. It will also provide the user with a list of 25 recipes (standard across all users), and ask them to rate each recipe. These recipes help the system to preliminary understand the user preferences. They were selected using a greedy-policy which chose the best recipe to add to the set as determined by the number of unused database tags it has.

As mentioned earlier, it provides users with the recommendations determined by the decision-making agent. The communication agent will present the user with 5 recipes, and the score each filter assigned it (helping the user to understand why this recommendation was made). It prompts users to select one of the recipes, and asks them to rate it next time they log on. The communication agent also uses this opportunity to provide implicit feedback about the un-selected recipes. Any un-selected recipe is given a score of '3' or 'average.' This will help the system to gradually filter out disliked recipes despite the user never selecting them to specify they dislike the recipe.

3.4 Learning Agent

Our learning agent completes the important goal of learning the user preferences. We utilize basic statistics to do this, specifically, we take the mean of all scores the user has given that tag or recipe. As users use the system, we will update the score for each element by recalculating the mean based on the supplied information. To do this, we save the number of times the user has rated this element, and the average score. When a new score is added, we multiply the average rating by number of ratings, add the new rating, and then divide by number of ratings +1. This gives us the new mean, for the recipe or tag, which the learning agent assumes is the preference.

We decided on this approach to account for the fact larger, more intensive machine-learning approaches require access to data about user preferences that we do not currently have. Using basic statistics, the system will be able to adapt without requiring as much background knowledge. After a period of prolonged use, a better approach such as reinforcement or deep learning could be used, but because we will not actually use the system for long enough for that to be practical, we decided to just use basic statistics.

4 Architectural Design

4.1 Environment

We would like to begin this section by noting: the environment is not part of the agent. However, it is still worth discussing, so we will briefly do that here.

Our system will pull three types of information from the environment: internal factors about the user, external factors about the environment, and sensed inputs. Internal factors will be provided by the user through the app. Upon creating their account (and updated when the user wishes), they will be asked to provide their: height, weight, dietary restrictions and preferences, how many meals should be recommended at once, amount of time they have to cook, their weight goal (whether they wish to gain, maintain, or lose weight), their cooking ability, and their expected activity level. Then, every time they use the app, they will be asked to rate recipes they have recently made. This information will assist the system in making appropriate recommendations that are in line with the user's goal and lifestyle.

In order to make accurate recommendations to the user, our system must know the user's activity level. In full implementation, we could gather information from the user's devices, such as a FitBit or their cell phone. These values could be used to determine how active the user is, and by extension, how many calories they burned and should consume. However, for our implementation and system at this time, we settled on a more practical approach. This approach is to simply ask the user to define their activity level. We define three activity levels: not very active (¡4,000 steps a day), moderately active (4,000-8,000 steps a day), very active (¿8,000 steps a day). These thresholds were determined because science indicates individuals who walk 4,400 steps a day had lower mor-

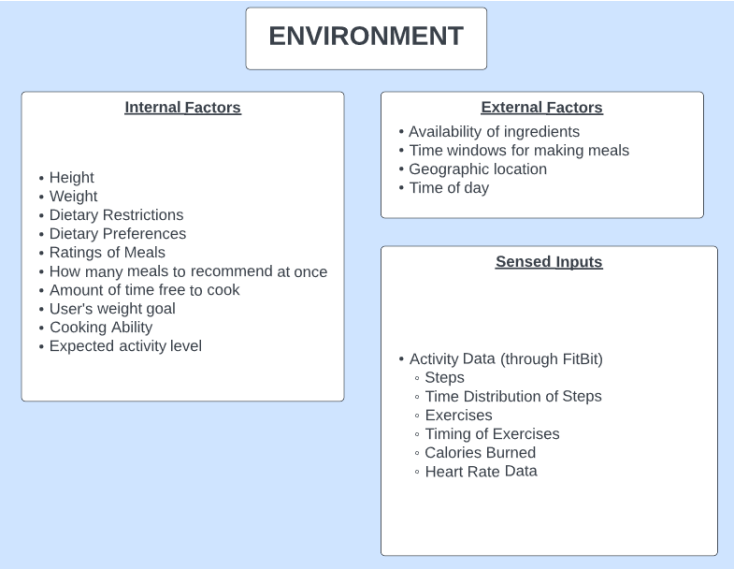


Figure 1: The environment of the system

tality rates than those who took fewer, and the health benefits continued until around 7,500 steps [11]. To keep the system as straight forward for users as possible, we rounded both of these numbers to the nearest thousand. Once the user chooses and activity option, it will be used by the knowledge-based filter to make future decisions.

4.2 Decision Making Agent

The recommendation system will take the information discussed above from the prior knowledge. It will then split up the information so that each filter gets the information necessary. Each recommendation filter will rate the possible recipes on a scale from 0-100. Using these scores, each filter will select the best, and pass that set to the next filter. Each filter reduces the size of the set of possible recommendations, until only 5 remain, which are then communicated to the user along with the score from each filter.

4.2.1 Knowledge-Based Filter

The architectural diagram for knowledge-based recommendation system is shown in figure 2. The inputs that the knowledge-based filter takes are:

- User Preference
- BMR of the person
- Recipe Data set

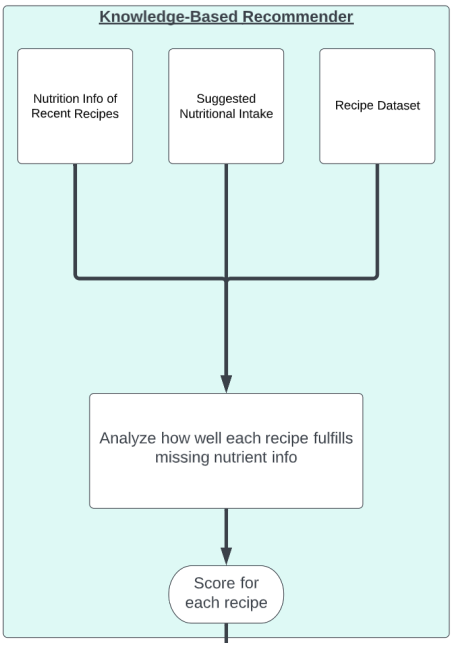


Figure 2: Architectural Design of the Knowledge-Based filter

- Recently suggested Recipes

The first step in the knowledge-based filter is to evaluate the calories per meal required for the user. We take the BMR and activity level of the person. Based on these two factors, we evaluate the calories needed for a person per meal. For simplicity, we have considered that a user would have four meals (breakfast, lunch, dinner, and snacks) a day and divided the value by four.

The second step is to filter the recipes based on the recipes that do not match the ideal calorie intake for the user. We do this based on the calorie tag present in the Epicurious database, which stores the number of calories in the meal. For this, we compare the calorie information in the database with the ideal intake per meal for the user in three different ranges. The first range would be, if the calorie information of the database falls in the range of plus or minus 50 of the ideal calorie intake, those recipes would get the highest score of 40. For the recipes falling in the range of plus or minus 100, we would score them with 30 and for the recipes falling in the range of plus or minus 200, we would score them with 20.

The third step the knowledge-based filter takes is filtering out recipes that the user could be allergic to. For this, we consider the user's dietary preference and compare it with tags such as vegetarian, soy-free, dairy-free, gluten-free and vegan. If the user has at least one of these restrictions and the recipe fails to satisfy this, the recipe will be set to a score of -1 and later dropped from the

list of recipes that can be sent to the next filter.

