Recipe Recommendation System

Team members:

Mohammad Abu Ali, mohammad.abuali@mail.huji.ac.il, abu.omar, 208887687

Abdalrhm Kharubi, Abdalrhm.Kharubi@mail.huji.ac.il, abd_khrubi, 208978585

Asem Elayan, asem.elayan@mail.huji.ac.il, asem.elayan, 315899336

Problem Description

We are trying to find a set of recipes tailored towards the user's taste. Given his choice of meat, carb source and spiciness level, the program advices him recipes which the user should find appealing. And we have differentiated between these categories due to the fact that ingredients are intertwined, and what suits a certain type of meat may not suit another.

Our idea revolves around starting with a data set and reduce it to fit the user's taste.

Data

Our data consists of the recipes that are found in the domain ***yummly.com***.

Each recipe in our data contained many fields, among which are the description of the recipe, the cooking time and the ingredients. We have chosen to work on the ingredients alone.

We have spilt our data among 24 categories, each category contained one type of meat (chicken, beef or pork), one type of carbohydrate sources (rice, bread, potato or pasta) and the spiciness level (hot or mild).

At the beginning we predicted we would be working on a data set of size 1.5GB, based on the number of recipes we wanted to scrape. However, after we have filtered the data, and focused on the recipes that fit only one of the categories mentioned above at once, we have managed to reduce the data to 0.15 GB.

Solution

At the start, we filtered the data we have scraped, so that the only attributes we have are: the recipes' names, the ingredients, the number of servings, the amount of likes(known as yums), the rating, the url of both the yummly page of the recipe and the image url, the cooking time and the spiciness level. Where we have separated the data into 24 json files, one for each category.

In order to manipulate the data to fit our purpose, we wanted to focus on the ingredients, and so after compiling all the ingredients from all the recipes from all the 24 categories, we had around 8800 ingredients. However, some of these ingredients were repeated with different words, for example: apples and fresh apples. Also we have found out that some of the ingredients where repeated in the singular and plural forms, and some had company names next to the ingredient.

At first, we tried to use part of speech tagging, by removing everything that isn't a noun, and so we would get rid of all the unnecessary adjectives. However, after some experiment, we have found out that the NLTK classifier was not accurate enough, and it kept misclassifying words. After which we have tried SpaCy POS tagger, but we received the same problem. And so we have found our self in a need to build filters, which will help us to accurately map any iteration of one ingredient to that specific ingredient, for example: frozen bananas, fresh bananas, chopped bananas and banana will all be mapped to banana. To do so we have built the following filters:

1) Remove_category: We have decided to remove certain categories and chose not to map any of their ingredients. Mainly due to the fact that ingredients from these categories will bring no value for our algorithm, for example: meat- since at the start we always choose one type of meat, then its iterations either will certainly be existing in the recipes, or none of them will. And so there is no need to map them to an ingredient
2) Clean_pepper: A filter for everything that belongs to the family of peppers. We have mapped everything pepper related to 5 categories: cayenne, chili powder and black pepper for spices, sweet pepper and hot pepper for physical peppers and finally hot sauce for sauces made from peppers.
3) Clean_starches: Since we wanted to differentiate between corn starch and corn and given that types of starches tend to have the same effect- which is thickening the food- then we have decided to map everything starch related to starch.
4) Clean_Dairy: Dairy products were tricky, the names of different types of cheese are so predominant in our world that the name of each kind is enough to make someone

understand we are talking about cheese. And since we wanted to treat all the cheese kinds the same, then we mapped all our cheese types to cheese. As for butter, we had different types of nut butter, apple butter and buttermilk. Each of those don't functions the same as butter in recipes, so we mapped each of them to their own category, the same with butter.

5) Clean_oils: Different types of oil where not as problematic. However, there were some intersections between types of oil, for example: canola oil, sesame oil and cooking oil all refer to frying oil, also sometimes the word spray replaces oil but does the same job. So we have mapped those to cooking oil. Also, we took care of olive oil, since it had many variations.

