MGMTMSA 408 – Operations Analytics

Homework 4 – Revenue Management and Assortment Optimization (Question Sheet)

Due June 3, 2020 at 1pm PST

1 Cloud Computing

Cirrus is a cloud computing provider that offers its users the ability to reserve computational capacity on its cloud. In this problem, we will analyze how Cirrus should allocate its capacity.

Cirrus sells its computational resources in units called *instances*. These are virtual computers that a user can reserve for a whole day. Each instance consist of three components:

- Central processing units (CPUs): each instance reserves some number of CPUs, range from 1 CPU to 64 CPUs.
- Memory: each instance reserves some amount of memory, ranging from 1 GB to 128GB, in increments of 1 GB.
- Graphical processing units (GPUs): each instance reserves some amount of GPU memory, ranging from 1 GB to 8 GB.

In total, on a given day, Cirrus has 512 CPUs, 1024 GB of memory and 64 GB of GPU memory available.

Based on its user requirements, offers the following instances:

Instance	Name	CPU (#)	Memory (GB)	GPU (GB)	Price	Rate (# / day)
1	C1	16	8	1	\$7	5.0
2	C2	32	16	1	\$12	5.0
3	C3	64	32	1	\$24	1.8
4	M1	8	32	1	\$22	3.0
5	M2	16	64	1	\$44	2.6
6	M3	32	128	1	\$88	1.0
7	G1	16	16	2	\$30	0.8
8	G2	32	32	6	\$90	0.4
9	G3	64	64	8	\$120	0.3

Columns 3 - 5 specify the hardware requirements of each instance type. Column 6 specifies the price for activating the instance. We will also assume that an instance is reserved for a whole day.

Column 7 specifies the arrival rate. This is the number of users requesting an instance of this type each day. In this problem, we'll assume that the demand for each type of instance follows a Poisson arrival process, with the arrival rate as given in the table above. We will assume that

Cirrus only allows reservations for a period of 5 days. We will also assume that this period of 5 days precedes the day for which the instance is reserved and is exclusive of this day (i.e., the requests are not fulfilled during the reservation period). For example, instance requests can come in on Monday, Tuesday, Wednesday, Thursday and Friday, at any time on those days; the instances that are accepted are then reserved for the whole day of Saturday. Instances are *not* reserved during the 5 day reservation period.

Cirrus is interested in understanding how it should allocate its limited capacity to the different types of instances. Cirrus currently accepts requests in a first-come first-serve fashion. In this problem, we will approach Cirrus's problem from a revenue management lens.

Part 1: Capacity control formulation

In this first part of the problem, we need to mathematically formulate the capacity control problem for Cirrus as a linear optimization problem. Let $x_1, \ldots, x_9$ be the number of instances that are reserved of each instance type. These will be the decision variables of our problem.

a) Based on the prices given in the table earlier, what is the objective function of the problem?

b) In terms of $x_1, \ldots, x_9$, what is the constraint on the total memory usage of the instances that are reserved?

c) Let's now consider the forecasted demand for each instance type. Recall that for a Poisson arrival process with rate λ per unit time, the total number of arrivals in a period of length T is a Poisson random variable Y with mean λT.
Over the five day period, what is the expected number of requests of instance type 5? Assuming there will be exactly this many requests over the $T = 5$ day period, what is the constraint on the number of requests of instance type 5 we may accept?

d) Using the logic in (a) – (c), write down the mathematical formulation of the $T = 5$ day capacity control problem as a linear optimization problem.

Part 2: Solving the capacity control problem in Python/Gurobi

Now, implement your formulation in Part 1(d) using Python and Gurobi.

a) Solve the problem. What is the optimal objective value?

b) In the optimal allocation, how many requests of instance type C1 are accepted?

c) The management of Cirrus is considering increasing the available GPU memory, since GPU instances appear to be more lucrative. Based on the solution to the LP problem, would you recommend doing this? Explain your answer.

d) Cirrus can rent an additional server from a smaller cloud provider. This server would add an additional 32 CPUs and 16 GB of memory to Cirrus's capacity. This server would be rented at \$5/day. Should Cirrus rent the server?