The final filter is to remove the recipes that were recently suggested. For this, we take the information of the recently suggested recipes from database and then filter all those recipes. In practice, the length of the excluded recipe list should be provided by the user, specified by how frequently they would like to repeat recipes. Once we have the list of valid recipes and their scores, we take the 1000 recipes with the highest scores and pass them to the collaborative filter.

4.2.2 Collaborative Filter

The collaborative filter (shown in figure3) requires two pieces of information: the current user's recipe dataset provided by the knowledge-based filter, and the matrix storing the ratings all users made of all recipes. Using this information, the collaborative filter will calculate the predicted rating for every possible recipe (mathematical formulas will be provided and discussed in the Formal Design section). It will then select the best recipes and provide those to the content filter.

However, as discussed above, each filter must rate the recipes on a scale from 0-100 for the communication agent. Predicting the user's rating will follow the same scale our system uses, which will be 1-5. So, the collaborative filter must standardize the scores before passing them on to the content-based filter. To do this, we decided to make all scores relative to the set provided by the knowledge-based filter. To do this, we simply converted all scores to a percentage of the best score (which would then get a 100%). We decided to do this because the scores will hopefully look higher than using a fully standardized method, which hopefully will make users view the recommendations in a better light.

4.2.3 Content-Based Filter

The content filter is the last node of the 3 layered sequential hybrid-filter setup. It receives a certain number of recipes from collaborative filter as input. Apart from it, content filter also uses information from prior knowledge and databases.

Here, we will discuss the diagrammatic flow of the content-based filter. The initial two blocks in Figure 4 represent the knowledge or information the content-based filter begins with. The first step in the content filter is taking the intersect of the two recipe lists it received. If there are no intersecting recipes, as seen in the "No" portion of the control flow, the recipe data from the collaborative filter is considered for further steps. The score for each recipe is calculated, discussed in later section, and stored in a hashmap, which is represented by the cylinder in Figure 4. The data filtered through this preliminary step in the content filter proceeds to the next stage described in the following paragraph.

In this second stage of the content filter seen in Figure 4, which corresponds to the flow diagram after the "Yes" and "No" branches, we use the data from prior-knowledge to store similar items. The diagram depicts initially getting similar items and it is followed by using the user preference to generate new

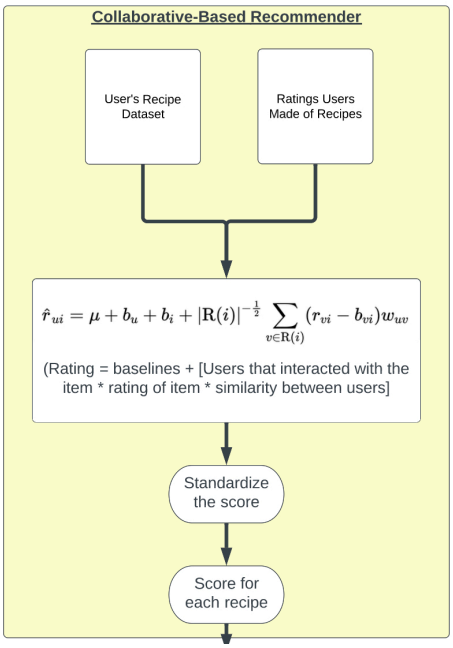


Figure 3: Architectural design of the collaborative filter

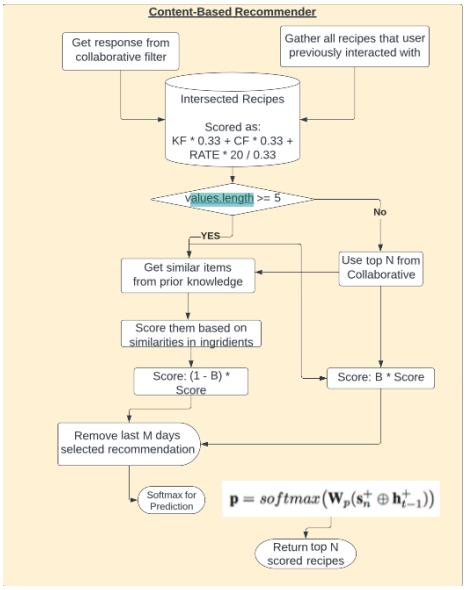


Figure 4: Content Based Filter Initial steps

Figure 5: Actuators

scores. Finally, the content filter filters out the recipes already selected by the user within the last 3 days and performs a softmax operation on the scores, returning the recipes and their scores.

4.3 Communication Agent

In order to understand communication agent, we will discuss it in two important parts. As seen in figure 5), actuators are responsible for sending information from agent to the user. The diagram shows two sections, asking user feedback on the last recommended recipe. The user gets to choose from the given list of 5 options about. Then getting processed results from agent to show to the user. Here also user input is needed where they'll choose most preferred of 5.

Now, the figure 6) refers the sensors. They are used to derive information from user and environment and pass it to the agent. As we mentioned in the previous paragraph, that the user is supposed to rate and select the recipes, this information is actually captured by the sensor and provided to learning agent in our case to process further. User inputs is just one of many things that are seen in the image. These information can be implicit and explicit as well as. For example, every user is able to provide information regarding their weight, height, dietary restrictions and we are also gathering information like geo-location, temperature, etc.

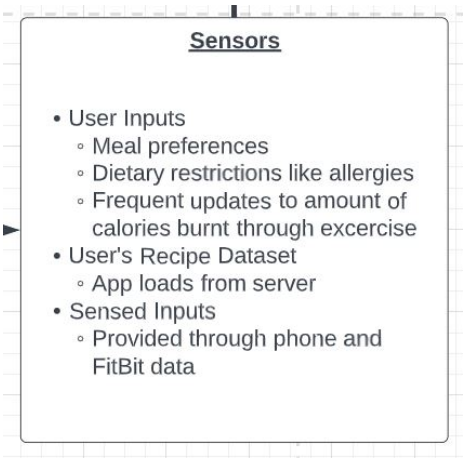


Figure 6: Sensors

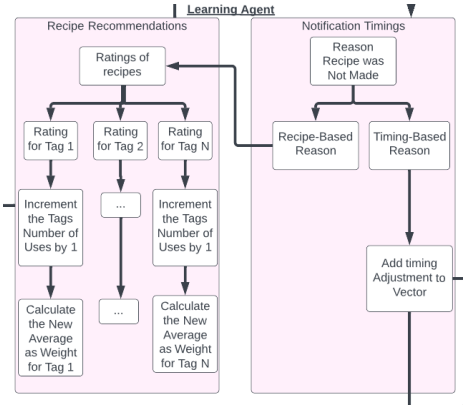


Figure 7: Architectural design of the learning agent

4.4 Learning Agent

Our system's learning agent (shown in figure 7) interfaces with the communication agent. The communication agent provides it with the user id, recipe, and the user's rating. It will then use this information to update how the user feels about the recipe and its tags. Over time, this should capture the user's dietary preferences. For example, if a user does not like mushrooms, and he consistently rates dishes with mushrooms poorly, the system will figure out the user does not like mushrooms, and will stop recommending them.

5 Formal Design

5.1 Decision Making Agent

5.1.1 Knowledge-Based Filter

In the knowledge-based filter, we consider four inputs:

- 'A': Allergy information of the user.
- 'C': Calorie intake per meal for the user based on BMR and activity levels.
- 'R': The recently suggested recipes.
- 'X': The recipes

Lastly, let 'L' denote the output, the list of recipes that will be passed to the collaborative filter.

