PROJECT 2

MEAL PLAN PROBLEM

GROUP 6

Qianran Ma (7798937244)

Yanan Zhou (4366462176) Presenter

Wanning Li (5898212888)

Yuxuan Cheng (1278632602)

Content

Project Outline

1. Background
2. Basic Model
3. Data Preparation & Problem Solving
4. Creative Ideas

01 ▸ Background

Over the hills and far away, Teletubbies come to play!

One day, Tinky-Winky, Dipsy, Laa-Laa and Po decided to make a meal plan for the coming week.

- Each of them has different eating preference
- Each of them need to obtain enough nutrition
- Each of them is allergic to some food
- Each of them has classes on different days
- They have limited money for ingredients
- ……

02 ▸ Basic Model

1. Assumptions
2. Decision Variables & Objective Function
3. Constraints

1 Assumptions

Assumptions:

- We only make plan for 5 days (Monday - Friday)
- We only make plan for lunch and dinner every day (so we have 10 meals in total)
- The cooking time of each meal is the same: 10:00-12:00 for lunch; 16:00-18:00 for dinner
- The cooking time includes shopping time
- Everyone can acquire the same amount of nutrition from the same dish in a meal
- Everyone needs the same amount of nutrition every day
- A dish costs the same amount of money every time
- The store has all the things we want

Unify the footer letters' meaning:

- i stands for Teletubbies: $i \in \{1, 2, 3, 4\}$, $\mathrm{n}_s = 4$
- j stands for dishes: $j \in \{1, 2, \dots, 240\}$, $\mathrm{n}_d = 240$
- k stands for meals: $k \in \{1, 2, \dots, 10\}$, $\mathrm{n}_m = 10$

2 Decision Variables & Objective Function

Decision Variable: $d(n_j \times n_m)$: $d_{jk} = 1$ *means dish j is cooked in meal k;*
$d_{jk} = 0$ *means dish j is not cooked in meal k.*

$y(n_s \times n_m)$: $y_{ik} = 1$ *means Teletubby i will cook meal k;*
$y_{ik} = 0$ *means Teletubby i will not cook meal k.*

Parameters: $r(n_s \times n_d)$: r_{ij} *means the rating score of dish j from Teletubby i*

Objective Function:

$$\sum_{i=1}^{n_s} \sum_{k=1}^{n_m} \sum_{j=1}^{n_d} r_{ij} d_{jk}$$

maximize the sum of all Teletubbies' ratings of all the selected dishes

$$\underbrace{\begin{bmatrix} r_{11} & r_{12} & \cdots & r_{1,240} \\ r_{21} & r_{22} & \cdots & r_{2,240} \\ r_{31} & r_{32} & \cdots & r_{3,240} \\ r_{41} & r_{42} & \cdots & r_{4,240} \end{bmatrix}}_{r} \times \underbrace{\begin{bmatrix} d_{11} & \cdots & d_{1,10} \\ d_{21} & \cdots & d_{2,10} \\ \vdots & \ddots & \vdots \\ d_{239,1} & \cdots & d_{239,10} \\ d_{240,1} & \cdots & d_{240,10} \end{bmatrix}}_{d} = \underbrace{\begin{bmatrix} s_{11} & s_{12} & \cdots & s_{1,10} \\ s_{21} & s_{22} & \cdots & s_{2,10} \\ s_{31} & s_{32} & \cdots & s_{3,10} \\ s_{41} & s_{42} & \cdots & s_{4,10} \end{bmatrix}}_{s}$$

s_{ik}: *The total rating score of meal k from Teletubby i*

$sum(s_{ik})$: *The total rating score of all meals from all Teletubbies*

3 Constraints

The number of dishes of each meal

- We cook no more than 3 dishes in each meal

$$\sum_{j=1}^{n_d} d_{jk} \leq 3\,, \qquad for\ \forall k \in \{1, 2, \dots, 10\}$$

Cooking frequency of each dish

- We eat the same dish at most twice

$$\sum_{k=1}^{n_m} d_{jk} \leq 2\,, \qquad for\ \forall j \in \{1, 2, \dots, 240\}$$

$$\begin{bmatrix} d_{11} & \cdots & d_{1,10} \\ d_{21} & \cdots & d_{2,10} \\ \vdots & \ddots & \vdots \\ d_{239,1} & \cdots & d_{239,10} \\ d_{240,1} & \cdots & d_{240,10} \end{bmatrix}$$

- Each column of d represents for a meal.
- The number of 1 in each colomn represents the number of dishes in each meal

3 Constraints

The nutrition requirements

Nutrition Vectors:

N^s $(1 \times n_d \ vector)$: N_j^s means the **Sodium** level of dish j

N^c $(1 \times n_d \ vector)$: N_j^c means the **Calory** level of dish j

N^f $(1 \times n_d \ vector)$: N_j^f means the **Fat** level of dish j

N^p $(1 \times n_d \ vector)$: N_j^p means the **Protein** level of dish j

Nutrition Requirements:

U^s , L^s: means the everyday **upper bound** and **lower bound** of **Sodium** for all Teletubbies

U^c , L^c: means the everyday **upper bound** and **lower bound** of **Calory** for all Teletubbies

U^f , L^f: means the everyday **upper bound** and **lower bound** of **Fat** for all Teletubbies

U^p , L^p: means the everyday **upper bound** and **lower bound** of **Protein** for all Teletubbies

3 Constraints

The nutrition requirements

Nutrition Constraints:

- Make sure the sum of nutrition of everyday meals satisfy the requirement
- The 1st & 2nd meals belong to Monday ;
- The 3rd & 4th meals belong to Tuesday…

$$[N_1^s \quad N_2^s \quad \cdots \quad N_{240}^s] \times \begin{bmatrix} d_{1k} \\ d_{2k} \\ \vdots \\ d_{240k} \end{bmatrix} = Total\ Sodium\ of\ meal\ k$$

$$\sum_{j=1}^{n_d} (N_j^s d_{jk} + N_j^s d_{j,k+1}) \le U^s, \qquad \sum_{j=1}^{n_d} (N_j^s d_{jk} + N_j^s d_{j,k+1}) \ge L^s \qquad for\ \forall k \in \{1,3,5,7,9\}$$

$$\sum_{j=1}^{n_d} (N_j^c d_{jk} + N_j^c d_{j,k+1}) \le U^c, \qquad \sum_{j=1}^{n_d} (N_j^c d_{jk} + N_j^c d_{j,k+1}) \ge L^c \qquad for\ \forall k \in \{1,3,5,7,9\}$$

$$\sum_{j=1}^{n_d} (N_j^f d_{jk} + N_j^f d_{j,k+1}) \le U^f, \qquad \sum_{j=1}^{n_d} (N_j^f d_{jk} + N_j^f d_{j,k+1}) \ge L^f \qquad for\ \forall k \in \{1,3,5,7,9\}$$

$$\sum_{j=1}^{n_d} (N_j^p d_{jk} + N_j^p d_{j,k+1}) \le U^p, \qquad \sum_{j=1}^{n_d} (N_j^p d_{jk} + N_j^p d_{j,k+1}) \ge L^p \qquad for\ \forall k \in \{1,3,5,7,9\}$$

3 Constraints

The limitation of budget

Price Vectors: $p\ (1\times n_d\ vector)$: $p_j\ means\ the\ cost\ of\ dish\ j$

Budget: P: $the\ budget\ of\ all\ the\ meals$

Budget Constraints:

- Make sure the sum of total cost of all the meals does not exceed budget
- Each column of d reprenents a meal
- $p\times d_{\bullet k}$ is the total cost of meal k

$$\sum_{k=1}^{n_m}\sum_{j=1}^{n_d} p_j d_{jk} \le P$$

$$[p_1 \quad p_2 \quad \cdots \quad p_{240}] \times \begin{bmatrix} d_{1k} \\ d_{2k} \\ \vdots \\ d_{240k} \end{bmatrix} = Total\ price\ of\ meal\ k$$

3 Constraints

The limitation of allergy

Allergy Matrix:

$a\ (n_d \times n_d\ matix)$: $All\ the\ elements\ not\ on\ diagonal\ are\ 0\ (a_{ij} = 0\ \ \forall i \neq j)$;
$a_{jj} = 0\ if\ there\ exists\ someone\ who\ is\ allergic\ to\ dish\ j$;
$a_{jj} = 1\ if\ no\ one\ is\ allergic\ to\ dish\ j$;

Allergy Constraints:

- Make sure no one is allergic to the selected dishes
- Each column of d reprenents a meal

$$\underbrace{\begin{bmatrix} a_{11} & \cdots & 0 \\ 0 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & 0 \\ 0 & \cdots & a_{240,240} \end{bmatrix}}_{a} \times \underbrace{\begin{bmatrix} d_{1k} \\ d_{2k} \\ \vdots \\ d_{240k} \end{bmatrix}}_{d_{\bullet k}} = \underbrace{\begin{bmatrix} a_{11} \times d_{1k} \\ a_{22} \times d_{2k} \\ \vdots \\ a_{240,240} \times d_{240,k} \end{bmatrix}}_{d_{\bullet k}'}$$

If no one is allergic to the selected dishes: $d_{\bullet k}' = d_{\bullet k}$

Thus, the constraints should be: $a_{jj} d_{jk} = d_{jk} \qquad \forall k \in \{1, 2, \dots, 10\}, \forall j \in \{1, 2, \dots, 240\}$

3 Constraints

Time availability

Availability Matrix: $Z(n_s \times n_k)$:

Z_{ik} *means whether Teletubby* i *is available during the cooking time of meal* k

Cooking Times Constraints:

- **The Teletubbies who cook at this meal must available**

  $y_{ik} \leq Z_{ik} \qquad \forall i \in \{1, 2, 3, 4\} \quad \forall k \in \{1, 2, \dots, 10\}$

- **Each meal will be cooked by at most 2 Teletubbies**

  $$\sum_{i=1}^{n_s} y_{ik} \leq 2, \qquad \forall k \in \{1, 2, \dots, 10\}$$

- **Each Teletubby will cook 3~4 times a week**

  $$\sum_{k=1}^{n_m} y_{ik} \leq 4, \qquad \sum_{k=1}^{n_m} y_{ik} \geq 3, \qquad \forall i \in \{1, 2, 3, 4\}$$

3 Constraints

Cooking ability

Cooking Ability Matrix: $M(n_s \times n_d)$: M_{ij} *means whether the Teletubby i can cook dish j*

$M_{ij} = 1$ --- *Teletubby i can cook dish j*

$M_{ij} = 0$ --- *Teletubby i can not cook dish j*

Whether can cook

$$\underbrace{\begin{bmatrix} M_{11} & M_{21} & M_{31} & M_{41} \\ M_{12} & M_{22} & M_{32} & M_{42} \\ \vdots & \vdots & \vdots & \vdots \\ M_{1,240} & M_{2,240} & M_{3,240} & M_{4,240} \end{bmatrix}}_{M^T} \times \underbrace{\begin{bmatrix} y_{11} & y_{12} & \cdots & y_{1,240} \\ y_{21} & y_{22} & \cdots & y_{2,240} \\ y_{31} & y_{32} & \cdots & y_{3,240} \\ y_{41} & y_{42} & \cdots & y_{4,240} \end{bmatrix}}_{y} = \underbrace{\begin{bmatrix} D_{11} & D_{1,2} & \cdots & D_{1,10} \\ D_{21} & D_{22} & \cdots & D_{2,10} \\ \vdots & \vdots & \ddots & \vdots \\ D_{240,1} & D_{240,1} & \cdots & D_{240,10} \end{bmatrix}}_{D}$$

Whether cook

D_{jk}: *The number of Teletubbies who:* { *are available at the cooking time of meal k* / *and can cook dish j in meal k*

3 Constraints

Cooking ability

Cooking Ability Constraints:

- **Each dish selected in each meal should be the dish that the cooking Teletubbies can cook**

$$d_{jk} \leq D_{jk}$$

If a dish is selected in meal k,

then there must be at least 1 cooking Teletubby can cook this dish

$$Plug\ in \quad D_{jk} = \sum_{i=1}^{n_s} M_{ij} y_{ik}$$

$$d_{jk} \leq \sum_{i=1}^{n_s} M_{ij} y_{ik} \qquad \forall k \in \{1, 2, \dots, 10\}, \forall j \in \{1, 2, \dots, 240\}$$

3 Constraints

Potential Problems:

- **What if there is no one available during the cooking time of meal k?**
- **What if the dishes the cooking Teletubbies can cook can not meet the nutrition requirement?**

3 Constraints

Potential Problems:

- What if there is no one available during the cooking time of meal k?
- What if the dishes the cooking Teletubbies can cook can not meet the nutrition requirement?

Eat Outside!

3 The Optimization Problem

$$\textit{Maximize} \quad \sum_{i=1}^{n_s}\sum_{k=1}^{n_m}\sum_{j=1}^{n_d} r_{ij} d_{jk} \qquad \textit{Subject to}$$

$$\sum_{j=1}^{n_d} d_{jk} \le 3 \qquad for\ \forall k \in \{1,2,\dots,10\}$$

$$\sum_{k=1}^{n_m} d_{jk} \le 2 \qquad for\ \forall j \in \{1,2,\dots,240\}$$

$$\sum_{j=1}^{n_d} (N_j^s d_{jk} + N_j^s d_{j,k+1}) \le U^s, \sum_{j=1}^{n_d} (N_j^s d_{jk} + N_j^s d_{j,k+1}) \ge L^s \qquad for\ \forall k \in \{1,3,5,7,9\}$$

$$\sum_{j=1}^{n_d} (N_j^c d_{jk} + N_j^c d_{j,k+1}) \le U^c, \sum_{j=1}^{n_d} (N_j^c d_{jk} + N_j^c d_{j,k+1}) \ge L^c \qquad for\ \forall k \in \{1,3,5,7,9\}$$

$$\sum_{j=1}^{n_d} (N_j^f d_{jk} + N_j^f d_{j,k+1}) \le U^f, \sum_{j=1}^{n_d} (N_j^f d_{jk} + N_j^f d_{j,k+1}) \ge L^f \qquad for\ \forall k \in \{1,3,5,7,9\}$$

$$\sum_{j=1}^{n_d} (N_j^p d_{jk} + N_j^p d_{j,k+1}) \le U^p, \sum_{j=1}^{n_d} (N_j^p d_{jk} + N_j^p d_{j,k+1}) \ge L^p \qquad for\ \forall k \in \{1,3,5,7,9\}$$

$$\sum_{k=1}^{n_m}\sum_{j=1}^{n_d} p_j d_{jk} \le P$$

$$a_{jj} d_{jk} = d_{jk} \qquad \forall k \in \{1,2,\dots,10\}, \forall j \in \{1,2,\dots,240\}$$

$$y_{ik} \le Z_{ik} \qquad \forall i \in \{1,2,3,4\}, \forall k \in \{1,2,\dots,10\}$$

$$\sum_{i=1}^{n_s} y_{ik} \le 2, \qquad \forall k \in \{1,2,\dots,10\}$$

$$\sum_{k=1}^{n_m} y_{ik} \le 4, \qquad \sum_{k=1}^{n_m} y_{ik} \ge 3, \qquad \forall i \in \{1,2,3,4\}$$

$$d_{jk} \le \sum_{i=1}^{n_s} M_{ij} y_{ik} \qquad \forall k \in \{1,2,\dots,10\}, \forall j \in \{1,2,\dots,240\}$$

$$y_{ik} \in \{0,1\} \qquad \forall i \in \{1,2,3,4\}, \forall k \in \{1,2,\dots,10\}$$

$$d_{jk} \in \{0,1\} \qquad \forall j \in \{1,2,\dots,240\}, \forall k \in \{1,2,\dots,10\}$$

03 Data Preparation & Problem Solving

1. Rating Matrix
2. Nutrition Requirement
3. Price and Budget
4. Allergy Matrix
5. Availability Matrix
6. Cooking Ability Matrix
7. Code and Result

1 Rating Matrix --- Data Preparation

Raw Data from result.json

	title	rating	calories	sodium	fat	protein	personal_rating
0	Curried Lentil, Tomato, and Coconut Soup	5.0	437.0	667.0	28.0	13.0	[[rhaeredekop from Winnipeg, MB , 5], [lizgold...
1	Roasted Butternut Squash with Herb Oil and Goa...	4.5	175.0	576.0	9.0	4.0	[[ilyssa2 from NY, NY , 5], [dory92064 from Sa...
2	Pumpkin Muffins	4.0	364.0	183.0	11.0	6.0	[[mtnmeye , 3], [greenstein.rebecca9820 from N...
3	Chopped Salad with Shallot Vinaigrette, Feta, ...	5.0	170.0	413.0	13.0	6.0	[[brushjl from solon, oh , 5], [tatyana_poirie...
4	Grain Salad with Olives and Whole-Lemon Vinaig...	3.5	330.0	483.0	19.0	8.0	[[auntwebbie from Allen, TX , 5], [krf from Be...
...	...	...	...	...	...	...	...
293	One-Pot Curried Cauliflower with Couscous and ...	4.5	606.0	1365.0	15.0	27.0	[[mkopke from Toronto, Canada , 4], [akraemer1...
294	Acorn Squash with Kale and Sausage	4.5	NaN	NaN	NaN	NaN	[[sminkoff , 3], [emdean from Madison, WI , 5]...
295	Hummus Dinner Bowls with Spiced Ground Beef an...	4.5	329.0	327.0	26.0	20.0	[[bradley2 from Foodietown, ,PA , 4], [chaurie...
296	Autumn Kale Salad	5.0	287.0	329.0	22.0	4.0	[[dottie60 from Boston , 5], [zeta from Victor...
297	Pumpkin Icebox Pie With Snickerdoodle Crust	4.5	3597.0	1869.0	223.0	33.0	[[Lois33 from EGR, Michigan , 5], [mccoyj25 fr...