Part 3: Simulating current practice

Cirrus would like to understand how well its current policy does. Currently, Cirrus simply accepts requests in a first-come first-serve fashion, without considering the revenue of the requests. In this part of the problem, we will simulate Cirrus's current policy.

Set your random seed to 10 using the `random.seed()` function in `numpy`. Using the provided function `generateArrivalSequences` in the `HW4_Cloud Code.ipynb` notebook, generate 100 sequences of request arrivals, using the rates provided in the table above, over a period of $T = 5$ days. This function will generate an array of arrays:

- `arrival_sequence_times`: This contains the time at which each request arrives, in the interval $[0, 5]$. For example, `arrival_sequence_times[0][1]` is the time at which the second request arrives in the first sequence.
- `arrival_sequence_types`: The instance type of each request. (*Note*: the instance types in this array are numbered from 0 to 8, in accordance with how Python's numbering starts at zero. Thus, for example, a value of 5 in this array indicates instance type 6 / M3.)

If you have done this step correctly, the first three times in `arrival_sequences_times[0]` should be 0.07414243 0.1246028 0.15928449.

a) What is the average number of arrivals of type C1 in the set of simulated sequences?

b) What is the average number of arrivals, of all types, over the set of simulated sequences? Does this make sense? (*Hint*: What is the expected value of the sum of Poisson random variables?)

c) Next, implement Cirrus's current policy. This policy accepts any request, so long as there is capacity for it. You may use the code skeleton given in `HW4_Cloud Code.ipynb` as a starting point.
What is the average revenue garnered by this policy over the 100 simulated sequences?

d) What is the average remaining capacity (of CPUs, memory and GPUs) of this policy?

Part 4: A bid-price control policy

Let's now develop a bid-price control policy, using the solution of the LP.

a) As a warm-up, suppose that we receive a request for instance type 5 (M2). Let π_1, π_2, π_3 be the shadow prices / dual values of the constraints for CPUs, memory and GPUs at a particular point in time. In terms of the shadow prices, what is the approximate opportunity cost of accepting this request?

b) As a further warm-up, suppose that we receive a request at time t. What is the expected number of requests of type i we will receive from time t to time T (including both t and T)?

c) Now, based on the logic in (a) and (b), implement a bid-price control policy in Python. Apply this policy to the same 100 simulated sequences from Part 3. You may use the code skeleton given in `HW4_Cloud Code.ipynb` as a starting point.
What is the average revenue garnered by this policy over the 100 simulated sequences?

d) What is the average remaining capacity (of CPUs, memory and GPUs) of this policy?

2 Designing a Sushi Menu

A high-end sushi restaurant is trying to decide what types of sushi to offer to its customers. The restaurant surveys $K = 500$ of its customers to obtain ratings of $n = 100$ different types of sushi. The ratings of the 500 customers for the 100 types of sushis are stored in the file `sushi_utilities_mat.csv` (rows are sushis, columns are customers). In addition, the restaurant also has additional information on the sushis: the file `sushi_info.csv` contains the name of each of the 100 sushi types (column 1), a category code for each sushi (column 2) and a price for each sushi (column 3). You may also find it helpful to consult the file `sushi_description.txt` which includes the name and a short description of each sushi.

We will assume that customers choose according to a first-choice model of choice, and that the utility of each option is the rating provided in `sushi_utilities_mat.csv`. Namely, each customer considers the available options, evaluates the utility of each option, and then selects the option that provides the highest utility. We will assume that the utility of the no-purchase option of each customer type is 3. So for example, suppose we offer sushi #2, #7 and #12. Let's fix customer #7. Customer #7 evaluates the utility of each of those options:

$$
\begin{aligned}
&u_{7,2} = 3.985 \quad \text{(utility of sushi \#2 (= anago / sea eel))} \\
&u_{7,7} = 3.976 \quad \text{(utility of sushi \#7 (= ikura / salmon roe))} \\
&u_{7,12} = 3.530 \quad \text{(utility of sushi \#12 (= hotategai / scallop))} \\
&u_{7,0} = 3 \quad \text{(utility of no-purchase / outside option)}
\end{aligned}
$$

If these are the options, customer #7 will choose sushi #2 (sea eel) because this provides the highest utility.