As discussed in the earlier sections, the knowledge-based filtering reduces the list of recipes based upon the user's dietary preference, the healthiness of the recipe based on the calories, and removal the recently suggested recipes. We will discuss how each of these methods function, and how they assign scores.

Overall, the formula for the knowledge-based filter is:

$$L = max(X_{S(f(A,R))} + X_{S(f(C))})$$

Definition, Nutritional Requirements: The knowledge-based filter begins by determining how well each recipe matches a user's nutritional requirements. The formula for BMR is:

$$\text{For men : } BMR = 10xWeight + 6.25xHeight - 5xAge + 5$$

$$\text{For women : } BMR = 10xweight + 6.25xheight - 5xAge - 161$$

$f(C) : C(e)| \sim |C(u)$ where C(e) is the calorie information in the Epicurious database and C(u) is the ideal calorie intake of the user per meal ($BMR_{\overline{4).}}$

f(C) is the similarity function between the calorie information from the Epicurious database and the calories that the user should be taking. To evaluate S(f(C)), we verify how close the calorie information of the recipe is to the ideal calorie intake per meal for the user. We first check if the calorie information of the recipe falls in the range of +/-50, if yes, we assign it a score of 40. If it falls within the range of +/-100, we assign it a score of 30 and 20 if it falls in the range of -/+200.

Definition, Dietary Matching: After calculating the nutritional requirements, the knowledge-based filter determines how well each recipe matches the user's dietary preferences and excludes recipes that the user has already consumed. In the equation below, we define this as f(A,R). We can define f(A,R) as,

$$f(A, R) = X(e) - X(A) \cup X(R)$$

As mentioned throughout this report, each filter of the recommendation system assigns a score to each recipe. For this portion of the knowledge-based filter, we denote the score as $S(f(A, R))$. X(A) would be the recipe not satisfying the dietary restrictions of the user and X(R) would be the recipes that were recently suggested. We filter these out from the Epicurious database by scoring them with -1. Such recipes are not considered to be picked in the top 1000. For all the recipes that are not X(A) or X(R), get a score of 60.

5.1.2 Collaborative Filter

As discussed above, the collaborative filter requires two pieces of information, the user's recipe data set, RD, and the matrix of scores, $\mathbf{S}$ will be a $m*n$ matrix where m is the number of users, and n is the number of tags and recipes. For this filter to work, we require three steps. First, we must calculate the similarity between users, followed by the rating for each recipe, and then the standardized rating for each recipe.

Definition, Cosine Similarity: We will use cosine similarity to calculate the similarity score between users. The formula for cosine similarity [1] is:

$$sim(x, y) = \frac{x \cdot y}{||x|| * ||y||}$$

where x is the item rating vector for user i (the active user) and y is the item rating vector for user j (the user being compared to). Additionally, $||x||$ and $||y||$ are the euclidean norm of vectors x and y, respectively. The formula to calculate these scores is (shown for x, but can be applied to y without loss of generality):

$$\sqrt{\sum_{i=0}^{len(x)} x_i^2}$$

In the end, this measure calculates the angle between two vectors, ranging from 0 (perfect agreement) to 90 (perfect disagreement). By taking the cosine of this score, we can calculate the similarity score, on a range from 1 to -1:

$$w_{i,j} = cos(sim(x, y))$$

Definition, Collaborative Recipe Ratings: Once the similarity measure is calculated, we can calculate the predicted rating for each recipe, $\hat{r}_{ui}$. As provided by [4], the equation to calculate $\hat{r}_{ui}$ is:

$$\hat{r}_{ui} = \mu + b_u + b_i + |R(i)|^{\frac{-1}{2}} \sum_{v \in R(i)} (r_{vi} - b_{vi}) w_{uv}$$

Clearly, there are lots of symbols to define in this equation. We will begin with the non-bias terms. Similarly, $R(i)$ is the set of users that rated the recipe, i. Meanwhile, $r_{v,i}$ is the rating user v gave recipe i. Lastly, $w_{u,v}$ was discussed in the previous section, and captures the similarity between users u and v.

Most bias terms are simple to calculate, however, one is not. μ is the bias of the system overall, which we can calculate by averaging the rating for all recipes, provided by all users. b_u is the bias of the current user, calculated by averaging all of their ratings. b_i is the bias for the specific recipe being rated. Just like the other biases, it is calculated by averaging all existing scores for that recipe. The final bias term appears later in the equation, b_{vi} is slightly more complicated than the previous bias terms. In layman's terms, it accounts for fluctuations in ratings over time. To do so, ratings must be split up into 'bins' to capture the time when the rating was reported. To calculate b_{vi} we will use the linear+ equation (which provides a better than average root mean square error while still being simple to calculate):

$$b_{v,i} = \mu + b_v + \alpha_v * dev_v(t_{vi}) + b_{v,t_{vi}} + b_i + b_{i,Bin(t_{vi})}$$

μ, b_v, and b_i have already been discussed, so we will not discuss how to calculate them again. Instead, we will focus on the new elements:

- $\alpha_v * dev_v(t)$ is a linear model to calculate how the ratings change over time, between dates t and t_v. In other terms, captures the linear changes (or slope) of the changes over time.
- t_{vi} is the rating user v provided for item i at that time t.
- $b_{v,t_{vi}}$ is the bias user v had for item i at time t. This may be calculated using a linear equation.
- $b_{v,Bin(t_{vi})}$ is another way of calculating the bias user v had for item i at time t. To calculate it, we can take the average of the ratings in the time-based bin.

Once all bias terms are calculated, we can find the predicted rating, $\hat{r}_{ui}$ user i will have for recipe u.

Definition, Standardized Collaborative Recipe Ratings: $\hat{r}_{ui}$ captures the predicted rating user i will have for recipe u. However, this scale will be 1-5, and this filter must provide its score on a scale of 0-100. So, we must standardize the scores to the same scale as the other filters. To do so, we will use the simple equation:

$$Score = (\frac{\hat{r}_{ui}}{max(\hat{r}_{ui})}) * 100$$

Once we have these standardized scores, we take the best recipes, and pass them along to the content-filter to be further narrowed down.

5.1.3 Content-Based Filter

Content based filter performs many operations and at each point we'll define the required variables. From the top of the function, we have:

- 'CR': The set of recipes from the collaborative filter.
- 'R': The set items user has previously rated
- 'U': String representing User ID
- 'ED': The matrix of Epicurious database.
- 'SID': A HashMap of similar items in database.
- 'RH': A set of recommendation history.

Definition, About: Before diving into the functioning of the filter, it is important to know how can we create the 'SID', i.e. the database of similar items. As mentioned previously, each value here indicates the number of common ingredients between each recipes (row vs col).