Each row represents a single dish, which is rated by 0 or more people

"allrate" Table

	index	name	rate	title	rating	calories	sodium	fat	protein
0	0	rhaeredekop from Winnipeg	5.0	Curried Lentil, Tomato, and Coconut Soup	5.0	437.0	667.0	28.0	13.0
1	0	lizgoldsmith1960 from Lincoln	5.0	Curried Lentil, Tomato, and Coconut Soup	5.0	437.0	667.0	28.0	13.0
2	0	bmaybeegeorge from Canada	5.0	Curried Lentil, Tomato, and Coconut Soup	5.0	437.0	667.0	28.0	13.0
3	0	adventurousz from Boulder	3.0	Curried Lentil, Tomato, and Coconut Soup	5.0	437.0	667.0	28.0	13.0
4	0	epaul77 from Mississippi	5.0	Curried Lentil, Tomato, and Coconut Soup	5.0	437.0	667.0	28.0	13.0
...	...	...	...	...	...	...	...	...	...
2919	297	ctripp from Colorado Springs	3.0	Pumpkin Icebox Pie With Snickerdoodle Crust	4.5	3597.0	1869.0	223.0	33.0
2920	297	mlrn853 from Philadelphia area	5.0	Pumpkin Icebox Pie With Snickerdoodle Crust	4.5	3597.0	1869.0	223.0	33.0
2921	297	waughbash158 from Edmond	1.0	Pumpkin Icebox Pie With Snickerdoodle Crust	4.5	3597.0	1869.0	223.0	33.0
2922	297	waughbash158 from Edmond	1.0	Pumpkin Icebox Pie With Snickerdoodle Crust	4.5	3597.0	1869.0	223.0	33.0
2923	297	mynwt from Annapolis	4.0	Pumpkin Icebox Pie With Snickerdoodle Crust	4.5	3597.0	1869.0	223.0	33.0

Each row represents single rate record from one person to one dish

1 Rating Matrix --- Data Preparation

```
col=pd.unique(C.name)
ind=pd.unique(C['index'])
M=pd.DataFrame(0,
               columns = col,
               index = ind)
M=M.astype(float)
for i in C.index:
    IND = C['index'][i]
    NAME = C['name'][i]
    M[NAME][IND] = C['rate'][i]
```

Matrix M (Mij)

title	Curried Lentil, Tomato, and Coconut Soup	Roasted Butternut Squash with Herb Oil and Goat Cheese	Pumpkin Muffins	Chopped Salad with Shallot Vinaigrette, Feta, and Dill	Grain Salad with Olives and Whole-Lemon Vinaigrette	Chilled Coconut Corn Soup	Vietnamese-Style Spaghetti Squash "Noodle" Bowls with Skirt Steak	Roasted Squash with Mint and Toasted Pumpkin Seeds	Butternut Squash Steaks with Brown Butter–Sage Sauce	Freeform Chicken Meatballs with Carrots and Yogurt Sauce	...	Twice-Roasted Squash with Parmesan Butter and Grains	Kale Salad with Butternut Squash, Pomegranate, and Pumpkin Seeds	Kabocha Squash Pilaf with Coconut	Chile-Marinated Pork with Vietnamese Brussels Sprouts
rhaeredekop from Winnipeg	5	0	0	0	0	0	0	0	0	0	...	0	0	0	0
lizgoldsmith1960 from Lincoln	5	0	0	0	0	0	0	0	0	0	...	0	0	0	0
bmaybeegeorge from Canada	5	0	0	0	0	0	0	0	0	0	...	0	0	0	0
adventurousz from Boulder	3	0	0	0	0	0	0	0	0	0	...	0	0	0	0
epaul77 from Mississippi	5	0	0	0	0	0	0	0	0	0	...	0	0	0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
kderevyanik from Southlake	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0
jcp76 from Pittsburgh	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0
mlrn853 from Philadelphia area	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0
waughbash158 from Edmond	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0
mynwt from Annapolis	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0

Mij means the rate of dish j by person i:

- **If person i rated dish j, Mij∈ {1,2,3,4,5}**
- **If person i did not rate dish j, Mij = 0**

1 Rating Matrix --- Initialize Personal Rating

Each person rate 24 dishes (10%)

Unnamed: 0	Curried Lentil, Tomato, and Coconut Soup	Roasted Butternut Squash with Herb Oil and Goat Cheese	Pumpkin Muffins	Chopped Salad with Shallot Vinaigrette, Feta, and Dill	Grain Salad with Olives and Whole-Lemon Vinaigrette	Chilled Coconut Corn Soup	Vietnamese-Style Spaghetti Squash __oodle_ Bowls with Skirt Steak	Roasted Squash with Mint and Toasted Pumpkin Seeds	Butternut Squash Steaks with Brown Butter__age Sauce	...	Twice-Roasted Squash with Parmesan Butter and Grains	Kale Salad with Butternut Squash, Pomegranate, and Pumpkin Seeds	Kabocha Squash Pilaf with Coconut	Chile-Marinated Pork with Vietnamese Brussels Sprouts	White Chicken Chili
Tinky-Winky	0	0	0	0	0	0	0	4	0	...	0	1	0	0	0
Dipsy	0	0	0	0	0	0	0	0	0	...	0	0	2	0	0
Laa-Laa	0	0	0	2	0	0	0	0	3	...	0	0	0	0	0
Po	0	0	0	0	0	0	0	3	0	...	0	0	0	0	0

Shape: (4, 240)

1 Rating Matrix --- Algorithm Selection

Sparse Matrix:

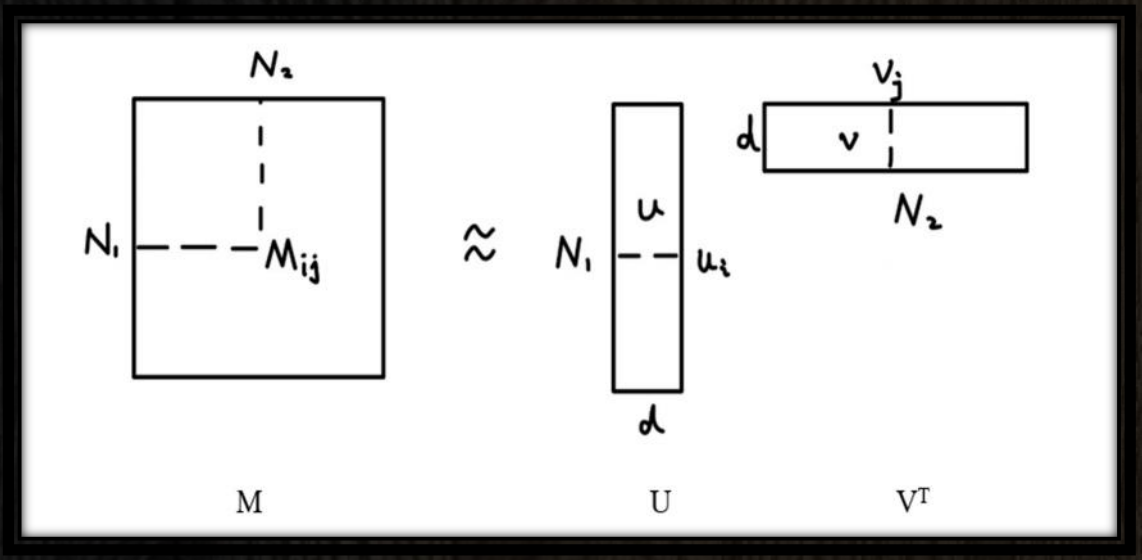
- No one rate all dishes
- No dishes receive ratings from everyone

Similarity:

- Many people with similar tastes
- Many dishes with similar attributes

Rating Matrix --- Algorithm Selection

Low-rank Factorization



$$M = UV^T$$

$$u_i = (u_{i1}, u_{i2}, \cdots, u_{id}) \ \textit{Features of the user } i$$

$$v_j = (v_{j1}, v_{j2}, \cdots, v_{jd}) \ \textit{Features of the dish } j$$

$$d \ll \min\{N_1, N_2\}$$

$$M_{ij} \approx u_i v_j^T = u_{i1}v_{j1} + u_{i2}v_{j2} + \cdots + u_{id}v_{jd}$$