6) Reduce_categories: We have decided to map some categories of food, in their entirety, to a specific ingredient. For example: different types of alcohol were all mapped to alcohol, also all seafood ingredients apart from anchovy where mapped to seafood, mainly due to the fact that those are not the source of protein and therefore they are exchangeable and thus we treated them the same. As for anchovies, we are aware it is used in other recipes such as pesto, where no other type of sea food would work, so we left it out.

7) Clean_tokens: We have found other ingredients that needed the same treatment as pepper and dairy, however there cases had not been as problematic and all of them required a simple checking if the name of the ingredient contained a substring of one of our tokens, and if so it maps the whole ingredient to the value of the key that is the token.

8) Clean_nuts: we checked all types of nuts available from a Wikipedia page, we then found out that all of them contains the substring "nut", apart from a few selection which we included in the function, we mapped all of them to nut. And we checked the case of coconut, which we map it to coconut. And finally, we checked if the ingredient is nut milk, and if saw then we did nothing as it is the job of clean_dairy to map it.

9) Clean_redundant: We have checked the ingredients and found out certain recurrent words that the nltk POS tagger couldn't identify correctly. And then we replaced those words and other words which have no effect on the ingredient, with an empty string

10) Clean_nlp: after all those filters we have finally filtered all the words which are not main words in the recipe using pos tagger. Since some of the important words in the ingredients where wrongly mapped to jj, we have taken the safe path by cleaning only the verbs, and since nltk usually maps some of the things ending with "ed" as verbs, then we have removed a lot of undesired words which would have been too much to filter by hand.

As a summary for this part, we apply certain filters in same order as above, to all the ingredients of all the recipes of the chosen category out of the 24 categories. And for each of those ingredients, we map it to specific ingredient based on the filters above.

Now for the how our program works. We start by asking the user on site which type of meat he desires, followed by which type of carb source he desires, after that we ask him how spicy he wants the dish. And based on that we pick the category which fits his current preference put of the 24 possible categories (all possible combinations of 3 types of meat, 4 types of carb sources and 2 spiciness levels). After he picks the category, we start by building a vector of weights.

The weight vector is built via a method that iterates over all the recipes in the category and takes from each recipe all its ingredients. First we build an object("ingredient_occurence"), where the keys are the ingredients and the values are the number of times this specific ingredient existed in all recipes, and we will use the ingredient amount in order to specify a weight for the ingredient. Then we iterate over the top 25% recipes based on the amount of likes(yums) each recipe has(we have created an ordered vector of recipes), and then we increase the weight vector(also built as an object with its keys as ingredients) of each occurrence of each ingredient in those recipes, by their weight- which we have built a method that calculates it by taking into account how many times it appears in recipes, where the weight of an ingredient of low frequency is higher than that of a high frequency- divided by two, in order to take into account the taste of the user as a bigger factor than the taste of other people while still considering the taste of the masses. And this way we build the initial weight vector.

For each recipe, we have given the user the ability to rate the recipes, with a thumbs up to indicate he likes it or a thumbs down to indicate he doesn't like it. Then for each recipe we increase or decrease the weight vector, where the coordinates fit the ingredients of the recipe by the weight we have given each ingredient. And then we suggest the user a new recipe. This continues for 10 iterations each time, where it becomes time to clean up the recipes.

At first we check the weight vector for the top ingredients(those with the highest weights) and the bottom ingredients, then we iterate over all recipes and giving them a core based on the difference between the amount of top ingredients and bottom ingredients(we consider him indifferent to what exist in the middle), and if the score is positive, we check if the recipe is in the relegation zone(a vector we insert recipes which are candidates for removal if they have had a negative score between top and bottom recipes). And if it were, we reduce the number of times it has been in the relegation zone by one. However, if the difference is negative, then we check if the recipe has already been in the relegation zone, if so, we check if it has been in the relegation zone for a certain number of times (along side the recipes the user voted negatively for), and if so we delete it, otherwise, we increment it by one

We also save the user's preferences, so that when he goes back to the same category, his precious choices are still saved.