In our model of the customer choice behavior, we assume each customer selects his/her highest utility option. We also assume that each customer has an equal weight/probability $(1/K)$. Therefore, if we offer the set $S \subseteq \{1, \ldots, n\}$, then the predicted revenue is the average of the revenue from the choice of each of the 500 customers: mathematically, this is

$$
R(S) = \frac{1}{K} \sum_{k=1}^{K} \left[\sum_{i \in S} r_i \cdot \mathbb{I}\{i \text{ is the first choice of customer } k \text{ in } S \cup \{0\}\} \right]
$$

where $\mathbb{I}\{A\}$ is the indicator function for the event A (it is 1 if A is true, and 0 if A is false) and the first choice is that choice which provides the highest value of $u_{k,i}$, i.e., it is $\arg\max_{i \in S \cup \{0\}} u_{k,i}$.

Part 1: Understanding the data

Before we start, let's understand some patterns in the data. Load the file `sushi_utilities_mat.csv`.

a) What are the five most preferred sushis for customer 1? *Hint*: use the command `argsort` from `numpy` on a particular row of the utility matrix.

b) What are the five least preferred sushis for customer 2?

c) For each customer, compute the *rank* of each of the 100 sushis according to their utilities. Which sushis are the top five, i.e., the five with the best average rank over the 500 customers? What do you notice about the sushi (specifically, the type of fish)?

Hint: Consider the following code snippet:

```
import numpy as np
temp = np.array([4.8, 2.3, 5.3, 3.9, 1.4, 5.1])
# temp contains the (hypothetical) utilities of 6 products
# Apply argsort once:
temp2 = np.argsort(temp) # What does temp2 contain?
temp3 = np.argsort(temp2) # What does temp3 contain?
```

d) Which sushi has the worst average rank over the 500 customers?

e) Which sushi is the most controversial, as measured by the standard deviation of its rank over the 500 customers?

Part 2: Common-sense solutions

Load the data into Python. Create a function that computes the expected per-customer revenue of an assortment of sushi items S, where each customer is picking the option that gives them the highest utility. As a check, if we offer the set of sushis $S = \{1, 2, 3, 4, 5\}$, then the per-customer revenue should be 14.2396. (*Note*: for some customers, it will be the case that all of the sushis will have a utility $u_{k,i}$ lower than the no-purchase utility $u_{k,0}$; these customers will never choose any sushi we offer them. Customers #1, 2 and 5 are example of this. Despite these customers not choosing anything, please continue to include them in the K of 500 for your revenue calculations.)

a) Suppose that we simply offer all sushi products, that is, we set $S = \{1, 2, \ldots, n\}$. What is the expected revenue in this case?

b) Suppose that we offer the ten highest revenue sushis, that is, we set $S = \{i_1, \ldots, i_{10}\}$, where $r_{i_1} \geq r_{i_2} \geq \cdots \geq r_{i_n}$. Which are the ten highest revenue sushis? What is the expected revenue in this case?

c) Suppose that for every customer k, we determine his/her most preferred sushi, $i_k^* = \arg\max_{1 \leq i \leq n} u_{k,i}$. We then offer all of the most preferred sushis, that is, we set $S = \{i_1^*, i_2^*, \ldots, i_K^*\}$. (Note that some sushis may be the top choice of more than one customer.)
What is the expected revenue in this case? What do you notice about this value?

d) Explain why the solution in (a) may be suboptimal.

e) Explain why the solution in (b) may be suboptimal.

Part 3: An integer optimization model

Let's see how to formulate the problem of selecting an optimal set of sushis as an integer optimization problem.