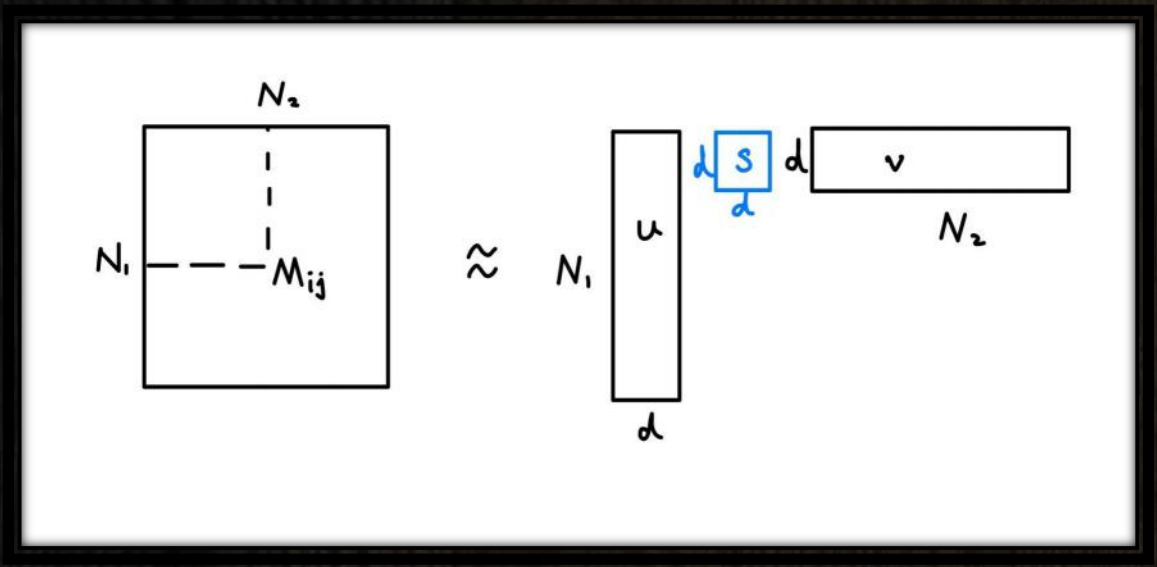
Capture the essential features of people and dishes:

- People: age / gender / country
- Dishes: type / ingredients

Serve as a predictor

Rating Matrix --- Algorithm Selection

Singular Value Decomposition



$$M = USV^T$$
$$U^TU = I \quad V^TV = I$$
$$S_{ii} \geq 0 \quad S_{11} \geq S_{22} \geq \cdots \geq S_{dd}$$
$$d = rank(M)$$
$$M_{ij} \approx u_i v_j^T = S_{11}u_{i1}v_{j1} + S_{22}u_{i2}v_{j2} + \cdots + S_{dd}u_{id}v_{jd}$$

Difference: Matrix S can be treated as "weight"

- SVD has "different weights" for each feature
- Low-rank has "equal weight" for each feature

Choice: Low-rank Factorization

- SVD needs every value (dense matrix), but there are lots of missing values
- Low-rank can handle missing values

1 Rating Matrix --- Low-rank Factorization

Objective Function

$$min_{u_i,v_j} \boxed{\sum_{(i,j)\in\Omega} \frac{\lambda}{2}||M_{ij} - u_i^T v_j||^2} + \boxed{\sum_{i=1}^{N_1} \frac{\mu}{2}||u_i||^2 + \sum_{j=1}^{N_2} \frac{\mu}{2}||v_j||^2} \triangleq L$$

make predicted rating as close as possible to the real rating

Regularization to avoid overfitting

$\Omega = \{(i,j): M_{ij} \text{ is measured}$
$(i,j) \in \Omega \text{ if user } i \text{ has rated dish } j$
$\Omega_{u_i} \text{ be the index set of dishes rated by user } i$
$\Omega_{v_j} \text{ be the index set of users who rated dish } j$

Partial Derivative:

$$D_{u_i} L = \sum_{j\in\Omega_{u_i}} \lambda(M_{ij} - u_i^T v_j)v_j - \mu u_i = 0 \quad \forall i = 1,2,\cdots,N_1$$

$$D_{v_j} L = \sum_{i\in\Omega_{v_j}} \lambda(M_{ij} - u_i^T v_j)u_i - \mu v_j = 0 \quad \forall j = 1,2,\cdots,N_2$$

$$u_i = \left(\frac{\mu}{\lambda} I + \sum_{j\in\Omega_{u_i}} v_j v_j^T\right)^{-1} \left(\sum_{j\in\Omega_{u_i}} M_{ij} v_j\right) \quad \forall i = 1,2,\cdots,N_1$$

$$v_j = \left(\frac{\mu}{\lambda} I + \sum_{i\in\Omega_{v_j}} u_i u_i^T\right)^{-1} \left(\sum_{i\in\Omega_{v_j}} M_{ij} u_i\right) \quad \forall j = 1,2,\cdots,N_2$$

Rating Matrix --- Low-rank Factorization

Low-rank Factorization Algorithm

1. $Initialize\ v_j \sim N\left(0, \lambda^{-1} I\right), u_i\ as\ vector\ 0.0$

2. $For\ each\ iteration$:

$for\ i = 1, 2, \cdots, N_1$:

$$u_i = \left(\frac{\mu}{\lambda} I + \sum_{j \in \Omega_{u_i}} v_j v_j^T\right)^{-1} \left(\sum_{j \in \Omega_{u_i}} M_{ij} v_j\right)$$

$for\ j = 1, 2, \cdots, N_2$:

$$v_j = \left(\frac{\mu}{\lambda} I + \sum_{i \in \Omega_{v_j}} u_i u_i^T\right)^{-1} \left(\sum_{i \in \Omega_{v_j}} M_{ij} u_i\right)$$

3. $Until\ converge$

$M_{ij} = u_i^T v_j$ $Round\ it\ to\ the\ closest\ rating$

Parameters to be chosen:

- $coeffieient\ \lambda$
- $coefficient\ \mu$
- $matrix\ shape\ d$
- $iteration\ round\ K$

1 Rating Matrix --- Code and Prediction

Initialize U and V

Initialize $v_j \sim N(0, \lambda^{-1} I)$

```
def initialize_v(lambd, N2, d):
    '''
    lambd: lambda from function
    N2: number of dishes
    d: shape
    '''
    # initialize v with normal distribution
    # shape v: (N2, d)
    mean = [0]*d
    std = [1/lambd]*d
    np.random.seed(1)
    v0 = np.array([np.random.normal(loc=mean, scale=std) for _ in range(N2)])
    return v0
```

Initialize u_i *as vector* 0.0

```
def initialize_u(N1, d):
    '''
    N1: number of user
    d: shape
    '''
    # initialize u as 0.0
    # shape u: (N1, d)
    u0 = np.array([[0.0 for _ in range(d)] for _ in range(N1)])
    return u0
```

1 Rating Matrix --- Code and Prediction

Update U and V in Each Iteration

$$u_i = \left(\frac{\mu}{\lambda} I + \sum_{j \in \Omega_{u_i}} v_j v_j^T\right)^{-1} \left(\sum_{j \in \Omega_{u_i}} M_{ij} v_j\right) \quad \forall i = 1, 2, \cdots, N_1$$

```python
def coordinate_ui(i, mu, lambd, d, v, M):
    '''
    i: row index of u
    mu, lambd: from function
    d: shape
    v: the second part of low rank factorization
    M: rating matrix
    '''
    # find the idx where the rating of movie from user i is not missing
    idx = np.where(M[i] != 0)
    # select the related part from v and M
    v = v[idx].T
    M = M[i, idx].reshape(-1, 1)
    # calculate new ui based on the function
    first_item = mu/lambd * np.identity(d) + np.dot(v, v.T)
    second_item = np.dot(v, M)
    ui = np.dot(np.linalg.inv(first_item), second_item)
    return ui.T
```

$$v_j = \left(\frac{\mu}{\lambda} I + \sum_{i \in \Omega_{v_j}} u_i u_i^T\right)^{-1} \left(\sum_{i \in \Omega_{v_j}} M_{ij} u_i\right) \quad \forall j = 1, 2, \cdots, N_2$$

```python
def coordinate_vj(j, mu, lambd, d, u, M):
    '''
    j: row index of v
    mu, lambd: from function
    d: shape
    u: the first part of low rank factorization
    M: rating matrix
    '''
    # find the idx where the rating of movie j is not missing
    M1 = M.T
    idx = np.where(M1[j] != 0)
    # select the related part from u and M
    u = u[idx].T
    M = M[idx][:, j].reshape(-1, 1)
    # calculate new vj based on the function
    first_item = mu/lambd * np.identity(d) + np.dot(u, u.T)
    second_item = np.dot(u, M)
    vj = np.dot(np.linalg.inv(first_item), second_item)
    return vj.T
```