We have also included the option to rerun the program so that the user can choose a different category, plus the options to remove his preferences for the current category and remove all his preferences.

Since we have built a website function as our program, then we have utilised the local storage to save the important data for each category, including the count of recipes we have recommended the user(so that we still filter in incrementations of 10) and the weight vector.

Evaluation

After manually rating the categories chicken_pasta_spicy, beef_rice_spice, chicken_rice_spicy and beef_potato_mild, from different team members, we noticed that at the start the recipes were all over the place, but after 60-90 recipes, there was an increase in the amount of liked recipes in each 10 iterations(we have deleted the manually liked recipes from the data set, but we have saved them in a special vector).

Then after resetting preferences and liking every recipe recommended, the recipes recommended were all over the place, as if they were random. And the same happens when trying to dislike every recipe. And so, the weight vector clearly works as intended.

Setup

In our design, we chose to build a website to run our program. Then we manually checked our program, there was no need for extra volunteers.

Results

The results we received were the following:

Each time we reach count%10 = 0, then the number of recipes in out data set decreases by at least 10 and at most 46.

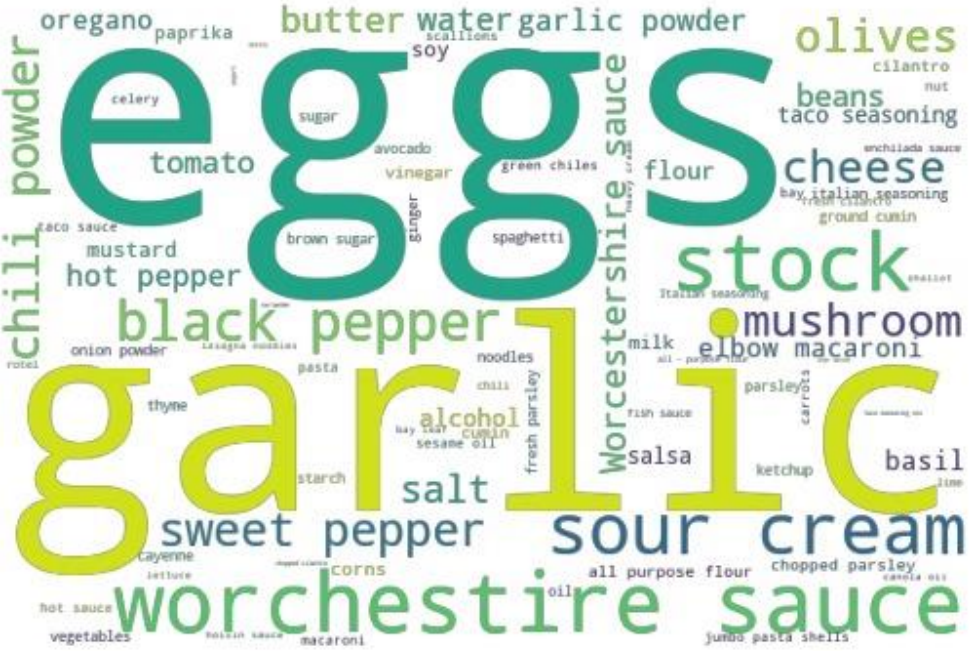
The number of recipes it took to reach a stage where the number of recommended recipes are mostly great for the taste of the user was a bit high, which we could have reduced by taking a more aggressive approach towards the score of recipes, but in the end we decided it’s better to suggest more recipes while knowing there is a chance the user wold not like them and thus increase the recipes required to be rated in order to reach the stage mentioned above, where if we were more aggressive we would have reached that stage earlier, but we would also have removed recipes which the user may like.