Consider the following integer optimization model:

$$\underset{\mathbf{x},\mathbf{y}}{\text{maximize}} \quad \frac{1}{K} \sum_{k=1}^{K} \sum_{i=1}^{n} r_i y_{k,i} \tag{1a}$$

$$\text{subject to} \quad \sum_{i=0}^{n} y_{k,i} = 1, \quad \forall\, k \in \{1, \ldots, K\}, \tag{1b}$$

$$\sum_{j=0}^{n} u_{k,j} y_{k,j} \geq u_{k,i} x_i + u_{k,0}(1 - x_i), \quad \forall\ k \in \{1, \ldots, K\},\ i \in \{1, \ldots, n\}, \tag{1c}$$

$$\sum_{j=0}^{n} u_{k,j} y_{k,j} \geq u_{k,0}, \quad \forall\ k \in \{1, \ldots, K\}, \tag{1d}$$

$$y_{k,i} \leq x_i, \quad \forall\ k \in \{1, \ldots, K\},\ i \in \{1, \ldots, n\}, \tag{1e}$$

$$x_i \in \{0, 1\}, \quad \forall\ i \in \{1, \ldots, n\}, \tag{1f}$$

$$y_{k,i} \in \{0, 1\}, \quad \forall\ k \in \{1, \ldots, K\},\ i \in \{1, \ldots, n\}, \tag{1g}$$

a) Explain how constraints (1b) – (1e) correctly model the preferences of each customer.

b) Implement the linear optimization *relaxation* of the above problem in Python. The relaxation is obtained by relaxing $\mathbf{x}$ and $\mathbf{y}$ to be continuous decision variables as opposed to binary decision variables – that is, we replace constraints (1f) and (1g) with:

$$0 \leq x_i \leq 1, \quad \forall\ i \in \{1, \ldots, n\}, \tag{2}$$

$$0 \leq y_{k,i} \leq 1, \quad \forall\ k \in \{1, \ldots, K\},\ i \in \{1, \ldots, n\}. \tag{3}$$

What is the optimal objective value of the relaxation?

c) A manager claims that it should be possible to obtain an assortment with an expected per-customer revenue of \$32. Explain why this is impossible based on your answer to (b).

d) Now, implement the integer version of the above problem (i.e., the binary constraints (1f) and (1g) are enforced) in Python. Solve the problem. What is the expected per-customer revenue of the optimal assortment? How much does this improve over the assortments from Part 2?

e) What is the optimal set of sushis the restaurant should offer?

Part 4: A constrained model (OPTIONAL)

After discussing your recommendation from Part 3 with the executive chef, some creative disagreements arose, and the chef requested that the assortment be further refined. In particular, the chef felt it important to ensure that twelve broad categories of sushis were represented in the offering of sushis. The category of each sushi is included as the second column of `sushi_info.csv`. The categories are:

- 0: aomono (blue-skinned fish)
- 1: akami (red-meat fish)
- 2: shiromi (white-meat fish)
- 3: tare (eel and eel-like creatures)
- 4: clam or shell-fish
- 5: squid or octopus

- 6: shrimp or crab
- 7: roe
- 8: other seafood
- 9: egg
- 10: meat other than fish
- 11: vegetables

The chef asked that the assortment be re-designed with the constraint that there be at least one sushi from each of these 12 categories.

a) (OPTIONAL) How can this requirement be modeled in terms of the decision variables of problem (1)? Write down the constraints.

b) (OPTIONAL) Implement this requirement as a collection of constraints in your integer optimization model. Solve the new integer optimization model. What is the optimal objective value?

c) (OPTIONAL) What is the new menu of sushis?

Part 5: Next steps (OPTIONAL)

a) (OPTIONAL) What are some limitations of our model? Provide at least three.

b) (OPTIONAL) The size of the IO formulation (1) scales with the number of customers whose utilities we model. Can you think of a heuristic approach by which we might reduce the number of customers K in our model? (*Hint*: Think of the methods we have discussed in earlier classes.)