Recipe Recommendation System

Table of Contents

Meal Planning Problem

1. Abstract

Have you ever stood in front of the food shelves with your hands holding two packs of frozen items, staring at both of their nutrition labels, running all the calculations in your head but just cannot decide which to put into your cart? Exactly. One of the hardest decisions in life is to decide what to eat for dinner, especially for those who undergo calorie control or have a hectic schedule. Unfortunately, people are programmed as a poor optimizer -- we are easily bogged down in the mud of dilemmas when making decisions. However, even with the aid of a recommender system, we could still struggle in choosing the one among those recommendations. The ordinary recommender system utilizes the technology that only suggests the predicted high-rating items based on your previous choices, ignoring all the other factors that could also contribute to your fondness for the item. A genuine "recommender" that takes all your other requirements into account is few and far between. Therefore, our recipe recommender application aims to resolve this common affliction.

2. Introduction

We have built a recipe recommendation application for two people, helping users to customize their cooking schedule, nutrition requirements, budget control, and ingredients selection. The whole process was a marriage of two techniques: a recommender system (gather data, complete the matrix, and recommend the highest rated recipe for users) and an optimization problem. We first scraped recipe information from Food.com, collecting ingredients, nutritional information, cost, preparing time, and ratings (average rating and ratings from all voters). After a series of data cleansing (e.g. remove null values), the rating matrix underwent a matrix completion process, which will be articulated in section 3.3. This is where the ordinary recommender system ends, but we carried on optimizing the recommendation by setting up an optimization problem for the meal plan and using Python to solve the result with the aid of Pyomo library.

Fig.1 demonstrates the flow of the whole process as well as the structure of this report. To reduce the load in both matrix completion and optimization, we winnow out the essential recipes and users in advance. The first filtering process takes place in data manipulation. Raters that voted for a certain amount of recipes will be chosen: this process dramatically trim down the number of raters in the matrix. The second one is applied after the matrix is complete, followed by several recipes being chosen with the estimated rates above a certain threshold (e.g. rating $\geq$ 3). This step ensures the recommended recipes have certain amounts of qualities and, in the meanwhile, reduce the load of solver.

As for users, we ask them about the available schedule for one week, dislike ingredients, favorite food type, and their body information so that we can compute how many calories they need everyday. We do have to consider if users are willing to have the same dishes in a week, and how many dishes do they want to cook in a day. In order to make a true recommendation system, we want to collect information from users as much as possible. After users provide the information, we will offer a five-day meal schedule containing what day they could cook and what dishes they could have for each day.

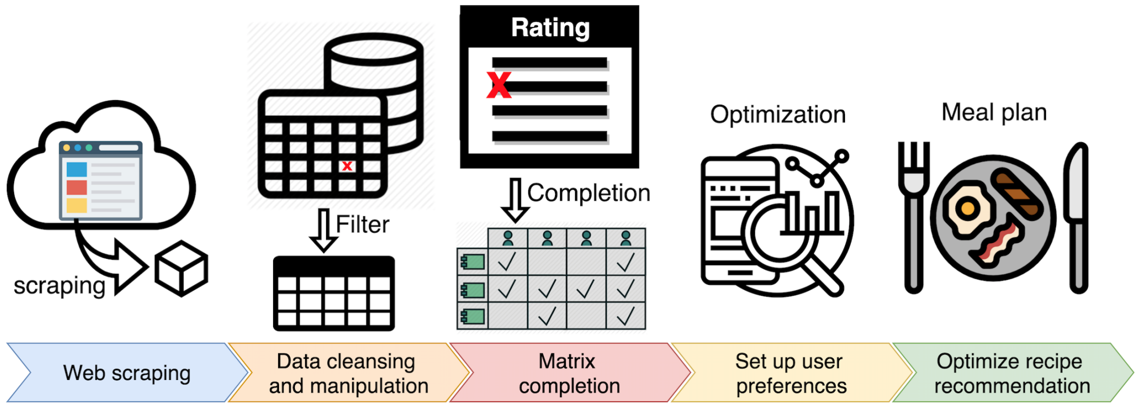


Figure 1 Flowchart of recipe recommendation application

3. Data Sources and processing

3.1. Web Scraping

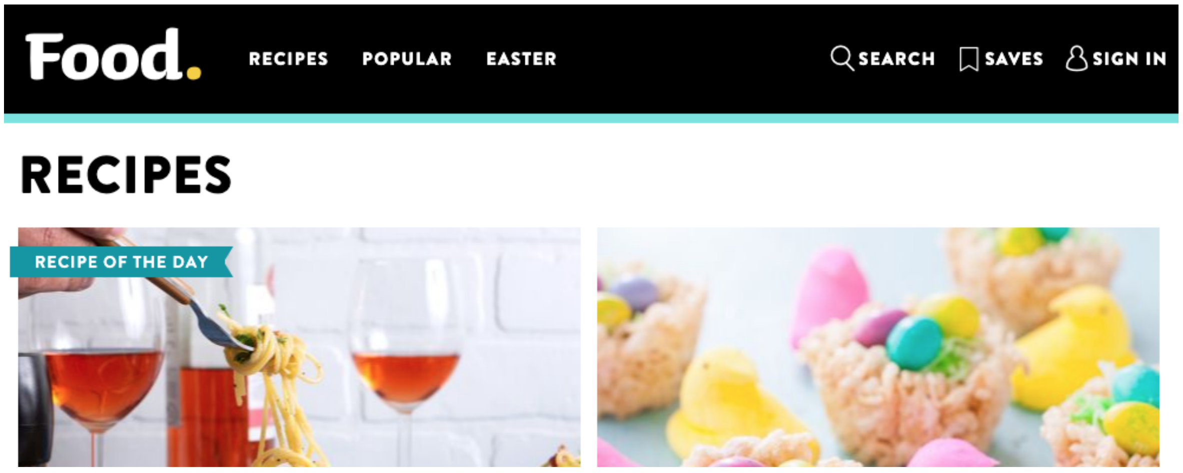
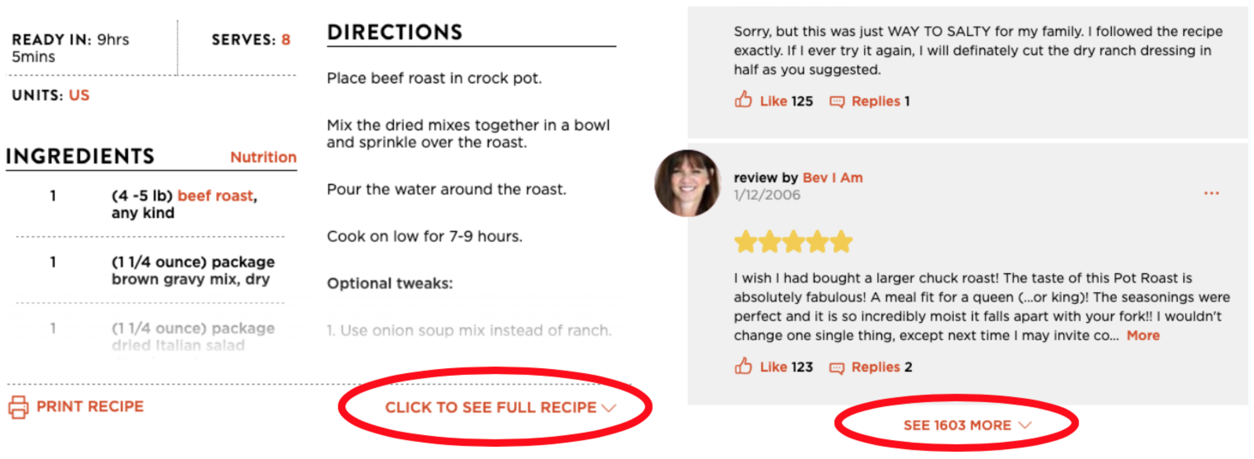