Visualization

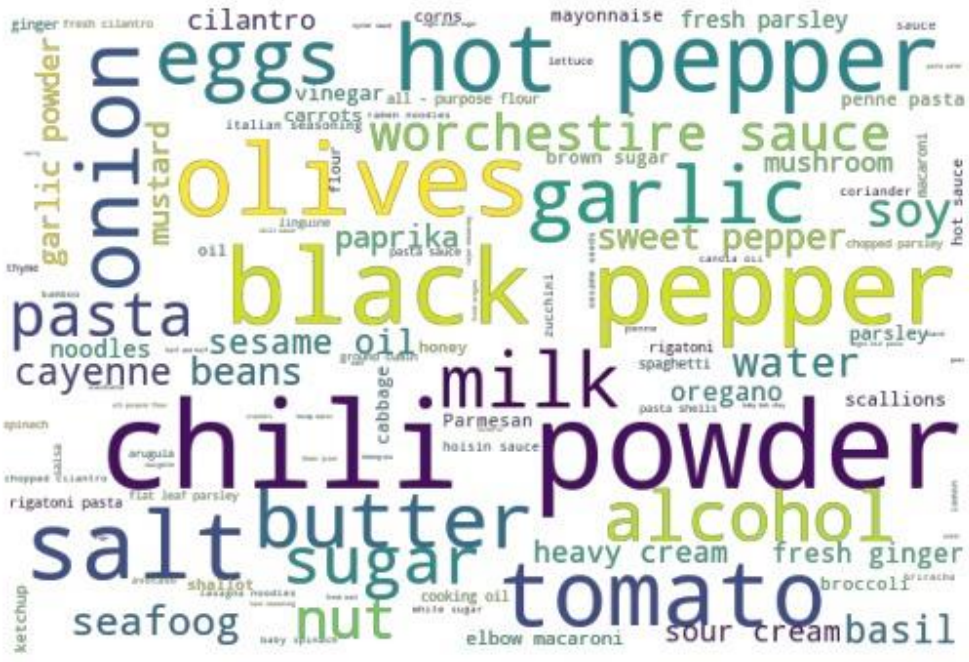
For visualization, we have chosen pasta as a carb source, and spiciness level as hot, and for each type of meat, we created a word cloud with the ingredients provided with their number of recurrences in the recipes of the category.