1 Rating Matrix --- Code and Prediction

Calculate the Objective Value

$$min_{u_i,v_j} \sum_{(i,j)\in\Omega} \frac{\lambda}{2}||M_{ij} - u_i^T v_j||^2 + \sum_{i=1}^{N_1} \frac{\mu}{2}||u_i||^2 + \sum_{j=1}^{N_2} \frac{\mu}{2}||v_j||^2 \triangleq L$$

```
def objective(lambd, mu, M, u, v, N1, N2):
    # the function has three parts, calculate individually
    # find the index where the rating is not missing
    idx = np.where(M != 0)

    first_part = 0
    for k in range(len(idx[0])):
        ii = idx[0][k]
        jj = idx[1][k]
        Mij = M[ii][jj]
        ui = u[ii]
        vj = v[jj]
        first_part += np.power(Mij-np.dot(ui, vj.T), 2) * (lambd/2)

    second_part = 0
    for i in range(N1):
        second_part += np.power(np.linalg.norm(u[i], 2), 2) * (mu/2)

    third_part = 0
    for j in range(N2):
        third_part += np.power(np.linalg.norm(v[j], 2), 2) * (mu/2)

    obj = first_part + second_part + third_part
    return obj
```

1 Rating Matrix --- Code and Prediction

Coordinate Descent Algorithm

```
def coordinate_alg(lambd, mu, M, K, d):
    '''
    lambd, mu: set by us
    M: rating matrix generated based on dataset and group members' rating
    K: total number of coordinate descent iteration
    '''
    # the shape of u and v
    # u: (N1, d), v: (N2, d) -> M = u*(v.T) -> M: (N1, N2)
    N1 = M.shape[0]
    N2 = M.shape[1]
    #d = np.linalg.matrix_rank(M)
    #d = 5

    # initialize v and u
    v = initialize_v(lambd, N2, d)
    u = initialize_u(N1, d)

    obj_l = []
    # coordinate descent algorithm
    for k in range(K):
        # update u based on v
        for i in range(N1):
            u[i] = coordinate_ui(i, mu, lambd, d, v, M)
        # update v based on u
        for j in range(N2):
            v[j] = coordinate_vj(j, mu, lambd, d, u, M)
        # calculate new objective value
        obj = objective(lambd, mu, M, u, v, N1, N2)
        obj_l.append(obj)

        #list.append(obj)
        # log process
        if k % 10 == 0:
            print(f'Iteration:', k)
            print(obj)
            print('----------------------')
        k += 1

    print(f'Final objective value:', obj)
    return u, v, obj_l
```

1 Rating Matrix --- Code and Prediction

Get the Predicted Result

```
def get_rating(u, v, M):
    '''
    calculate predicted M of 4 group members = u*(v.T)
    '''
    # shape: (4, N2)
    # rating of 4 group members of all dishes
    N1 = M.shape[0]
    # the last 4 rows are group members' rating
    M_predicted = np.dot(u[N1-4:], v.T)
    # refine the result: round to int, project the range between 1-5
    M_predicted = np.round(M_predicted)
    M_predicted = np.where(M_predicted < 1, 1, M_predicted)
    M_predicted = np.where(M_predicted > 5, 5, M_predicted)
    return M_predicted


def update_rating(df_member, M_predicted):
    '''
    update the missing rating to predicted value
    '''
    df_member_new = df_member.copy()
    for i in range(df_member.shape[0]):
        for j in range(df_member.shape[1]):
            # if the dish is not rated by group member, update to predicted rating
            if df_member.iloc[i, j] == 0:
                df_member_new.iloc[i, j] = M_predicted[i][j]
    return df_member_new
```

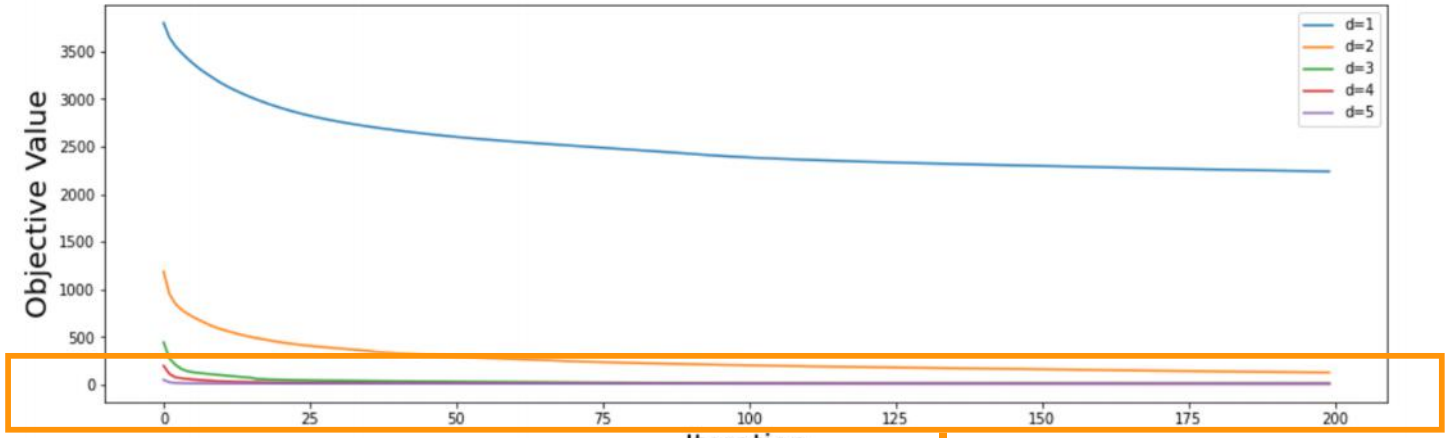
Rating Matrix --- Result

Comparison between Different d

lambda=0.999, mu=0.001, K=200

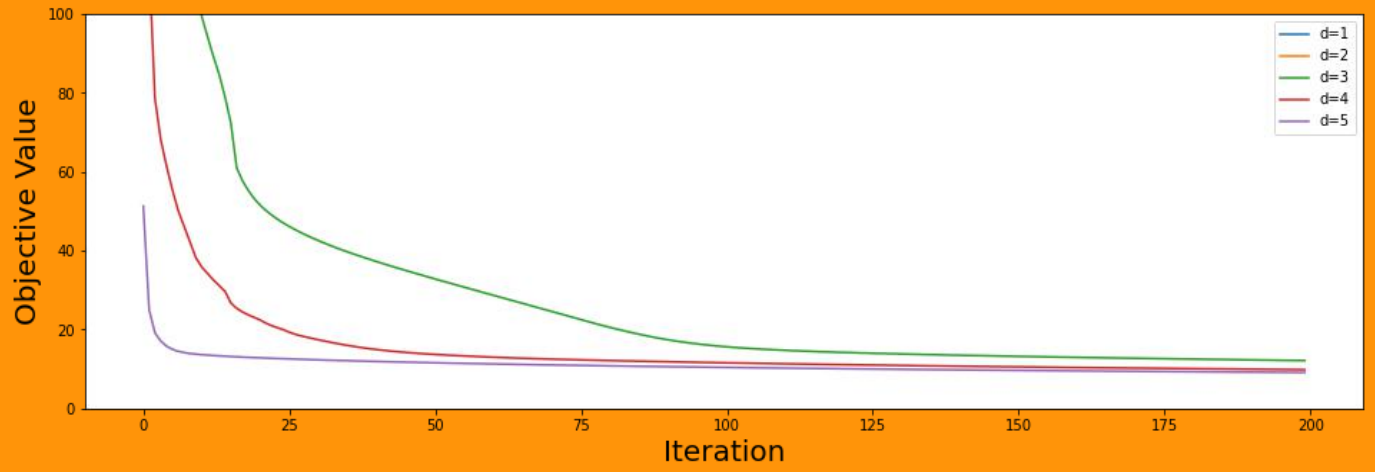
	Objective Value	Running Time
d=1	2239.36	54 seconds
d=2	128.48	1 minute and 22 seconds
d=3	12.14	1 minute and 5 seconds
d=4	9.95	1 minute and 18 seconds
d=5	9.11	1 minute and 25 seconds

1 Rating Matrix --- Result



The result of d=1 is bad

Zoom in: Limit the y axis to [0, 100]



The result of d=5 is the best:

- Although d=5 has slightly longer running time, it converges faster
- With early-stopping condition, it will be the best choice