Definition, Matrix Multiplication: We solved it

Definition, Commonality: As stated in the architectural diagram above, after getting all the required information, the first operation is finding common elements from both the sets.

$$I = CR \cap R$$

Definition, Combining Scores: Assigning a score to the recipes is done by taking scores from every filter and combining them with equal weights. Since the ratings are given in range 1-5, we first multiply it by factor of 20 to make sure all scores are normalized

$$\begin{aligned} newScore_i = (0.33 * CR_i.knowledgeScore)+ \\ (0.33 * CR_i.collaborativeScore)+ \\ (0.33 * (20 * R_i.score)) \end{aligned}$$

Definition, Getting Similar Items: Let the newly generated set of recipes be *newRecipes*. Now we use the SID to get similar recipes. As it is a hashmap, the time-complexity for finding the recipe is in $O(1)$. So, it will return set of recipes and their scores in the following manner:

$$similarItems(I_i) = \{\{recipeId, score\}, ...\}$$

Definition, Adding Bias: Now that we have $newRecipes$ and $similarItems$ we can use the $U.longTermPreference$ attribute which would add a bias, consider repeating factor, for similar items (not recommended previously) and items (recommended previously) in the following way:

$$score(similarItem_i) = (1 - U.longTermPreference) * score(similarItem_i)$$

$$score(newRecipes_i) = U.longTermPreference * score(newRecipes_i)$$

Definition, Unique results in form of probability: It is essential to remove all the recipes which user has already chosen in last few days. After doing that, the last step to apply softmax function to convert every score to probabilistic score by the following method.

$$\sigma(I_i) = \left(\frac{e^{I_i}}{\sum_j e^{I_j}} \right) j = 1, ..., n$$

5.2 Learning Agent

The learning agent is responsible to update the average ratings of the recipes once the user has rated them.

- 'R': The recent rating of the recipe
- 'A': The average rating of the recipe
- 'O': The number of times the recipe has been rated.

Lastly, let's say 'S' be the new average for the recipe. To evaluate the average rating, we use the formula

$S = \frac{(A \times O) + R}{O}$

This score is taken as the new average rating for the recipe.

5.3 Communication Agent

The communication agent consists of two important modules, actuators and sensors. They are responsible for providing user the output, asking user about feedback and continuously inform the learning agent about the activity (usually rating or choosing the recipes) from user.

Apart from providing user recipe list, we are also asking them to rate the recipe they selected the last time. Based on what user rated, we receive an implicit feedback that the remaining 4 recipes weren't as good as others. Since there is no indication about why the other 4 recipes weren't selected, we give

them a general score of 3. By giving it a fair score, it allowed us to make sure we are not eliminating the recipe entirely as well as making sure to change the average rating so as to better evaluate next time.

Let's say we have:

- 'I': The set of recipes recommended to user
- 'R': Recipe user rated
- 'S': Numeric rating value for selected recipe
- 'C': Numeric hyperparameter decided by us to rate every other recipe

$$score(R) = S$$
$$score(I_i) = C \qquad \forall \qquad i \neq R$$

6 Implementation

We implemented a command line-based version of our recommender system in Python. We will structure the implementation discussion based on how the user will encounter it. As such, we will approach this discussion in the order of: cold-start solution, knowledge-filter, collaborative-filter, content-filter, communication agent, and finally the learning agent. Our entire system is based on the Epicurious database [10], providing 20,052 recipes and 674 'tags' helping to define the recipe.

6.1 Cold-Start Solution

Throughout the paper we have considered the cold-start solution to be part of the communication agent, and still do. However, for ease of flow, we have separated it for this section.

When the user first runs the system, it will ask them to login. The user will initially not have an account, so if the name the user types is not previously in the system, it will run the cold-start solution to learn initial user preferences.

The system will begin by collecting data about the user's preferences. It will begin by asking about dietary restrictions. In our current implementation, we only inquire about: vegetarianism, veganism, gluten-free, soy-free, and dairy free. In a real implementation, we would like to cater to a more diverse set of dietary restrictions, but these were the common restrictions that we could easily define using the current data set.

After collecting the dietary restrictions, the system will gather physiological data from the user. Specifically, wuser'slect, the users height and weight, activity level, weight goal (lose, maintain, or gain weight), gender, and age. These are necessary for the knowledge-filter to calculate how many calories the user should consume.

The final 'physiological data' we ask the user is how frequently they would like to repeat recipes, which determines the frequency with which the content-filter overrides the other filters.

While these questions are necessary for our decision-making agent to function, we still have no way to quantify the user's preferences. As such. to solve the cold start, users will be asked to rate twenty-five recipes, on a scale from one to five. The scale provided is:

1. Would not make
2. Probably would not make
3. Unsure
4. Probably would make
5. Would make

After rating one recipe, the user will automatically be presented with the next recipe. All users will be shown the same twenty five recipes, regardless of dietary preferences. This decision was made to guarantee all users had shared recipes (a requirement for the collaborative filter). We determined these recipes by deploying a greedy algorithm to determine what recipe would add the most new tags, and had the code loop twenty-five times. In all, our 25 recipes use 285 of the 674 tags, giving the system a strong head start to calculate recipe preferences.

These cold start ratings will be treated as a normal recipe, for the sake of initial data collection, and will be passed to the learning agent as if a normal rating. While the user will technically encounter the learning agent at this time, we will discuss that implementation at the end of this section.

After finishing the cold-start solution, the system will use the decision-making agent to make a recommendation. This process begins with the knowledge filter.

6.2 Decision-Making Agent

6.2.1 Knowledge-Filter

The implementation of the knowledge-based filter begins with taking the four different inputs. We have stored all the information needed in a CSV file and to load it, we use the read_csv() from pandas. This information is:

- BMR of the person - We load the information from the bodyData.csv file into a data frame named as body_data. We match the username with the user column in the CSV to get the specific user's body data
- Recipe Data set - We load this information from Epicurious database and store in a data frame named as df

- Recently suggested recipes - We save the recently suggested recipes in a csv and load it into a data frame
- User Preference - From the user.csv file, we load the user's dietary preference and store it in a data frame named user_data

After loading of all the required data, we begin the filtering process by evaluating the BMR of the person. For this, we use the gender-specific formula, For men :

$$BMR = (10 * user\_details['weight'] + (6.25 * user\_details['height']) - (5 * user\_details['age']) + 5$$

And for women :

$$BMR = (10 * user\_details['weight']) + (6.25 * user\_details['height']) - (5 * user\_details['age']) - 161$$

After we have the BMR, based on activity levels, we multiply it with a constant value. This score is further divided by 4 considering the user would have four meals a day.

Once we have the per-meal-caloric-intake value, the next task would be to filter the recipes that do not satisfy the calorie criteria. For this, we load the Epicurious database and compare the calories tag in the database with the calorie-intake-per-meal and score based on the three different ranges, plus or minus 50, plus or minus 100 and plus or minus 200. We append the recipe IDs into an array and perform further processing only on these recipes.

For further processing of recipes based on allergy information, we check the user's dietary preference and if any, we check the recipes from the Epicurious database for those particular tags. If the recipe fails to meet the dietary preference, we set the score to -1. If it satisfies, then add 60 to the score.

After that, we check if the recipe was suggested in the last 50 recipes, if yes, we set the score to -1. This is done by loading the file last_recommendations stored by the communication agent.

Once we have all the scores, we drop all the recipes with -1 score from the data frame and then sort it descending. We pull out the top 1000 recipes based on the score and pass it on to the collaborative filter

6.2.2 Collaborative-Filter

The collaborative filter operates by taking the reduced recipe set from the knowledge-filter, and the current user. Initially, it must calculate the cosine similarity score to understand how to weigh every user in relation to the current one. To calculate cosine similarity, we begin by reading all the user's ratings from ratings.csv and use numpy to calculate the similarity score [8]. Before calculating scores, we also randomly shuffle the list provided by the knowledge filter. This means, in the case of long-standing ties, the system will be more likely to make different recommendations.

After calculating the cosine scores, we calculate the bias scores. This is done by averaging all necessary data (determined by the kind of bias). User

bias averages all of the user's ratings, universal bias averages all ratings in the system, and item bias averages all ratings for the recipe being calculated.