Figure 2: Food.com website

"Food.com", as shown in **Fig.2**, is the recipe website that we use to scrape all the data sources. This website provides lots of common foods of various ingredients and has a whopping 16 million visitors every month which can give us access to an extensive recipe database.

We scrape the website by using "Selenium", which is primarily for automating web applications for testing purposes, but it is certainly not limited to just that. It allows us to open a browser of our choice and perform tasks as a human being. Compared to the method of "BeautifulSoup", by using Selenium, we can click the specific button to let the web page show the entire information of the recipe. Therefore, we can collect the data of ingredients, nutrition of foods and all the personal ratings.



READY IN: 9hrs 5mins | SERVES: 8

UNITS: US

INGREDIENTS Nutrition

1 (4 -5 lb) beef roast, any kind

1 (1 1/4 ounce) package brown gravy mix, dry

1 (1 1/4 ounce) package dried Italian salad

PRINT RECIPE

DIRECTIONS

Place beef roast in crock pot.

Mix the dried mixes together in a bowl and sprinkle over the roast.

Pour the water around the roast.

Cook on low for 7-9 hours.

Optional tweaks:

1. Use onion soup mix instead of ranch.

CLICK TO SEE FULL RECIPE

Sorry, but this was just WAY TO SALTY for my family. I followed the recipe exactly. If I ever try it again, I will definately cut the dry ranch dressing in half as you suggested.

Like 125 Replies 1

review by Bev I Am
1/12/2006

I wish I had bought a larger chuck roast! The taste of this Pot Roast is absolutely fabulous! A meal fit for a queen (...or king)! The seasonings were perfect and it is so incredibly moist it falls apart with your fork!! I wouldn't change one single thing, except next time I may invite co... More

Like 123 Replies 2

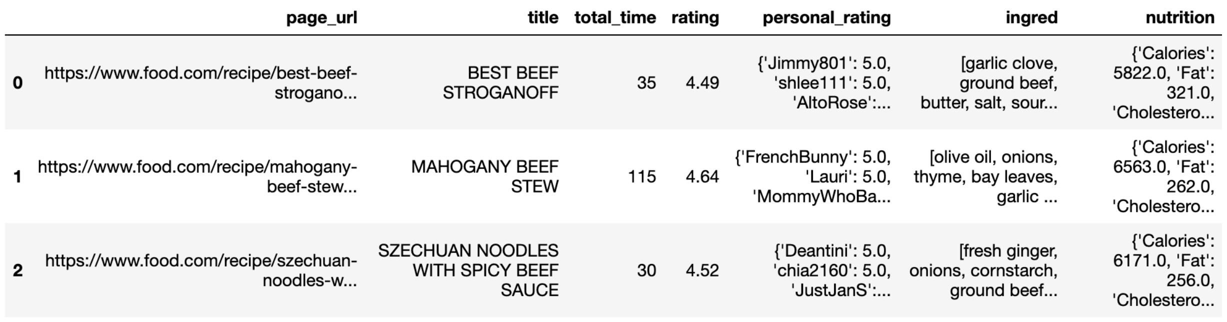
SEE 1603 MORE

Figure 3: Buttons in the webpage

In addition, we search for recipes by different types of main ingredients and collect the recipe data in a categorical way. For instance, we type in some keywords respectively such as chicken, beef and vegetarian which is helpful for us to categorize the scraped recipe. This is how we classified the various types of recipes after scraping the data from "Food.com".

3.2. Data Manipulation

After Web scraping, we will acquire the URL, title, total time, total rating score, ingredients, nutrition, and whole personal ratings of each recipe. We use a dictionary as the data structure for nutrition and personal ratings, and using arrays as the data structure to store ingredients. The following picture is the result of web scraping.



	page_url	title	total_time	rating	personal_rating	ingred	nutrition
0	https://www.food.com/recipe/best-beef-strogano...	BEST BEEF STROGANOFF	35	4.49	{'Jimmy801': 5.0, 'shlee111': 5.0, 'AltoRose':...	[garlic clove, ground beef, butter, salt, sour...	{'Calories': 5822.0, 'Fat': 321.0, 'Cholestero...
1	https://www.food.com/recipe/mahogany-beef-stew...	MAHOGANY BEEF STEW	115	4.64	{'FrenchBunny': 5.0, 'Lauri': 5.0, 'MommyWhoBa...	[olive oil, onions, thyme, bay leaves, garlic ...	{'Calories': 6563.0, 'Fat': 262.0, 'Cholestero...
2	https://www.food.com/recipe/szechuan-noodles-w...	SZECHUAN NOODLES WITH SPICY BEEF SAUCE	30	4.52	{'Deantini': 5.0, 'chia2160': 5.0, 'JustJanS':...	[fresh ginger, onions, cornstarch, ground beef...	{'Calories': 6171.0, 'Fat': 256.0, 'Cholestero...

Figure 4: The result of web scraping

For the scraping result, there are some places we need to specify. Firstly, In the ingredient part, we use "set" data structure temporarily to store the data for fear of scraping duplicate ingredients. Then, we convert into a "list" data structure so that those can serialize in JSON format files. Moreover, the reason why we scrape the URL of pages as well is that we can give the users not only the title of the recipe but also the picture of it when they get the result of recipes that we recommend.

Additionally, we drop all the rows with NaN data. Eventually, for the personal rating part, we only consider the recipes with at least 30 reviews. More importantly, in order to make the result of simulation better, we remove the reviewers who vote twice and below as shown in **Fig.4**. And due to the removal, some recipes will become NaN in their personal rating column. Therefore, we will have to drop these rows of data again.