Choose d=5

Rating Matrix --- Result

Initial sparse rating

Unnamed: 0	Curried Lentil, Tomato, and Coconut Soup	Roasted Butternut Squash with Herb Oil and Goat Cheese	Pumpkin Muffins	Chopped Salad with Shallot Vinaigrette, Feta, and Dill	Grain Salad with Olives and Whole-Lemon Vinaigrette	Chilled Coconut Corn Soup	Vietnamese-Style Spaghetti Squash __oodle_ Bowls with Skirt Steak	Roasted Squash with Mint and Toasted Pumpkin Seeds	Butternut Squash Steaks with Brown Butter__age Sauce	...	Twice-Roasted Squash with Parmesan Butter and Grains	Kale Salad with Butternut Squash, Pomegranate, and Pumpkin Seeds	Kabocha Squash Pilaf with Coconut	Chile-Marinated Pork with Vietnamese Brussels Sprouts	White Chicken Chili
Tinky-Winky	0	0	0	0	0	0	0	4	0	...	0	1	0	0	0
Dipsy	0	0	0	0	0	0	0	0	0	...	0	0	2	0	0
Laa-Laa	0	0	0	2	0	0	0	0	3	...	0	0	0	0	0
Po	0	0	0	0	0	0	0	3	0	...	0	0	0	0	0

Predicted rating

Unnamed: 0	Curried Lentil, Tomato, and Coconut Soup	Roasted Butternut Squash with Herb Oil and Goat Cheese	Pumpkin Muffins	Chopped Salad with Shallot Vinaigrette, Feta, and Dill	Grain Salad with Olives and Whole-Lemon Vinaigrette	Chilled Coconut Corn Soup	Vietnamese-Style Spaghetti Squash __oodle_ Bowls with Skirt Steak	Roasted Squash with Mint and Toasted Pumpkin Seeds	Butternut Squash Steaks with Brown Butter__age Sauce	...	Twice-Roasted Squash with Parmesan Butter and Grains	Kale Salad with Butternut Squash, Pomegranate, and Pumpkin Seeds	Kabocha Squash Pilaf with Coconut	Chile-Marinated Pork with Vietnamese Brussels Sprouts	White Chicken Chili
Tinky-Winky	1	5	1	2	3	1	1	4	4	...	1	1	3	1	1
Dipsy	1	4	1	4	2	2	1	4	2	...	1	3	2	1	1
Laa-Laa	1	2	1	2	3	1	1	4	3	...	1	1	2	1	2
Po	1	1	1	1	1	1	1	3	1	...	1	1	1	1	1

2 Nutrition Requirement

Collect N^s, N^c, N^f, N^p from the dataset

	index	title	calories	sodium	fat	protein
0	0	Curried Lentil, Tomato, and Coconut Soup	437.0	667.0	28.0	13.0
1	1	Roasted Butternut Squash with Herb Oil and Goa...	175.0	576.0	9.0	4.0
2	2	Pumpkin Muffins	364.0	183.0	11.0	6.0
3	3	Chopped Salad with Shallot Vinaigrette, Feta, ...	170.0	413.0	13.0	6.0
4	4	Grain Salad with Olives and Whole-Lemon Vinaig...	330.0	483.0	19.0	8.0
...	...	...	...	...	...	...
264	292	White Chicken Chili	534.0	968.0	14.0	45.0
265	293	One-Pot Curried Cauliflower with Couscous and ...	606.0	1365.0	15.0	27.0
267	295	Hummus Dinner Bowls with Spiced Ground Beef an...	329.0	327.0	26.0	20.0
268	296	Autumn Kale Salad	287.0	329.0	22.0	4.0
269	297	Pumpkin Icebox Pie With Snickerdoodle Crust	3597.0	1869.0	223.0	33.0

240 rows × 6 columns

Collect (U^s, L^s), (U^c, L^c), (U^f, L^f), (L^p, L^p) from the website: https://health.gov/

Table E3.1.A4. Nutritional goals for each age/sex group used in assessing adequacy of USDA Food Patterns at various calorie levels

	Source of goal*	child 1 - 3	female 4 - 8	male 4 - 8	female 9--13	male 9--13	female 14-18	male 14-18	female 19-30	male 19-30	female 31-50	male 31-50	female 51+	male 51+
Calorie level(s) assessed		1000	1200	1400, 1600	1600	1800	1800	2200, 2800, 3200	2000	2400, 2600, 3000	1800	2200	1600	2000
Macronutrients														
Protein, g	RDA	13	19	19	34	34	46	52	46	56	46	56	46	56
Protein, % kcal	AMDR	5--20	10--30	10--30	10--30	10--30	10--30	10--30	10--35	10--35	10--35	10--35	10--35	10--35
Carbohydrate, g	RDA	130	130	130	130	130	130	130	130	130	130	130	130	130
Carbohydrate, %kcal	AMDR	45--65	45--65	45--65	45--65	45--65	45--65	45--65	45--65	45--65	45--65	45--65	45--65	45--65
Dietary Fiber, g	14g/1000kcal	14	16.8	19.6	22.4	25.2	25.2	30.8	28	33.6	25.2	30.8	22.4	28
Total fat, %kcal	AMDR	30-40	25-35	25-35	25-35	25-35	25-35	25-35	20-35	20-35	20-35	20-35	20-35	20-35
Saturated fat, %kcal	DG	10%	10%	10%	10%	10%	10%	10%	10%	10%	10%	10%	10%	10%
Linoleic acid, g	AI	7	10	10	10	12	11	16	12	17	12	17	11	14
Linolenic acid, g	AI	0.7	0.9	0.9	1	1.2	1.1	1.6	1.1	1.6	1.1	1.6	1.1	1.6
Minerals														
Calcium, mg	RDA	700	1000	1000	1300	1300	1300	1300	1000	1000	1000	1000	1200	1200
Iron, mg	RDA	7	10	10	8	8	15	11	18	8	18	8	8	8
Magnesium, mg	RDA	80	130	130	240	240	360	410	310	400	320	420	320	420
Phosphorus, mg	RDA	460	500	500	1250	1250	1250	1250	700	700	700	700	700	700
Potassium, mg	AI	3000	3800	3800	4500	4500	4700	4700	4700	4700	4700	4700	4700	4700
Sodium, mg	UL	1500	1900	1900	2200	2200	2300	2300	2300	2300	2300	2300	2300	2300
Zinc, mg	RDA	3	5	5	8	8	9	11	8	11	8	11	8	11
Copper, mg	RDA	0.34	0.44	0.44	0.7	0.7	0.89	0.89	0.9	0.9	0.9	0.9	0.9	0.9

3 Price and Budget

Price vector p :

- **Divided all the dishes into 5 types:**

  Salad/Soup, Dessert/Noodles/Rice/Bread, Meat with vegetables, Pure meat, Seafood

- **Estimate a basic price level for each type of dish**
- **Generate a random value between 0~5 to estimate the specific price of each dish**

Set budget P manually

```
# set the basic price of different level
# 1: vegetable & soup, 2: dessert & staple food, 3: meat & vegetables/staple food, 4: meat, 5: seafood
basic_price = {1: 6,
               2: 12,
               3: 14,
               4: 16,
               5: 30}

# generate a random price with basic price for each dish
random.seed(2)
r_price = []
for i in price['pricing']:
    r_price.append(basic_price[i] + random.randint(0, 600)/100)

price['price'] = r_price
price
```

	index	title	pricing	price
0	0	Curried Lentil, Tomato, and Coconut Soup	1	6.57
1	1	Roasted Butternut Squash with Herb Oil and Goa...	2	12.93
2	2	Pumpkin Muffins	2	12.86
3	3	Chopped Salad with Shallot Vinaigrette, Feta, ...	1	9.69
4	4	Grain Salad with Olives and Whole-Lemon Vinaig...	1	7.73
...	...	...	...	...
235	292	White Chicken Chili	4	17.03
236	293	One-Pot Curried Cauliflower with Couscous and ...	1	7.07
237	295	Hummus Dinner Bowls with Spiced Ground Beef an...	3	17.13
238	296	Autumn Kale Salad	1	8.03
239	297	Pumpkin Icebox Pie With Snickerdoodle Crust	2	12.16

4 Allergy Matrix

- Collect each Teletubby's allergy food
- Build the allergy matrix

```
# get the dish list and the allergic food list
title = nutrition2['title'].tolist()
allergy_food = ['Shrimp', 'Pork', 'Shell', 'Fish']
```

Set the allergy food

```
# construct the allergy matrix
allergy_matrix = np.zeros((240, 240)).tolist()
for j in range(240):
    al = 0
    for food in allergy_food:
        if food in title[j]:
            al += 1
    if al == 0:
        allergy_matrix[j][j] = 1
```

```
# count the number of allergic food in the dish list
ll = []
for j in range(240):
    ll.append(allergy_matrix[j][j])

240 - sum(ll)
```

```
22.0
```