After calculating these biases, we calculate the time bias. The method we are using requires us to use time-based bins. In a real implementation, these bins should be larger, likely week based. However, to implement and evaluate our system, we use day-sized bins.

To do so, we use the statsmodels python package [21]. Using the ols method within the statsmodels package, we can calculate necessary time biases using linear regression. We factor in the slope to represent $\alpha_u * dev_u(t)$, and use linear regression to predict the score for the current time frame to represent $b_{u,t_{uj}}$. Finally, we must calculate $b_{j,Bin(t_{uj})}$, which we get by averaging the scores in the necessary time bin.

After calculating these biases, we use the equation defined in the Formal Design section to calculate the predicted score. We then use pandas [24], which lets us quickly and efficiently compare the score for each recipe, ranking them. We then take the top 50 recipes, calculate their score on a scale of 1-100, and send them to the content-filter.

6.2.3 Content Filter

Before going into the content filter implementation, we will briefly discuss how the item-item similarity matrix is generated. To perform this computationally heavy task, we implemented the code using Python's numpy library, and ran the computations on a Google Colab GPU. The multiplication of the matrices took a matter of minutes. To store the data, JSON's were used to map each recipe to a list of the five most similar recipes in the database. The scoring was done relatively, meaning the most similar recipes achieved a score of 100 and the remaining recipes were scaled to fit between 0 and 100.

The content filter code starts by reading the data from the files statically, i.e. recipes database and the item similarity file. All the files which are dynamically changed throughout the recommender, like user history or ratings, are loaded each time they are used to properly represent the changes. The main function takes in the recipes outputted by the collaborative filter as well the user's preferences covered in prior knowledge. It also loads the previously interacted recipes from ratings.csv and processes them using the technique described in the methods section. Numpy arrays were used for most of the code implementations of array maths, allowing the computational overhead to be as minimal as possible. The list of recipes is finally passed to a final_results function which is responsible for considering multiple important factors

- Don't add the recipes recommended in last 3 days
- Add a multiple of bias to the final score of similar recipes as well as the original list of recipes
- Perform softmax on final list to normalize the score and smoothening the values.

The results from this content filter are final, and are then displayed to the user and our communication agent listens for user input to process further.

6.3 Communication Agent

After the final recommendations are calculated, the user encounters the communication agent. The agent will present all of the provided recipes, and the rating each filter gave it. These ratings will be provided using basic terms that are indicative of the filter's meaning. Knowledge-based is provided as 'Match based on nutrition:', collaborative states 'Match based on users like you:', and content states 'Match based on other items you like:'. Our hope is that these explanations will help the user to understand why these recommendations were made.

The user will be prompted to choose one of these recipes. The recipe they choose will be logged, as it is the recipe the user indicated they would like to make. The other recipes will be given an average score of 3, as implicit feedback. Our hope is that this will help the system to stop recommending items if the user continuously picks others above it.

The next time the user logs into the system, they will be prompted to rate the recipe rather than go through the cold-start solution. To define each of the five possible scores, we prompt the user using:

1. Strongly Disliked
2. Disliked
3. Unsure
4. Liked
5. Strongly Liked

Once the user rates the item, the system will send the score to the learning agent, and then generate the next set of recipes, beginning the process again.

6.4 Learning Agent

Throughout this section, we have been discussing the learning agent. The learning agent will use the methods discussed in previous sections, primarily averaging the scores for all tags and recipes. These values will all be read, and saved from csv files stored within the system.

As mentioned above, The learning agent is responsible for updating the user preferences. For this, we take the input from the communication agent on what the user rated for the particular recipe they chose to make previously. Based on this, we update the rating for this particular recipe by adjusting the average score for the recipe.

Calculate the new average : totalScore = float(ratings[recipe][tag]) * float(occurences[recipe][tag])
Add in the new ratings : totalScore += float(rating)

Calculate the new average : totalScore = totalScore / (float(occurences[recipe][rating]) + 1.0)

We then update the two csv namely, ratings.csv with the new average for this recipe and increment the times the user interacted with the recipe in occurrences.csv.

7 Evaluation

To evaluate our system, we are coordinating with group 4. All nine members between both groups plan to test both systems. We plan to generate 10 recipes a day from each system, over the course of a week. While doing so, we plan to answer survey questions about the system, as well as use a third baseline system. The survey questions will allow us to qualitatively compare our systems, through measures such as 'I found it easy to tell the system what I like / dislike' that may be challenging to capture using raw data. We will also quantitatively compare our systems by analyzing the ratings users provided for each recipe.

7.1 Baseline Recommender

As mentioned before, we intend to use a third recommender system to serve as a baseline in comparison to our systems. This system contains no learning, or personalization. It uses the Epicurious database, and randomly selects one of the 20,052 recipes for the user. Every ten recipes, the user will be asked to answer the survey. Our hope is that without personalization or learning, this will be an accurate representation of a baseline recommender to compare our systems to.

7.2 Survey Evaluation

Most of our evaluation will come from the surveys the user fills out. To design this survey, we took the relevant questions directly from [14], a published paper that outlines a survey to evaluate recommender systems. The questions we ask in our survey are shown in table 1, and all will be asked in the format of a 5-point Likert scale. Participants will be asked to complete this survey after every 10 recommendations, which should correspond to once a day for our evaluation.

We plan to evaluate our system using statistical analysis. Specifically, we intend to use Kruskal-Wallis tests to detect differences between our systems, and between the baseline. The Kruskal-Wallis test is the non-parametric version of the ANOVA test, and is used to identify significance between groups.

7.3 Offline Evaluation

We also plan to evaluate how user's rating changes over time. To do this, we will linear regression to model how each user's responses changed over time. This will allow us to calculate the slope and intercept for every user's ratings, for each

Survey Question
The recipe(s) matched my interests.
The recommender explains why the recipes are recommended to me.
The recommender allows me to tell what I like / dislike.
I found it easy to tell the system what I like / dislike.
I found it easy to inform the system if I dislike / like the recommended item.
I felt in control of modifying my taste profile.
The recommender allows me to modify my taste profile.
I found it easy to modify my taste profile in the recommender.
Overall, I am staisfied with the recommender.
I am convinced of the items recommended to me.
I will use this recommender frequently.
The recommender made me more confident about my selection / decision.
The recommender can be trusted.
I will use this recommender again.
I will use this recommender frequently.
I will tell my friends about this recommender.
The recommender helped me find the ideal item.
Using the recommender to find what I like is easy.
The recommender gave me good suggestions.

Table 1: The questions from our evaluation survey

system. Since this data was also collected through a Likert scale, measuring how well users liked the recipe, we will also analyze this data with Kruskal-Wallis tests to determine how each system performed compared to each other.

7.4 User Study

Between groups 1 and 4, we recruited 7 friends to use all three systems, in a random order. The name of each system was anonymized to reduce bias in the responses. When doing the randomized baseline and group 1's recommender, participants were asked to rate 70 recipes, and take the survey every 10 recipes. Because group 4's recommender suggests 3 meals at a time, participants were asked to rate 24 days worth of recipes (72 recipes total), and take the survey every 3 'days.'

It appears not all participants followed the provided methodology exactly, likely due to the informal nature of our evaluation. Additionally, there was one extra participant who only analyzed group 4's system. In all, we received 56 survey responses for the baseline recommender (instead of the expected 49), 63 from group 4s recommender (instead of the expected 64), and 53 for our recommender (instead of the expected 49). We leave these extra surveys in the data for our analysis, because it would may impact results to guess which responses to remove.