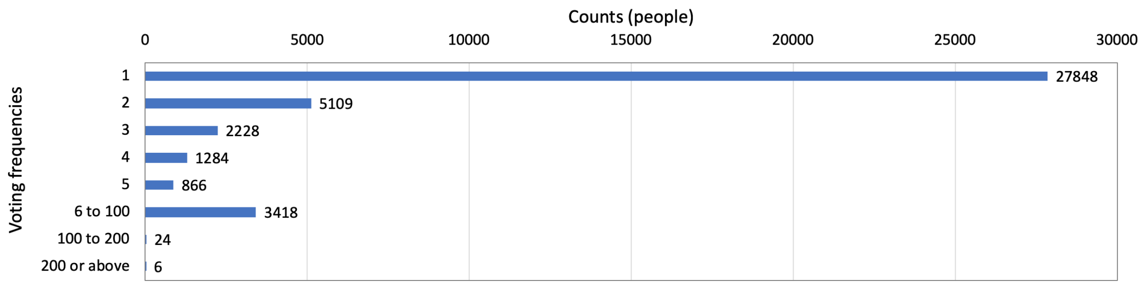


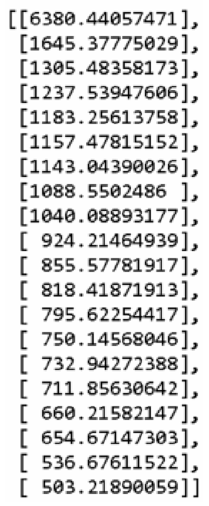
Figure 5: The count of people with different voting frequencies.

3.3. Matrix Completion

We found an open source soft_impute.py online that implements matrix completion by solving a convex optimization program. The program is written as below:

$$f_{\lambda}(Z) := \frac{1}{2}||P_{\Omega}(Z - X)||_{F}^{2} + \lambda||Z||_{*} .$$

One parameter that we are allowed to tune is the number of latent dimensions of the eigenvalues when executing SVD. We can choose a number of eigenvalues (important features) with $\lambda = 0$. **Fig.6 (Left)** shows the outcome when we set the number of eigenvalues to 20. Given that 20 eigenvalues take more than 30 mins to run for 5000 iterations, we decide to run with only 10 eigenvalues, 5000 iterations but 10 replications for each λ. With the regularizer λ, we are able to tune the bias-variance tradeoff by running through a set of λ. The result (**Fig.6** (**Right**)) suggests that $\lambda = 0$has the best performance (lest mean MSE).



```
[[6380.44057471],
 [1645.37775029],
 [1305.48358173],
 [1237.53947606],
 [1183.25613758],
 [1157.47815152],
 [1143.04390026],
 [1088.5502486 ],
 [1040.08893177],
 [ 924.21464939],
 [ 855.57781917],
 [ 818.41871913],
 [ 795.62254417],
 [ 750.14568046],
 [ 732.94272388],
 [ 711.85630642],
 [ 660.21582147],
 [ 654.67147303],
 [ 536.67611522],
 [ 503.21890059]]
```

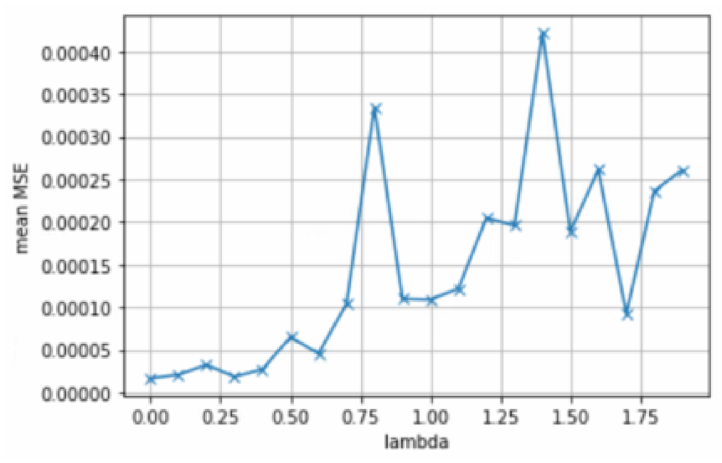
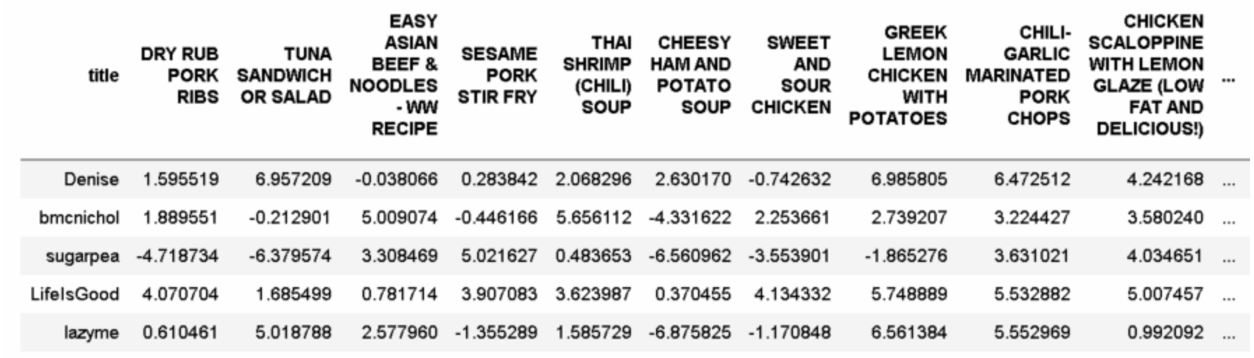


Figure 6: **(Left)** Eigenvalues of the SVD; **(Right)** Bias -variance tradeoff



title	DRY RUB PORK RIBS	TUNA SANDWICH OR SALAD	EASY ASIAN BEEF & NOODLES - WW RECIPE	SESAME PORK STIR FRY	THAI SHRIMP (CHILI) SOUP	CHEESY HAM AND POTATO SOUP	SWEET AND SOUR CHICKEN	GREEK LEMON CHICKEN WITH POTATOES	CHILI-GARLIC MARINATED PORK CHOPS	CHICKEN SCALOPPINE WITH LEMON GLAZE (LOW FAT AND DELICIOUS!)	...
Denise	1.595519	6.957209	-0.038066	0.283842	2.068296	2.630170	-0.742632	6.985805	6.472512	4.242168	...
bmcnichol	1.889551	-0.212901	5.009074	-0.446166	5.656112	-4.331622	2.253661	2.739207	3.224427	3.580240	...
sugarpea	-4.718734	-6.379574	3.308469	5.021627	0.483653	-6.560962	-3.553901	-1.865276	3.631021	4.034651	...
LifeIsGood	4.070704	1.685499	0.781714	3.907083	3.623987	0.370455	4.134332	5.748889	5.532882	5.007457	...
lazyme	0.610461	5.018788	2.577960	-1.355289	1.585729	-6.875825	-1.170848	6.561384	5.552969	0.992092	...