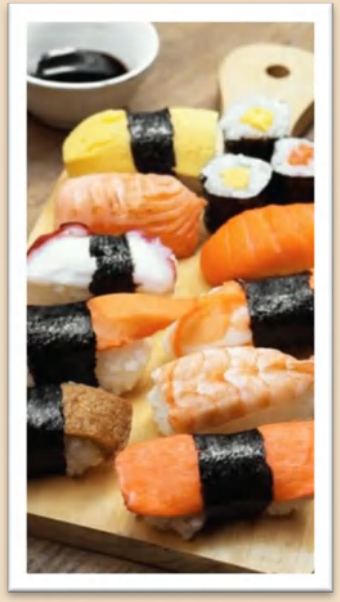
Count the number of allergic dishes

5 Availability Matrix

Collect the time availability matrix manually:

```
# construct the availability matrix
availability_matrix = [[1,1,0,1,1,1,0,1,1,1],
                       [1,0,0,1,1,1,0,1,1,1],
                       [1,1,0,1,1,1,0,1,1,1],
                       [1,1,0,1,1,1,0,1,1,1]]
```






6 Cooking Ability Matrix

- Collect the dishes each Teletubby can or can not cook
- Build the cooking ability matrix

```
# construct the cooking ability matrix
cooking_ability_matrix = np.ones((4, 240)).tolist()

# Tinky-Winky can only cook ['Pasta', 'Noodle', 'Noodles', 'Burger', 'Salad']
for j in range(240):
    cooking_ability_matrix[0][j] = 0
    for food in ['Pasta', 'Noodle', 'Noodles', 'Burger', 'Salad']:
        if food in title[j]:
            cooking_ability_matrix[0][j] = 1

# Dipsy can not cook ['Beef', 'Lamb']
for j in range(240):
    for food in ['Beef', 'Lamb']:
        if food in title[j]:
            cooking_ability_matrix[1][j] = 0

# Laa-Laa can not cook ['cake', 'Pie']
for j in range(240):
    for food in ['cake', 'Pie']:
        if food in title[j]:
            cooking_ability_matrix[2][j] = 0

# Po can not cook ['Shrimp', 'Shell', 'Fish']
for j in range(240):
    for food in ['Shrimp', 'Shell', 'Fish']:
        if food in title[j]:
            cooking_ability_matrix[3][j] = 0
```

Code and Result

```
# set up the optimization problem
meals = pulp.LpProblem("Meal_Plan", pulp.LpMaximize)
# the number of meals
meal = list(range(10))
nm = len(meal)

# the number of Teletubbies
Teletubby = ['Tinky-Winky', 'Dipsy', 'Laa-Laa', 'Po']
ns = len(Teletubby)

# the number of dishes
dish = nutrition2['title'].tolist()
nd = len(dish)

# set the variable

# d[j][k] --- whether dish j is selected in meal k
d = pulp.LpVariable.dicts("dish", (dish, meal), 0, 1, pulp.LpInteger)

# y[i][k] --- whether Teletubby i cooks in meal k
y = pulp.LpVariable.dicts("cook", (Teletubby, meal), 0, 1, pulp.LpInteger)
```

Code and Result

```
# set up the parameters
# rating
r = rating.values.tolist()
r = pulp.makeDict([Teletubby, dish], r, 0)

# nutrition
n_sodium = nutrition2['sodium'].tolist()
n_sodium = pulp.makeDict([dish], n_sodium, 0)
up_sodium = 2300 * 4
low_sodium = 0

n_calory = nutrition2['calories'].tolist()
n_calory = pulp.makeDict([dish], n_calory, 0)
up_calory = 2000 * 4
low_calory = 0

n_fat = nutrition2['fat'].tolist()
n_fat = pulp.makeDict([dish], n_fat, 0)
up_fat = 35 * 4
low_fat = 25 * 4

n_protein = nutrition2['protein'].tolist()
n_protein = pulp.makeDict([dish], n_protein, 0)
up_protein = 46 * 4
low_protein = 0
```

```
# price and budget
p = price['price'].values.tolist()
p = pulp.makeDict([dish], p, 0)
budget = 300

# allergy
a = allergy_matrix
a = pulp.makeDict([dish, dish], a, 0)

# availability
z = availability_matrix
z = pulp.makeDict([Teletubby, meal], z, 0)


# cooking ability --- M[i][j] means whehter student i can cook dish j
M = cooking_ability_matrix
M = pulp.makeDict([Teletubby, dish], M, 0)
```

7 Code and Result

```
# objective function
meals += (pulp.lpSum([r[i][j] * d[j][k] for i in Teletubby for k in meal for j in dish]), 'sum_of_rating')
```

```
# constraits

# the number of dishes in every meal --- no more than 3 dishes per meal
for k in meal:
    meals += (pulp.lpSum([d[j][k] for j in dish]) <= 3)

# we eat the same dish at most twice
for j in dish:
    meals += (pulp.lpSum([d[j][k] for k in meal]) <= 2)
# nutrition
for k in [1, 3, 5, 7, 9]:
    # sodium
    meals += (pulp.lpSum([n_sodium[j] * d[j][k - 1] + n_sodium[j] * d[j][k] for j in dish]) <= up_sodium)
    meals += (pulp.lpSum([n_sodium[j] * d[j][k - 1] + n_sodium[j] * d[j][k] for j in dish]) >= low_sodium)

    # calory
    meals += (pulp.lpSum([n_calory[j] * d[j][k - 1] + n_calory[j] * d[j][k] for j in dish]) <= up_calory)
    meals += (pulp.lpSum([n_calory[j] * d[j][k - 1] + n_calory[j] * d[j][k] for j in dish]) >= low_calory)

    # fat
    meals += (pulp.lpSum([n_fat[j] * d[j][k - 1] + n_fat[j] * d[j][k] for j in dish]) <= up_fat)
    meals += (pulp.lpSum([n_fat[j] * d[j][k - 1] + n_fat[j] * d[j][k] for j in dish]) >= low_fat)

    # protein
    meals += (pulp.lpSum([n_protein[j] * d[j][k - 1] + n_protein[j] * d[j][k] for j in dish]) <= up_protein)
    meals += (pulp.lpSum([n_protein[j] * d[j][k - 1] + n_protein[j] * d[j][k] for j in dish]) >= low_protein)
```

```
# budget and price
for k in meal:
    meals += (pulp.lpSum([p[j] * d[j][k] for j in dish]) <= budget)

# allergy
for k in meal:
    for j in dish:
        meals += (a[j][j] * d[j][k] == d[j][k])

# cooking times constraints of y with z:
# the Teletubbies who cook at meal k must be available
for i in Teletubby:
    for k in meal:
        meals += (y[i][k] <= z[i][k])

# each meal will be cooked by no more than 2 Teletubbies
for k in meal:
    meals += (pulp.lpSum([y[i][k] for i in Teletubby]) <= 2)

# the Teletubbies will cook 3~4 meals in total
for i in Teletubby:
    meals += (pulp.lpSum([y[i][k] for k in meal]) <= 4)
    meals += (pulp.lpSum([y[i][k] for k in meal]) >= 3)
# ability of cooking
for k in meal:
    for j in dish:
        meals += (d[j][k] <= pulp.lpSum([y[i][k] * M[i][j] for i in Teletubby]))
```

Code and Result

```
# solve the problem
status = meals.solve()

# the optimizaiton objective function value is printed to the screen
print("Total Rating Score = ", pulp.value(meals.objective))
print('---------------------------')
print('-------------------------------------------------------------------------\n')

# print the dishes selected for each meal and Teletubbies who cook for each meal
weekday = {0:'Monday', 1:'Tuesday', 2:'Wednesday', 3:'Thursday', 4:'Friday'}
for k in meal:
    # print weekday and lunch/dinner
    if k%2 == 0:
        w = weekday[k/2]
        print(w + ' Lunch:')
    else:
        w = weekday[(k-1)/2]
        print(w + ' Dinner:')

    # print cooking Teletubby and menu
    ls_cook = []
    ls_dish = []
    for v in meals.variables():
        if (str(v).startswith('cook')) & (str(v).endswith(str(k))) & (v.varValue == 1):
            ls_cook.append(str(v)[:-2][5:])
        if (str(v).startswith('dish')) & (str(v).endswith(str(k))) & (v.varValue == 1):
            ls_dish.append(str(v)[:-3][5:])

    if ls_cook == []:
        print('No one is available! Eat outside!') # if all Teletubbies have class during the cooking time, eat outside
    elif ls_dish == []:
        print('The meal does not meet the nutrition requirements! Eat outside!') # if all Teletubbies who is available can not cook the dishes selected
    else:
        print('Teletubbies who cook: ' + str(ls_cook))
        print('Dishes: ')
        for i in ls_dish:
            print(i)

    print('-------------------------------------------------------------------------\n')
```

Solve the potential problems by printing “Eat Outside!”