Survey Question	Group 1 Mean	Baseline Mean	P-Value	Group 4 Mean	P-Value
The recipe(s) matched my interests.	3.811	2.982	0.00268231612952654*	3.746	0.6217297861238711
The recommender explains why the recipes are recommended to me.	3.585	1.464	2.396468538509173e-12*	2.635	0.0005007608994421055*
The recommender allows me to tell what I like / dislike.	4.302	3.161	0.00021680499933640266*	4.079	0.12311403781496184
I found it easy to tell the system what I like / dislike.	4.283	3.304	0.0012766641319767243*	4.222	0.6203155331742591
I found it easy to inform the system if I dislike / like the recommended item.	4.34	3.232	0.00058169487856453 08*	4.27	0.3355572492294075
I felt in control of modifying my taste profile.	4.019	2.643	1.1973984761745783e-05*	3.794	0.15724815345715232
The recommender allows me to modify my taste profile.	4.151	2.857	1.9706294431271 18e-06*	4.032	0.11627164018358391
I found it easy to modify my taste profile in the recommender.	4.057	2.679	3.6065341947732986e-07*	3.825	0.07016263886129291
Overall, I am satisfied with the recommender.	3.77	2.571	2.055111972664 77e-05*	3.667	0.6583335022684695
I am convinced of the items recommended to me.	3.604	2.571	0.00020231555897020785*	3.651	0.9768605369226923
I will use this recommender frequently.	3.509	2.143	8.558266053139899e-07*	3.381	0.3966338561358015
The recommender made me more confident about my selection / decision.	3.642	2.625	0.00019901258920833193*	3.714	0.9286962541174476
The recommender can be trusted.	3.604	2.482	4.056874955781394e-05*	3.587	0.5293918326855787
I will use this recommender again.	3.49	2.268	1.3074460886281324e-05*	3.333	0.2955043144942807
I will use this recommender frequently.	3.491	2.214	3.8978434301362 12e-06*	3.365	0.3521836590498747
I will tell my friends about this recommender.	3.377	2.393	0.00037316524037059 9*	3.238	0.3766139022218258
The recommender helped me find the ideal item.	3.868	2.464	9.425888826425992e-07*	3.81	0.5202044005433342
Using the recommender to find what I like is easy.	3.698	2.5	1.2586039267012737e-05*	3.762	0.7601651989263191
The recommender gave me good suggestions.	3.83	2.732	5.0204840328385e-05*	3.857	0.7595614219827067

Table 2: The average score each system received in the survey, and the significance when compared to our system. * denotes significance at 95% confidence.

8 Results

In this section, we analyze the results of the user study for our system compared to the other systems.

8.1 Survey Responses

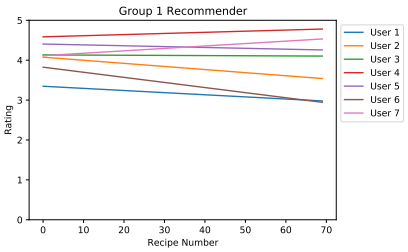
The average Likert scores for each question and system are shown in table 2 and are graphed in the appendix. Our system received significantly better scores than the random recommender for every question. With this, we can safely assume that our system is better than the random baseline. Our system is capable of making better recommendations, making recommendations more reliably, explaining why the recommendation was made, and was generally liked better than the random recommender. Thus, we can conclude our system works well since it was significantly better than the random baseline recommender in every analyzed category.

Our system was highly comparable to group 1's system. Of all survey questions, only one was significantly different between the systems. Participants reported ours did a better job explaining why recommendations were made ($p = 0.0005$). This is likely due to our communication agent reporting the score from each filter at the same time as the recipe. In all, this shows while our decision-making and learning agents were similar between systems, ours had a marginally better communication agent.

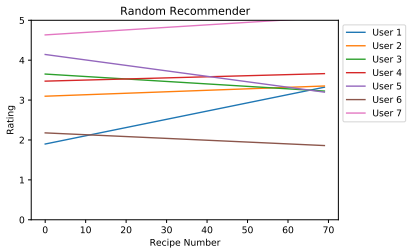
8.2 Rating Responses

We will now analyze the ratings users gave each recipe (shown in figure 8). We will begin by discussing the difference in starting coefficients, then the differences in slope, and finally significance tests between the systems.

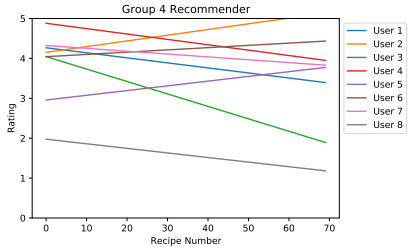
For our system, the average beginning rating (coefficient) was 4.06 (on a scale from 1-5). This is the highest of the three systems, with the baseline's beginning rating being 3.30, and group 4's being 3.83. This means, when provided with only cold-start information, our system outperforms the other recommendation



(a) Recipe ratings for group 1's system



(b) Recipe ratings for baseline system



(c) Recipe ratings for group 4's system

Figure 8: Recipe ratings for each system and user

systems. This result is indicates our cold start solution works better than group 4, and better than no cold start at all.

Interestingly, only one system's average slope increased. On average, only the random recommender improved over time, with an average slope of 0.0012. This increase is marginal, which is to be expected because the system is not learning, it is simply pulling random recipes from the list. As such, we would expect no change, which the average slope is very close to. Yet, both group 4's system and our own got slightly worse over time. Our system's ratings had an average slope of -0.0028, and group 4 had an average slope of -0.0055. Clearly, these ratings are also very close to 0, but still have a negative slope. This shows our systems did not learn well over time, meaning the learning agent for both systems could be improved.

Lastly, we performed Kruskal-Wallis tests to compare the ratings between each system. Overall, the average rating for our system was the highest: 4.07. Group 4's system had the next highest average rating: 3.83. The system with the lowest average rating was the baseline, with an average rating of 3.30. When comparing the ratings of each system with a Kruskal-Wallis test, we find that our system is significantly better than the baseline ($p = 2.16e\text{-}08$) and group 4's system ($p = 7.53e\text{-}06$). Thus, not only did our system start with the best recommendations immediately after the cold start, but it also gave the best recommendations overall.

9 Discussion

Our results indicate our decision-making and communication agents work well, and that our learning agent could be improved. Both the survey and rating analysis indicate our system provides significantly better recommendations than the baseline, while analyzing just the ratings themselves indicates our system makes better recommendation than group 4's system. The recommendations themselves are produced by our decision-making agent, so these results indicate our decision-making agent comprised of the three types of filters works well.

Our results also indicated our communication agent worked well. According to the survey responses, the only area in which participants were consciously aware our system was better than group 4's was in explaining why each recommendation occurred. This is likely due to the fact our system's communication agent will provide the score from each filter in the decision-making agent. These scores help the user to understand how well the recommender thinks the recipe matches the user's nutrition requirements, taste based on similar users, and taste based on similar items. Thus, we believe our communication agent to be effective.

The learning agent of our system, however, seems to have underperformed. Ideally, the user's ratings should have increased while using the system. This would indicate the system was correctly learning the user's preferences, and making better recommendations with this information. However, the average user's rating slightly decreased. While this decrease was small, and better than group 4's, it still indicates our learning agent is underperforming. As a result, we could improve our system by utilizing a more robust approach. One such approach could be using reinforcement learning, which would be similar to our current approach, but would use more robust equations to determine adjust preferences than simply taking the mean. While we believe this would improve our learning agent, we feel it is worth noting, the decrease in average rating was very small (-0.0028), indicating our simple approach is not horrible, but simply does not have the performance we would hope for.