Figure 7 A glance of the completed rating matrix

4. Models and Algorithm

4.1. Variables

4.1.1. System/User Defined Variables:

$\boldsymbol{K}$: total number of recipes

$\boldsymbol{N_{Mi}}$: maximum nutrition requirement for person i

$\boldsymbol{N_{mi}}$: minimum nutrition requirement for person i

$\boldsymbol{N_k}$: nutrition content for recipe k

$\boldsymbol{T_k}$: preparation and cooking time for recipe k

$\boldsymbol{T_{id}}$: total available time person i have on day d

$\boldsymbol{D_M}$: having at most D_M dishes per day

$\boldsymbol{D_m}$: having at least D_m dishes per day

$\boldsymbol{Y_M}$: making at most Y_M dishes per day

$\boldsymbol{I_i}$: Person i's preference for each ingredients (range from 1-5)

$\boldsymbol{I_k}$: ingredient content for each recipe

$\boldsymbol{R_r}$: maximum recipe repeat time

$\boldsymbol{P_d}$: food preserve day, meaning that the food cannot be placed in the frige more than this many days.

$\boldsymbol{B}$: total budget

$\boldsymbol{b_k}$: cost of recipe k

$\boldsymbol{r_{ik}}$: person i's rating on recipe k

4.1.2. Decision Variables:

$\boldsymbol{x_{dk}}$: dish k will be served on day d

$\boldsymbol{y_{idk}}$: person i will **make** dish k on day d

4.2. Model 1

$$\underset{x,\,y}{\text{maximize}} \quad \sum_{d=1}^{5}\sum_{k=1}^{K}(r_{1k}+r_{2k})x_{kd} - \sum_{d=1}^{5}\sum_{k=1}^{K}\sum_{i=1}^{2}(5-d)(T_{id}-T_k y_{idk}) + \sum_{d=1}^{5}\sum_{k=1}^{K}\sum_{i=1}^{2}(I_i - I_k)x_{kd} \tag{1a}$$

$$\text{subject to} \quad \sum_{k=1}^{K} x_{kd} \geq D_m, \qquad \forall d \in \{1,\ldots,5\} \tag{1b}$$

$$\sum_{k=1}^{K} x_{kd} \leq D_M, \qquad \forall d \in \{1,\ldots,5\} \tag{1c}$$

$$\sum_{d=1}^{5}\sum_{k=1}^{K} b_k x_{dk} \leq B \tag{1d}$$

$$\sum_{k=1}^{K} N_k x_{dk} \leq N_{Mi}, \qquad \forall i \in \{1,2\}, \forall d \in \{1,\ldots,5\} \tag{1e}$$

$$\sum_{k=1}^{K} N_k x_{dk} \geq N_{mi}, \qquad \forall i \in \{1,2\}, \forall d \in \{1,\ldots,5\} \tag{1f}$$

$$\sum_{k=1}^{K} T_k y_{idk} \leq T_{id}, \qquad \forall i \in \{1,2\},\ d \in \{1,\ldots,5\} \tag{1g}$$

$$\sum_{k=1}^{K} y_{idk} \leq Y_M, \qquad \forall i \in \{1,2\},\ d \in \{1,\ldots,5\} \tag{1h}$$

$$\sum_{d=1}^{5}\sum_{k=1}^{K} y_{1dk} + y_{2dk} \leq 1, \qquad \forall d \in \{1,\ldots,5\},\ \forall k \in \{1,\ldots,K\} \tag{1i}$$

$$\sum_{d=1}^{5}\sum_{k=1}^{K}((y_{1dk}+y_{2dk}) - x_{dk}) = 0, \tag{1j}$$

$$\sum_{k=1}^{K}((y_{11k}+y_{21k}) - x_{1k}) * x_{1k} = 0, \tag{1k}$$

$$\sum_{k=1}^{K}(\sum_{d=1}^{2}((y_{1dk}+y_{2dk}) - x_{dk})\sum_{d=1}^{2} x_{dk}) = 0, \tag{1l}$$

$$\sum_{k=1}^{K}(\sum_{d=1}^{3}((y_{1dk}+y_{2dk}) - x_{dk})\sum_{d=1}^{3} x_{dk}) = 0, \tag{1m}$$

$$\sum_{k=1}^{K}(\sum_{d=1}^{4}((y_{1dk}+y_{2dk}) - x_{dk})\sum_{d=1}^{4} x_{dk}) = 0, \tag{1n}$$

$$\sum_{k=1}^{K}(\sum_{d=1}^{5}((y_{1dk}+y_{2dk}) - x_{dk})\sum_{d=1}^{5} x_{dk}) = 0, \tag{1o}$$

Explanation:

In order to separate the dish-making schedule from daily meal plan, we introduce two set of decision variables: x_{dk}(assuming that two people will have the same dish on the same day) and y_{idk} (customized cooking schedule for each person).

***Objective Function*:**

The ultimate goal of this optimization problem is to maximize users' fondness of the meal plan. To incorporate with the result coming from the matrix completion, the first term of the objective function inevitably will be the sum of the predicted ratings from both of the users. Another basic goal of our model is to leverage both of their available time to cook the assigned meals as soon as possible, so that they will not starve even when both of them have no time making dinner on some days. A concise and effective way to achieve this is simply penalize the cooking time by timing a reversed "day" factor -- the later you make the dish, the more you are penalized. The final term represents users' preference toward ingredients. On a scale of 1 to 5, for 5 being an extra bonus on that dish, 1 being extra penalty for that dish, and the default is 3.

Constraints:

Constrain 1(b) to 1(h) are the rudimental requirements. The explanations are listed in **table 1** below. 1(i) to 1(k) are the constraints that guaranteed the relationship between $x_{dk}$and y_{idk}. We first need to make sure that those users will not cook the same dish on the same day in that we decide not to sophisticate the problem by calculating the new estimated cooking time for two people making one dish. 1(i) ensures users will cook their own dish. Intuitively, the second thing we need to make sure is the number of dishes they cooked has to be the same as the number of which they ate. 1(j) guarantees the sum will be the same. Last but not least, we need to make sure that all the meals that users will have on a certain day have to be made before or on that day. By multiplying x_{dk}, we can focus on "what they will eat" on that particular day, so $((y_{1dk} + y_{2dk}) - x_{dk}) * x_{dk}$ with a particular d can only be 0 or -1, where 0 means the meal they will eat on day d will certainly be made on that day, and -1 means otherwise. That is to say, the summation of $((y_{1dk} + y_{2dk}) - x_{dk}) * x_{dk}$, meaning all the dishes they will have versus those they will make before and on the day d, has to be 0 at the end of the day.