chicken_pasta_hot

Beef_pasta_hot

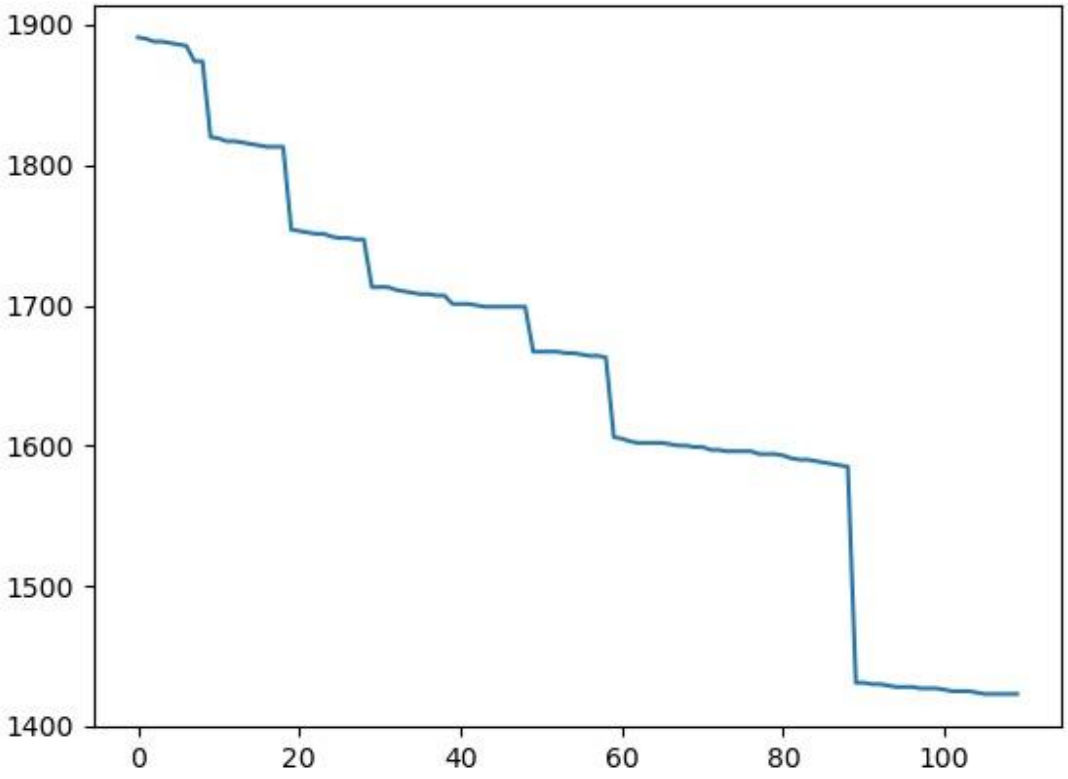


Pork_pasta_hot

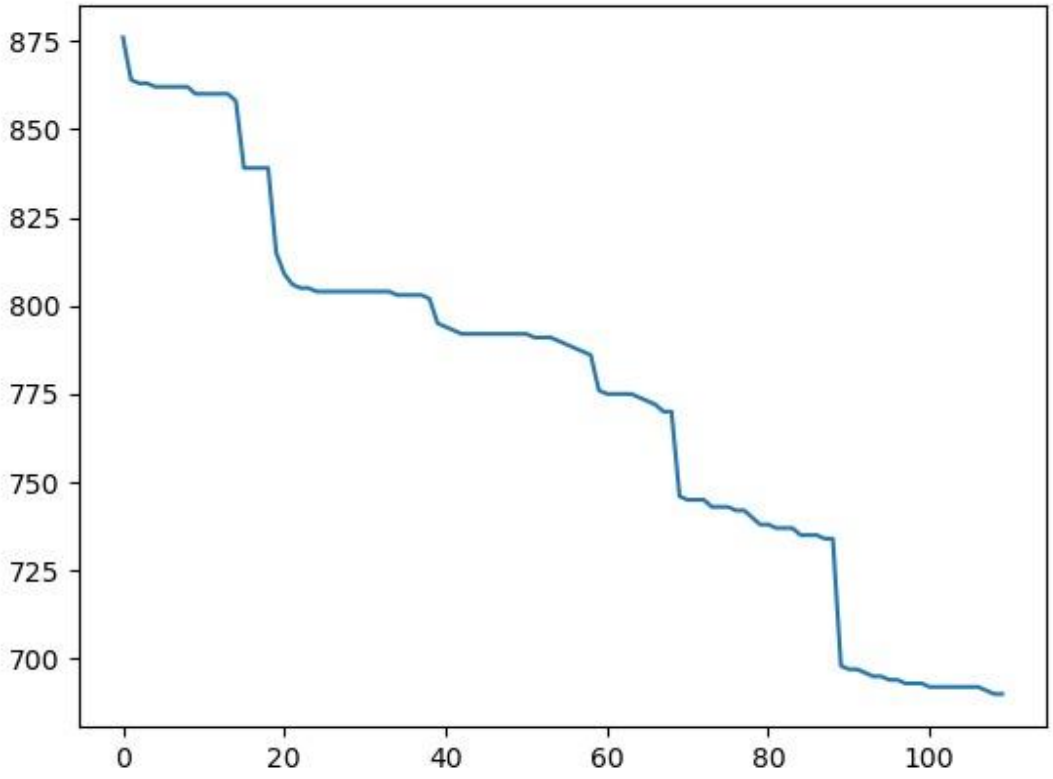


We have also visualized the number of recipes that drops after each iteration of the algorithm.