We also would like to take a moment to discuss a few bugs and flaws in our implementation that we discovered. First and foremost, the system may randomly crash after completing the cold start. This occurs for roughly 1/10 participants, at which point that participant becomes permanently unusable due to incomplete data. We were able to identify where the bug occurs, and in the cause, but not the context. Occasionally, the system may mistake a user for a recipe, and attempt to get the nutritional information about a user. Even when using print statements and reading the entire list of 1,000 recipes to analyze how the user appeared as a recipe, no cause could be found (because the user was not present in the printed list). As such, we wrote code to mitigate this problem when it occurs. Running the script will remove the participant from the system, they will have to redo the cold start, but the system will likely be usable after this. Additionally, we identified a flaw with the time component of the collaborative filter. During implementation, we forgot to add the time bias for ratings after the cold start. The system would still calculate

the change over time, but it could only compare to the preferences at the time of the cold start, because those were the only time these preferences were saved. Lastly, participants expressed they found the cold start question provided by the collaborative filter confusing. Users did not understand what to put when asked "How likely are you to repeat a recipe? On a scale from 0 (unlikely) to 1 (likely)." If a participant provided a low value here, the collaborative filter would override the other filters very frequently. However, the phrasing of this question is not indicative of this outcome, which then also led to participants being confused about why the collaborative filter was overriding the other filters so frequently. As such, while the system evaluation indicates our system works well, especially our decision making and communication agents, there were still places these could have been improved in our implementation.

10 Conclusion

For this project, we created a dietary recommendation system. Our system uses a hybrid filter approach, combining knowledge, content, and collaborative filters to produce a recommendation. Each filter produces a score which the communication agent then provides with a short list of 5 possible recommendations, helping the user to select the recipe that best matches their interests. Our system learns by taking the mean of all ratings for the relevant tags for the recipe, but this is shown to underperform, meaning more robust methods could greatly improve our system. We evaluated this system in a small user study, and found our system's communication and decision-making agents perform significantly better than average (as provided from by a random recommendation baseline), but our learning agent could be improved.

References

[1] Cosine similarity.

[2] mhealth apps market size, share & covid-19 impact analysis, by app type (disease & treatment management, wellness management, others), by application (monitoring services, fitness solutiosn, diagnostic services, treatment services, others), by market place (google play store, apple app store, other), and regional forecast, 2021-2028.

[3] Nutrition apps - united states: Statista market forecast.

[4] *Recommender Systems Handbook*. Springer US, Boston, MA, 2011.

[5] Burke, R. Knowledge-based recommender systems.

[6] E.KingMD, D., G.MainousIIIPhD, A., MarkCarnemollaBS, J.EverettPhD, C., Sturm, R., Strine, T., King, D., Mokdad, A., Stampfer, M., Knoops, K., and et al., May 2009.

[7] Hardesty, L. The history of amazon's recommendation algorithm, Dec 2022.

[8] Harris, C. R., Millman, K. J., van der Walt, S. J., Gommers, R., Virtanen, P., Cournapeau, D., Wieser, E., Taylor, J., Berg, S., Smith, N. J., Kern, R., Picus, M., Hoyer, S., van Kerkwijk, M. H., Brett, M., Haldane, A., del Río, J. F., Wiebe, M., Peterson, P., Gérard-Marchant, P., Sheppard, K., Reddy, T., Weckesser, W., Abbasi, H., Gohlke, C., and Oliphant, T. E. Array programming with NumPy. *Nature 585*, 7825 (Sept. 2020), 357–362.

[9] Hassannejad, H., Matrella, G., Ciampolini, P., De Munari, I., Mordonini, M., and Cagnoni, S. Food image recognition using very deep convolutional networks. MADiMa '16, Association for Computing Machinery, p. 41–49.

[10] HugoDarwood. Epicurious - recipes with rating and nutrition, Feb 2017.

[11] Lee, I.-M., Shiroma, E. J., Kamada, M., Bassett, D. R., Matthews, C. E., and Buring, J. E. Association of step volume and intensity with all-cause mortality in older women. *JAMA Internal Medicine 179*, 8 (Aug 2019), 1105–1112.

[12] Mokdara, T., Pusawiro, P., and Harnsomburana, J. Personalized food recommendation using deep neural network. In *2018 Seventh ICT International Student Project Conference (ICT-ISPC)* (2018), pp. 1–4.

[13] Phanich, M., Pholkul, P., and Phimoltares, S. Food recommendation system using clustering analysis for diabetic patients. In *2010 International Conference on Information Science and Applications* (2010), pp. 1–8.

[14] Pu, P., Chen, L., and Hu, R. A user-centric evaluation framework of recommender systems. *RecSys* (2011), 8.

[15] Rabbi, M., Aung, M. H., Zhang, M., and Choudhury, T. Mybehavior: Automatic personalized health feedback from user behaviors and preferences using smartphones. In *Proceedings of the 2015 ACM International Joint Conference on Pervasive and Ubiquitous Computing* (New York, NY, USA, 2015), UbiComp '15, Association for Computing Machinery, p. 707–718.

[16] Raghavendra, C., Srikantaiah, K., and Venugopal, K. Personalized recommendation systems (pres): A comprehensive study and research issues. *International Journal of Modern Education & Computer Science 10*, 10 (2018).

[17] Russell, S., and Norvig, P. *Artificial Intelligence A Modern Approach*, 4 ed. Pearson.

[18] Salter, J., and Antonopoulos, N. Cinemascreen recommender agent: combining collaborative and content-based filtering. *IEEE Intelligent Systems 21*, 1 (2006), 35–41.

[19] Samad, S., Ahmed, F., Naher, S., Kabir, M. A., Das, A., Amin, S., and Islam, S. M. S. Smartphone apps for tracking food consumption and recommendations: Evaluating artificial intelligence-based functionalities, features and quality of current apps. *Intelligent Systems with Applications 15* (2022), 200103.

[20] Schafer, J. B., Frankowski, D., Herlocker, J., and Sen, S. Collaborative filtering recommender systems. 35.

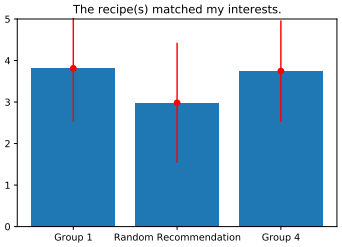
[21] Seabold, S., and Perktold, J. statsmodels: Econometric and statistical modeling with python. In *9th Python in Science Conference* (2010).

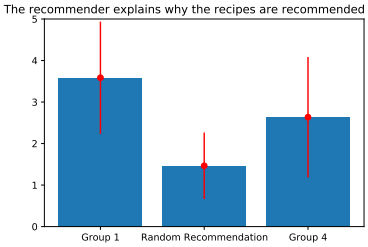
[22] Van Hulle, M. M. Self-organizing maps. *Handbook of natural computing 1* (2012), 585–622.

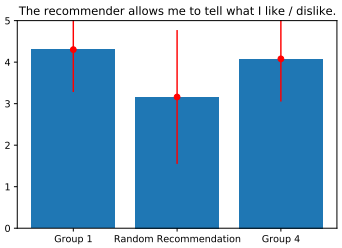
[23] van Metern, R., and van Someren, M. Using content-based filtering for recommendation.