Label	Constraints	Explanation
1b	$\sum_{k=1}^{K} X_{kd} \geq D_{m}, \quad \forall d$	The total number of dishes they have per day has to be greater than the minimum dishes requirement.
1c	$\sum_{k=1}^{K} X_{kd} \leq D_{\mathrm{M}}, \quad \forall d$	The total number of dishes they have per day has to be less than the maximum dishes requirement.
1d	$\sum_{d=1}^{5} \sum_{k=1}^{K} b_{k} X_{dk} \leq B$	The total cost of the dishes cannot exceed the budget.
1e	$\sum_{k=1}^{K} N_{k} X_{dk} \leq N_{\mathrm{M}i}$	The total nutrition of the dishes should meet the minimum daily nutrition requirement.
1f	$\sum_{k=1}^{K} N_{k} X_{dk} \geq N_{mi}$	The total nutrition of the dishes should meet the maximum daily nutrition requirement.
1g	$\sum_{k=1}^{K} T_{k} y_{i_{dk}} \leq T_{id}$	The total cooking time for a person should not exceed the available time per day.
1h	$\sum_{k=1}^{K} y_{i_{dk}} \leq Y_{m}$	The dishes that users make per day should not exceed their capability.
1i	$y_{1dk} + y_{2dk} \leq 1$	Users will not cook the same dish on the same day.
1j	$\sum_{d=1}^{5} \sum_{k=1}^{K} (y_{1dk} + y_{2dk}) - x_{dk} = 0$	The total number of dishes they will eat is the same as the total number of dishes they will make.
1k ~ 1o	$\sum_{k=`}^{K} (\sum_{d=1}^{D} ((y_{1dk} + y_{2dk}) - x_{dk}) \sum_{d=1}^{D} x_{dk})$, D ∈ {1.. 5}	Make su re every dish they will have each day will be made beforehand.

Table 1: Explanation for constraints in model1.

Drawback:

One critical drawback that deter us from using this model in real-time practice is that its nonlinear constraints (1(k) ~ 1(o)) compromise the efficiency -- it takes hours to solve with a relatively small rating matrix. Therefore, we come up with the second model that the solver can solve within 3 seconds.

4.3. Model 2

$$\underset{x,\,y}{\text{maximize}} \quad \sum_{d=1}^{5}\sum_{k=1}^{K}(r_{1k}+r_{2k})x_{kd} - \sum_{d=1}^{5}\sum_{k=1}^{K}\sum_{i=1}^{2}(5-d)(T_{id}-T_k y_{idk}) + \sum_{d=1}^{5}\sum_{k=1}^{K}\sum_{i=1}^{2}(I_i - I_k)x_{kd} \tag{2a}$$

$$\text{subject to} \quad \sum_{k=1}^{K} x_{kd} \geq D_m, \qquad \forall d \in \{1,\ldots,5\} \tag{2b}$$

$$\sum_{k=1}^{K} x_{kd} \leq D_M, \qquad \forall d \in \{1,\ldots,5\} \tag{2c}$$

$$\sum_{d=1}^{5}\sum_{k=1}^{K} b_k x_{dk} \leq B \tag{2d}$$

$$\sum_{k=1}^{K} N_k x_{dk} \leq N_{Mi}, \qquad \forall i \in \{1,2\}, \forall d \in \{1,\ldots,5\} \tag{2e}$$

$$\sum_{k=1}^{K} N_k x_{dk} \geq N_{mi}, \qquad \forall i \in \{1,2\}, \forall d \in \{1,\ldots,5\} \tag{2f}$$

$$\sum_{k=1}^{K} T_k y_{idk} \leq T_{id}, \qquad \forall i \in \{1,2\},\ d \in \{1,\ldots,5\} \tag{2g}$$

$$\sum_{k=1}^{K} y_{idk} \leq Y_M, \qquad \forall i \in \{1,2\},\ d \in \{1,\ldots,5\} \tag{2h}$$

$$\sum_{d=1}^{5}\sum_{k=1}^{K} y_{1dk} + y_{2dk} \leq 1, \qquad \forall d \in \{1,\ldots,5\},\ \forall k \in \{1,\ldots,K\} \tag{2i}$$

$$\sum_{d=1}^{5}(5-d)(y_{1dk}+y_{2dk}-x_{dk}) \leq P_d, \forall k \in \{1,\ldots,K\} \tag{2j}$$

$$\sum_{d=1}^{5}(5-d)(y_{1dk}+y_{2dk}-x_{dk}) \geq 0, \ \forall k \in \{1,\ldots,K\} \tag{2k}$$

$$\sum_{d=1}^{5} y_{1dk} + y_{2dk} - x_{dk} = 0, \qquad \forall k \in \{1,\ldots,K\} \tag{2l}$$

Explanation:

We replace constraints 1(j) to 1(o) with 2(j) to(2l). 2(j) and 2(l) ensures no dish will be made after the day to serve and no dish will be made before the maximum preserved day. **Table 2** sums up the explanation for the revised constraints.

Label	Constraints	Explanation
2j	$\sum_{d=1}^{5}(5-d)(y_{1dk}+y_{2dk}-x_{dk}) \le P_d, \quad \forall k$	The weight of the dish being made or ate is the reverse count of the day (5-d). This creates enough gradients to calculate the preserved days.
2k	$\sum_{d=1}^{5}(5-d)(y_{1dk}+y_{2dk}-x_{dk}) \ge 0, \quad \forall k$	Greater than or equal to zero means no dish will be served before they make it.
2l	$\sum_{d=1}^{5} y_{1dk}+y_{2dk}-x_{dk} = 0$	Make sure every dish they will have each day will be made beforehand.

Table 2: Explanation for revised constraints in model2.

Drawback:

Constraint 2(j) and 2(k) require looping through all the recipes, which could be unscalable when the number of recipes grows. For this reason, we filter the recipes before solving the program.

5. Analysis and Results

5.1. Test Condition

We simulate two users (the first and the second of the rating matrix), having requirements as **table 3**.

Parameters		Value	Unit	Type
Number of Recipes		1229	--	Count
Schedule[1]	**user0**	[60 30 0 30 60]	minutes per day	List
	user1	[60 0 60 60 0]	minutes per day	List
Nutritions				
Calories	**user0**	[1000 4000]	gram	Range
	user1	[1000 4000]	gram	Range

[1] Schedule: provide a chunk of time for each day instead of a fragmented time schedule.