Code and Result

7

```
Total Rating Score =  348.0
----------------------------
--------------------------------------------------------------------

Monday Lunch:
Teletubbies who cook: ['Po']
Dishes:
French_Spiced_Bread
Quick_Sesame_Chicken_with_Broccoli
Sheet_Pan_Steak_Fajitas
--------------------------------------------------------------------

Monday Dinner:
Teletubbies who cook: ['Laa_Laa', 'Tinky_Winky']
Dishes:
Honeynut_Squash_with_Radicchio_and_Miso
Roasted_Squash_with_Mint_and_Toasted_Pumpkin_Seeds
Sheet_Pan_Steak_Fajitas
--------------------------------------------------------------------

Tuesday Lunch:
No one is available! Eat outside!
--------------------------------------------------------------------

Tuesday Dinner:
Teletubbies who cook: ['Laa_Laa', 'Tinky_Winky']
Dishes:
Butternut_Squash_Sandwich_with_Cheddar_Cheese_and_Pickled_Red_Onion
Easy_Green_Curry_with_Chicken,_Bell_Pepper,_and_Sugar_Snap_Peas
Shredded_Chicken_Salad_with_Creamy_Miso_Dressing
--------------------------------------------------------------------

Wednesday Lunch:
Teletubbies who cook: ['Dipsy']
Dishes:
Marinated_Mixed_Beans
Orange_Sweet_Rolls
Smoky_Pumpkin,_Spelt,_Pomegranate,_and_Feta_Salad
--------------------------------------------------------------------

--------------------------------------------------------------------

Wednesday Dinner:
Teletubbies who cook: ['Dipsy']
Dishes:
Marinated_Mixed_Beans
Roasted_Squash_with_Mint_and_Toasted_Pumpkin_Seeds
Shredded_Chicken_Salad_with_Creamy_Miso_Dressing
--------------------------------------------------------------------

Thursday Lunch:
No one is available! Eat outside!
--------------------------------------------------------------------

Thursday Dinner:
Teletubbies who cook: ['Po']
Dishes:
Butternut_Squash_Sandwich_with_Cheddar_Cheese_and_Pickled_Red_Onion
Grilled_Cheese_Tacos
Smoky_Pumpkin,_Spelt,_Pomegranate,_and_Feta_Salad
--------------------------------------------------------------------

Friday Lunch:
Teletubbies who cook: ['Laa_Laa', 'Po']
Dishes:
Istanbul_Style_Wet_Burger_(Islak_Burger)
Orange_Sweet_Rolls
Quick_Sesame_Chicken_with_Broccoli
--------------------------------------------------------------------

Friday Dinner:
Teletubbies who cook: ['Dipsy', 'Tinky_Winky']
Dishes:
French_Spiced_Bread
Honeynut_Squash_with_Radicchio_and_Miso
Istanbul_Style_Wet_Burger_(Islak_Burger)
--------------------------------------------------------------------
```

04 ▶ Creative Ideas

1. Eat Outside Solution
2. Other Creative Ideas

1 Eat Outside Solution

Buy selected dishes in the restaurant

The Optimization Problem:

$$Maximize \quad \sum_{i=1}^{n_s}\sum_{k=1}^{n_m}\sum_{j=1}^{n_d} r_{ij}d_{jk}$$

$$Subject\ to$$

$$\sum_{j=1}^{n_d}(N_j^s d_{jk} + N_j^s d_{j,k+1}) \le U^s, \sum_{j=1}^{n_d}(N_j^s d_{jk} + N_j^s d_{j,k+1}) \ge L^s \qquad for\ \forall k \in \{1,3,5,7,9\}$$

$$\sum_{j=1}^{n_d}(N_j^c d_{jk} + N_j^c d_{j,k+1}) \le U^c, \sum_{j=1}^{n_d}(N_j^c d_{jk} + N_j^c d_{j,k+1}) \ge L^c \qquad for\ \forall k \in \{1,3,5,7,9\}$$

$$\sum_{j=1}^{n_d}(N_j^f d_{jk} + N_j^f d_{j,k+1}) \le U^f, \sum_{j=1}^{n_d}(N_j^f d_{jk} + N_j^f d_{j,k+1}) \ge L^f \qquad for\ \forall k \in \{1,3,5,7,9\}$$

$$\sum_{j=1}^{n_d}(N_j^p d_{jk} + N_j^p d_{j,k+1}) \le U^p, \sum_{j=1}^{n_d}(N_j^p d_{jk} + N_j^p d_{j,k+1}) \ge L^p \qquad for\ \forall k \in \{1,3,5,7,9\}$$

Nutrition constraints

$$\sum_{k=1}^{n_m}\sum_{j=1}^{n_d} p'_j d_{jk} \le P'$$

Budget constraints

$$a_{jj}d_{jk} = d_{jk} \qquad \forall k \in \{1,2,\dots,10\}, \forall j \in \{1,2,\dots,270\}$$

Allergy constraints

Add more constraints

1 Eat Outside Solution

If there exists someone who can cook meal k → do not buy dishes for this meal
If no one is available for cooking meal k → select 2-4 dishes and buy it in the restaurant

$$if \sum_{i=1}^{4} y_{ik}^{*} = 1 \text{ or } 2 \rightarrow \sum_{j=1}^{240} d_{jk} = 0$$

$$if \sum_{i=1}^{4} y_{ik}^{*} = 0 \rightarrow \sum_{j=1}^{240} d_{jk} = 2 \text{ or } 3 \text{ or } 4$$

Transform to Constraints, use y^{*} we calculated in the previous model

$$\sum_{i=1}^{4} y_{ik}^{*} \sum_{j=1}^{240} d_{jk} = 0 \qquad for\ k = 1, 2, \dots, 10$$

$$\sum_{i=1}^{4} y_{ik}^{*} + \frac{1}{2} \sum_{j=1}^{240} d_{jk} \geq 1 \qquad for\ k = 1, 2, \dots, 10$$

$$\sum_{i=1}^{4} y_{ik}^{*} + \frac{1}{2} \sum_{j=1}^{240} d_{jk} \leq 2 \qquad for\ k = 1, 2, \dots, 10$$

$$if\ a = 1 \text{ or } 2 \rightarrow b = 0$$
$$if\ a = 0 \rightarrow b = 2 \text{ or } 3 \text{ or } 4$$

Transform to Constraints

$$a \times b = 0$$

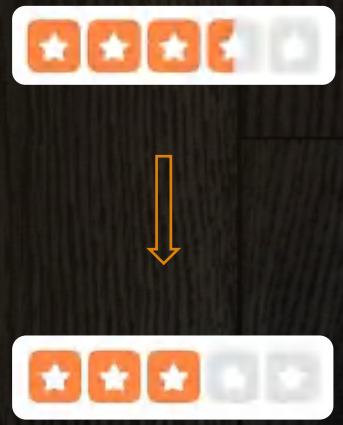
$$a + \frac{1}{2} b \geq 1$$

$$a + \frac{1}{2} b \leq 2$$

2 Other Creative Ideas

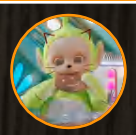
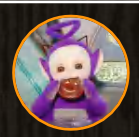
Balanced Diet

Cooking Time



Update Rating

Contribution & Timeline

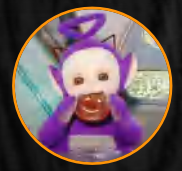
Model Building	Rating Matix Estimation	Creative Ideas	Report
 • Basic model building • Takeout model building • Data collection	 • Data cleaning and Preparation  • Code for Low-rank Fractorization	 All	 All

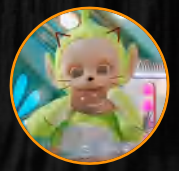
3.10 — Discussion & initial creative ideas

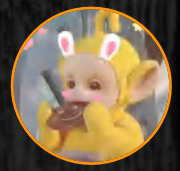
3.19 — Finish initial model building

4.10 — Finish other creative ideas

4.10 — Final report

 Wanning Li

 Yuxuan Cheng

 Qianran Ma

 Yanan Zhou

Reference

[1] Sheen, Spencer. (2016). A Coordinate Descent Method for Robust Matrix Factorization and Applications. SIAM Undergraduate Research Online. 9. 10.1137/15S014472.

[2] Yang, J. (2015). Notes on Low-rank Matrix Factorization. ArXiv, abs/1507.00333.

The sun is setting in the sky, Teletubbies **say goodbye!**

Thank you for your attention!