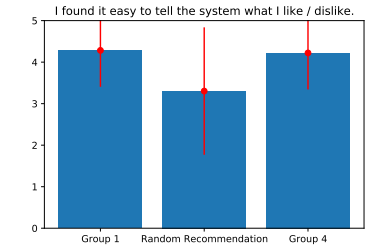
[24] Wes McKinney. Data Structures for Statistical Computing in Python. In *Proceedings of the 9th Python in Science Conference* (2010), Stéfan van der Walt and Jarrod Millman, Eds., pp. 56 – 61.

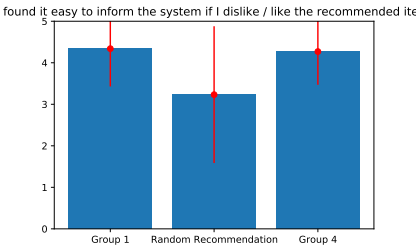
11 Appendix

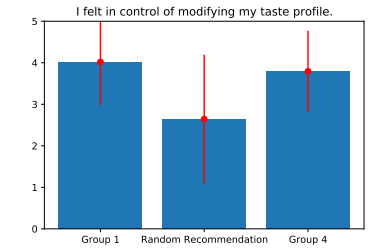

The recipe(s) matched my interests.
5
4
3
2
1
0
Group 1
Random Recommendation
Group 4

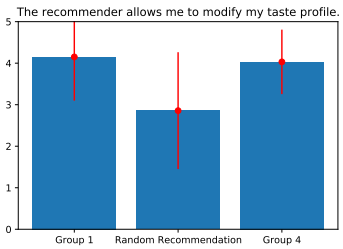

The recommender explains why the recipes are recommended to me.
5
4
3
2
1
0
Group 1
Random Recommendation
Group 4

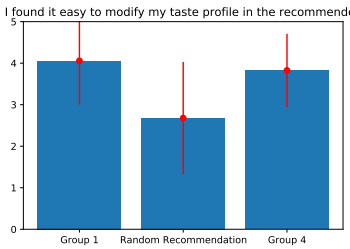

The recommender allows me to tell what I like / dislike.
5
4
3
2
1
0
Group 1
Random Recommendation
Group 4

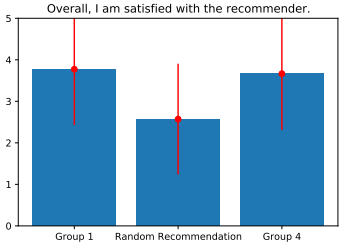

I found it easy to tell the system what I like / dislike.
5
4
3
2
1
0
Group 1
Random Recommendation
Group 4


found it easy to inform the system if I dislike / like the recommended ite
5
4
3
2
1
0
Group 1
Random Recommendation
Group 4

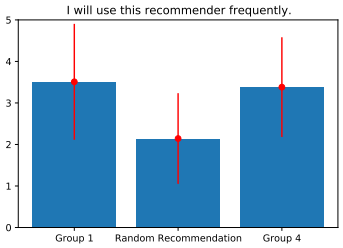

I felt in control of modifying my taste profile.
5
4
3
2
1
0
Group 1
Random Recommendation
Group 4

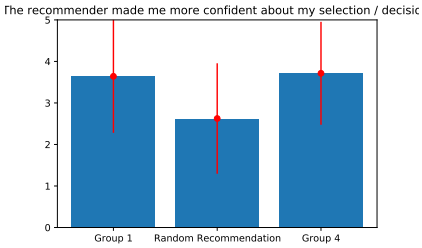

The recommender allows me to modify my taste profile.
0
1
2
3
4
5
Group 1
Random Recommendation
Group 4

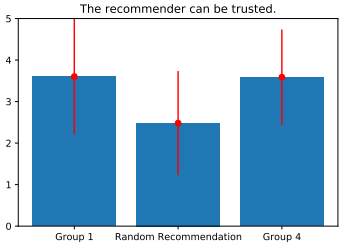

I found it easy to modify my taste profile in the recommender.
0
1
2
3
4
5
Group 1
Random Recommendation
Group 4

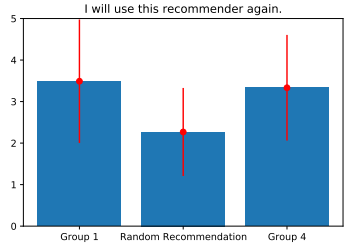

Overall, I am satisfied with the recommender.
0
1
2
3
4
5
Group 1
Random Recommendation
Group 4

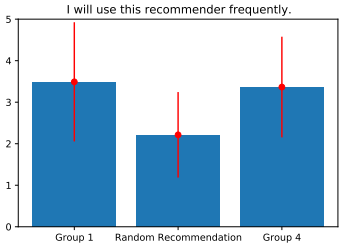

I am convinced of the items recommended to me.
0
1
2
3
4
5
Group 1
Random Recommendation
Group 4

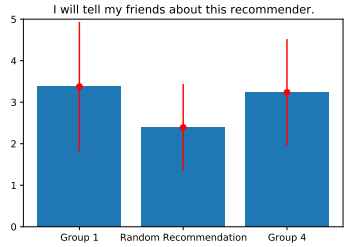

I will use this recommender frequently.
0
1
2
3
4
5
Group 1
Random Recommendation
Group 4


The recommender made me more confident about my selection / decisio
0
1
2
3
4
5
Group 1
Random Recommendation
Group 4


The recommender can be trusted.
0
1
2
3
4
5
Group 1
Random Recommendation
Group 4


I will use this recommender again.
0
1
2
3
4
5
Group 1
Random Recommendation
Group 4


I will use this recommender frequently.
0
1
2
3
4
5
Group 1
Random Recommendation
Group 4


I will tell my friends about this recommender.
0
1
2
3
4
5
Group 1
Random Recommendation
Group 4

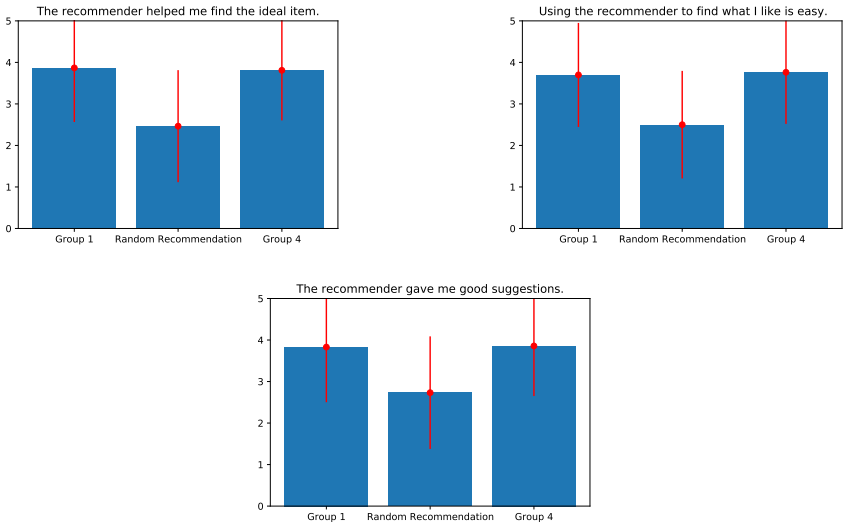


Figure 11: Responses to the evaluation question survey