Fat	**user0**	[0 200]	gram	Range
	user1	[0 200]	gram	Range
Cholesterol	**user0**	[0 200]	gram	Range
	user1	[0 200]	gram	Range
Sodium	**user0**	[1000 4000]	microgram	Range
	user1	[1000 4000]	microgram	Range
Carbohydrate	**user0**	[500 3000]	microgram	Range
	user1	[500 3000]	microgram	Range
Protein	**user0**	[100 600]	gram	Range
	user1	[100 600]	gram	Range
Dishes Served		[1 2]	per day	Range
[2]Maximum Dishes Cooked		3	per day	Count
Recipe Repeat Times		2	--	Count
[3]Ingredients Fondness	**user0**	[3, 3, 3… ,3]	--	List
	user1	[3, 3, 3… ,3]	--	List
Maximum Food Preserved Days		3	day	Count
Total Days		5	day	Count

Table 3: Values of all the parameters

5.2. Results

5.2.1. Single Solution

In **Fig.8**, we can see the optimized meal planning for these two people. The number in the colored bar represents the index of the recipes, and the length of the bar is the estimated total time (cooking + preparing). To visualize the result clearly, we draw the red dotted line to segment each day, 2.5 hours per day (for cooking) in the

[2] Maximum Dishes Cooked: the maximum capacity of recipes a person can cook per day.
[3] Ingredients Fondness: each index in the list represents an ingredient. Scale from 1 to 5, where 1 means least favorite, 5 means most favorite, and 3 means no preference as default.

figure. The top figure tells which person should cook what on which day, and the bottom one shows the dishes they can have for each day.

The result is aligned with our expectations. We can see that

1. The model is trying to have the users make the food as early as possible
2. Every dish is planned to be made before the day they are going to eat.
3. Every dish that is made in advance will be eaten within the maximum persevered day.

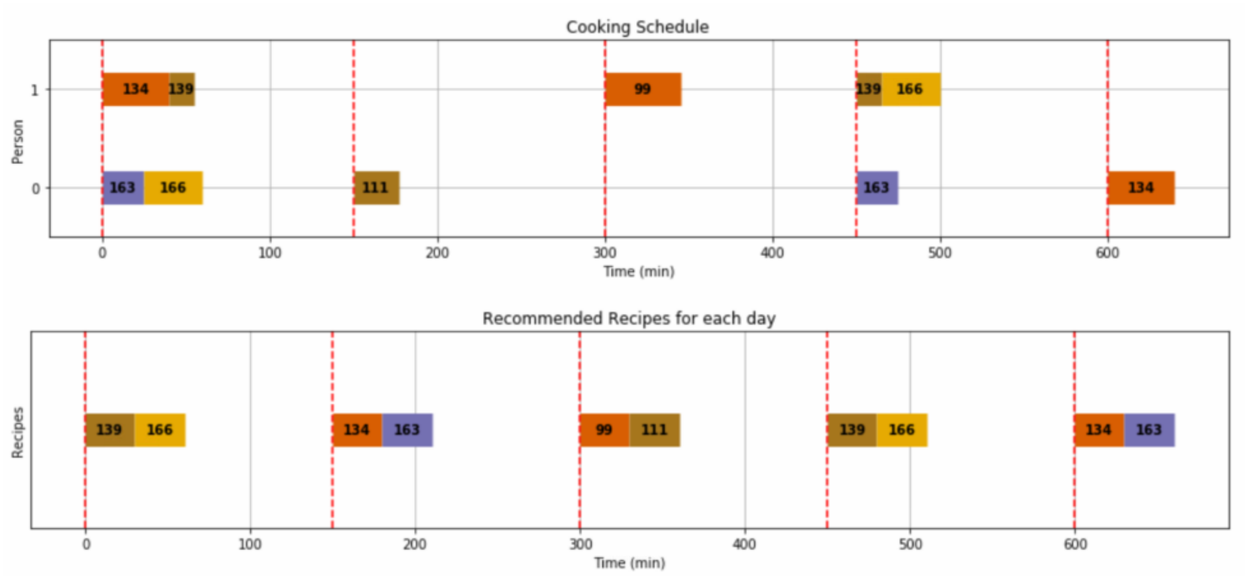


Figure 8: The meal planning schedule outcome with each dish can repeat 2 times max.

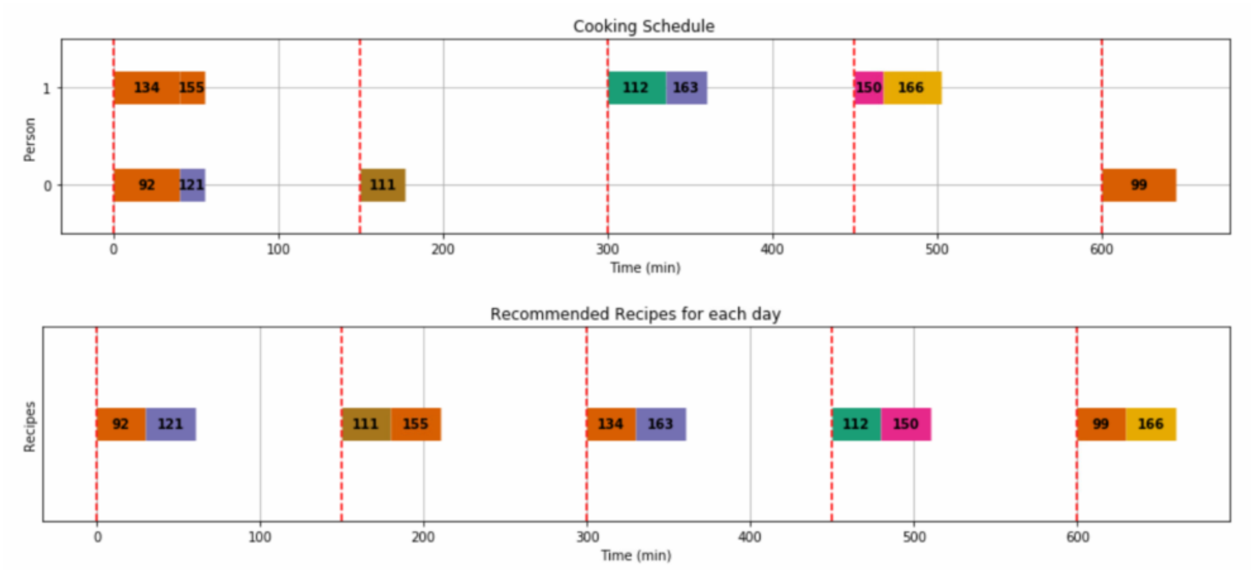


Figure 9: A glance of the outcome with no repeated dish. **Fig.8** and this show that our model is robust enough to handle the original case and the extended case (with repeated dish).