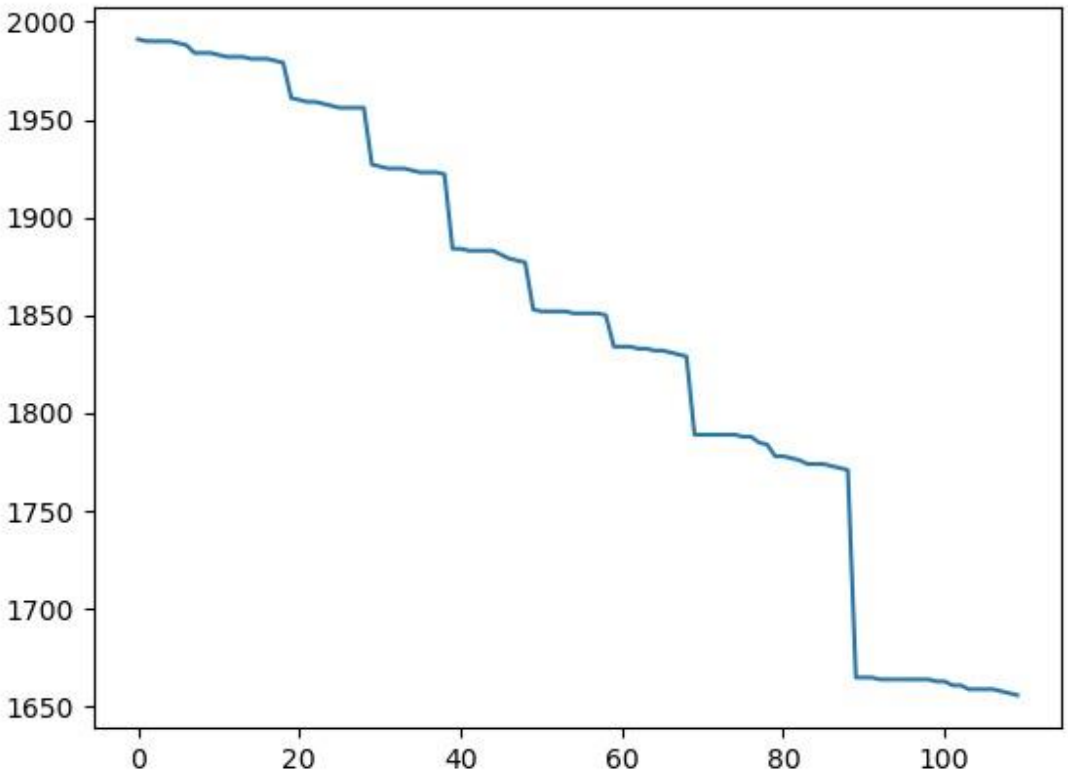
Pork_potato_mild



Beef_rice_spicy



Chicken_bread_mild



Impediments

Learning javascript, react, html and CSS, was not an easy task. And it made us face several problems we could not solve, which forced us to adapt and find other ways to finish building our algorithm. And the lack of debugging in the js environment made our job incredibly hard.

Scraping from the yummly site kept getting blocked. We later found out it the server had a certain threshold for the recipes one could scrape from a given category. So, our data was smaller than what we were expecting, and we believe if we had been able to get all the recipes we wanted, then the amount of recipes deleted at each iteration would have been larger.

Future Work

It's possible to build a machine learning algorithm that will learn the relationships between all the ingredients, that will build a recipe tailored for the user.

Another thing possible is to build a program tailored around fitness and body building. Where the user could insert his desired daily caloric intake or his desired macros(how many grams of protein, carbohydrates and fat) or a combination of both(for example a specific caloric intake number with the amount of protein desired) plus the amount of meals he would like to have. And the program would advise him recipes where the caloric intake would be split between the users meals.

The link to our project: https://abd-khrubi.github.io/DataScienceProject/