5.2.2. Multiple Optimal Solutions

Since this optimization problem has pure binary decision variables, we can find the other optimal solutions easily. According to [1, 2], we can derive other optimal solutions by plugging in Equation

$$\sum_{j\in O} X_j - \sum_{j\in Z} X_j \leq |O| - 1,$$

where

$$O = \{j \mid X^*_j = 1\},$$
$$Z = \{j \mid X^*_j = 0\},$$

for every time we try to find a new solution (Balas and Heroslow, 1972).

After applying this method, we are able to solve for the other optimal solutions, for this kind of problem usually comes with a set of similar solutions. In **Fig.10**, we observe the solutions are simply the combination of the same set of recipes. However, we still can take advantage of these solutions to pick the most suitable one to suggest to users. For example, we can eliminate solutions that advise having the same dish two days in a row, or remove the critical case (**Fig.11**) that we currently cannot solve with the highlighted constraints in **table 2**. The muli-optima solutions compensate for the weakness that is inherent in the original programming, and thus we can still benefit from the multi-optima solutions even though they are suggesting the same set of recipes.

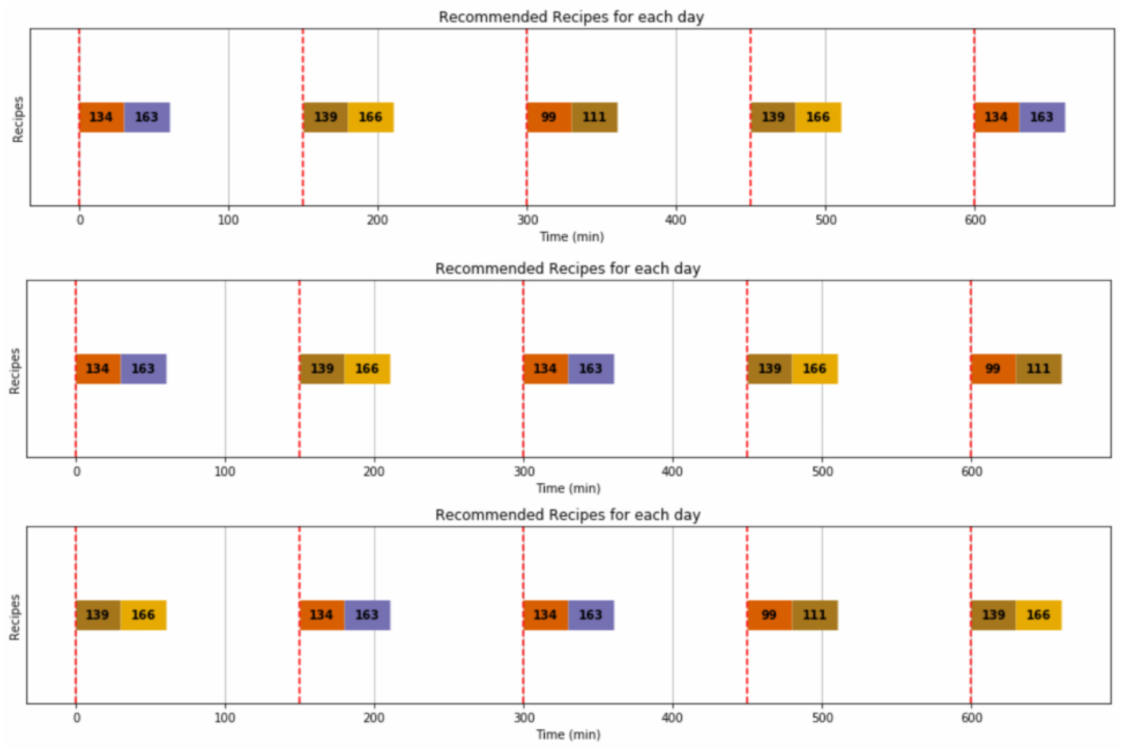


Figure 10 Multi-optima solutions.

In Figure 10, there are a total of 5 solutions in this solution set, but only 3 are listed here for demonstration.

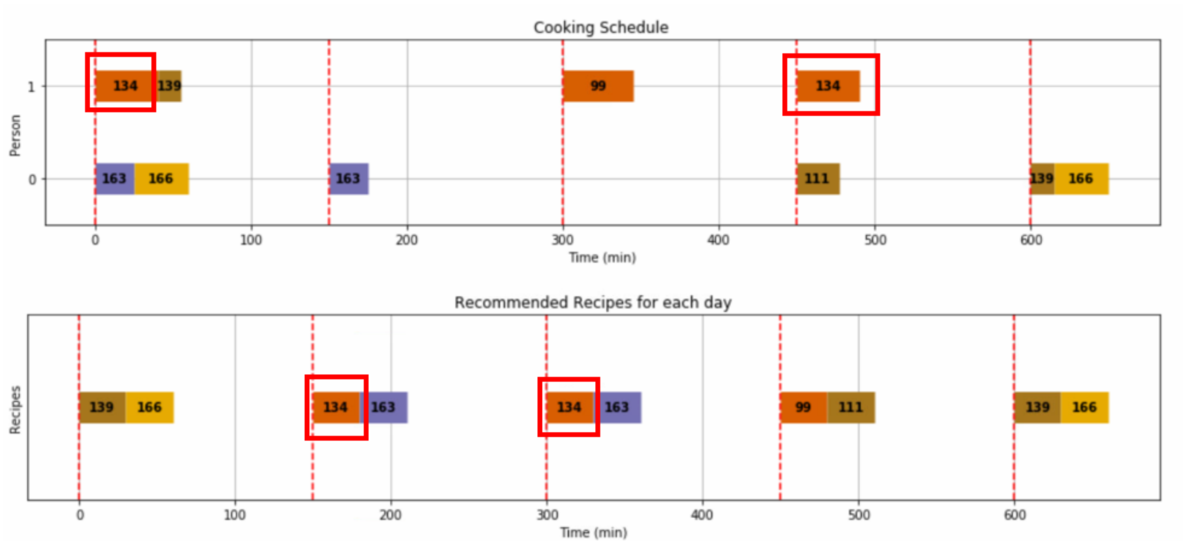


Figure 11 Invalid solution.

In figure 11, recipe #134 is invalid since it is made on Thursday but is served on Wednesday. This is the only case that will fail because we allow the dish to appear twice a week and, unfortunately, 5 - 4 - 3 + 2 = 0.

6. Conclusion and Future work

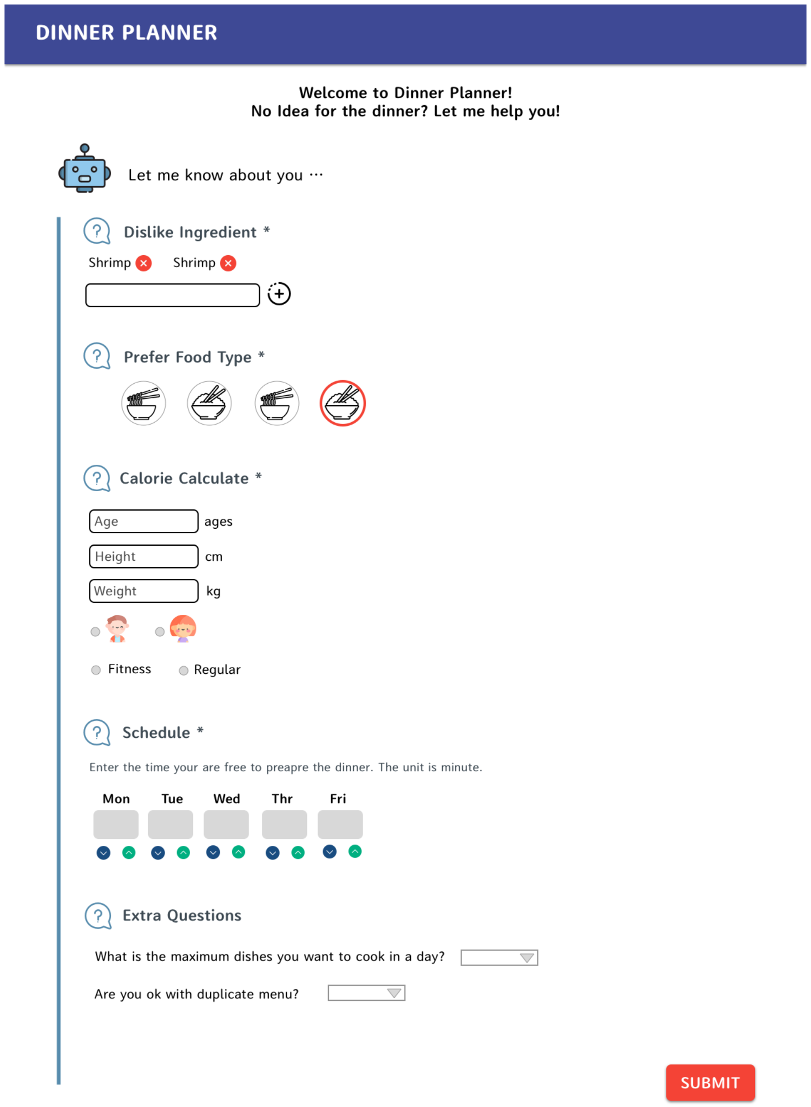
6.1 Conclusion

This project is a combination of three different skills: web scraping, recommender system, and optimization problem modeling. We have learned as much on integrating the skills as on solving puzzles coming from building models and details in improving the efficiency. Luckily, the result turns out to be satisfying, but still exists several flaws that need to be remedied.

One major future work is to fix the problem stated in **Fig.11**. Ever since we separate the “preparing” and “cooking” schedule, how we can build the relationship between these two variables becomes another thorny problem that we need to solve. On top of the two models mentioned in **section 4.2.** and **section 4.3.** , we come up with a third two-stage model that is inspired by stochastic programming with no stochastic variable in the second stage instead. The idea is to first select a set of dishes users are going to have, and then rearrange the dishes in the second stage to meet their schedule. This model is just a few steps away from finished, we will present it in the updated report in future revision.

6.2. Future work - Website develop

In order to let users have better user experience, we decide to deploy our system on the website and application. We will use the Django framework to build the frontend and backend. This is the user interface that we designed, so that users can input their information easily and directly. From the picture, users can easily understand the information we need based on the picture. We want to make our website as easy as possible.


DINNER PLANNER

Welcome to Dinner Planner!
No Idea for the dinner? Let me help you!

Let me know about you …

Dislike Ingredient *
Shrimp Shrimp

Prefer Food Type *

Calorie Calculate *
Age ages
Height cm
Weight kg

Fitness Regular

Schedule *
Enter the time your are free to preapre the dinner. The unit is minute.
Mon Tue Wed Thr Fri

Extra Questions
What is the maximum dishes you want to cook in a day?
Are you ok with duplicate menu?

SUBMIT

Figure 12 Setting page of the demo website

6.3. Future work - Mobile/ Desktop application development

Nothing can be more regretful when having a great thought in mind but fails to implement it. Other than building a website, we are attempting to build a desktop application for demonstration (hopefully we can finish it in this semester) first, and then transplant it to a mobile one. The platform we choose is C# WPF for desktop application (**Fig.13**), and Xamarin (C# based, cross-platform) for mobile application.

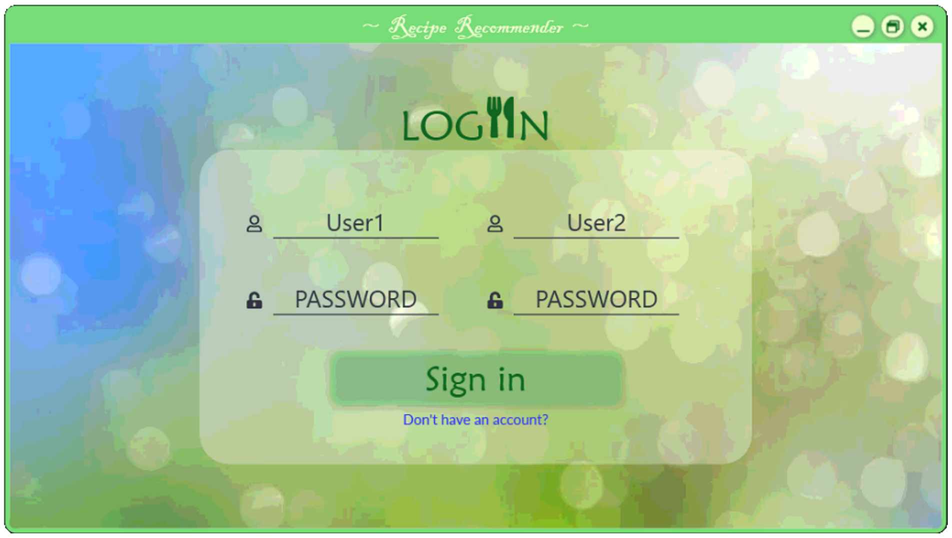


Figure 13 Desktop application (DEMO)

7. Reference

[1] A.C. Trapp and R.A. Konrad, "Finding diverse optima and near-optima to binary integer programs," *IIE Transactions*, Vol. 47, 11 2015, pp. 1300-1312.

[2] J.D. Camm, "How to Influence and Improve Decisions Through Optimization Models," *Recent Advances in Optimization and Modeling of Contemporary Problems*, INFORMS, 2018, pp. 1-19.