ORIE3120_Recitation_11

April 23, 2022

0.0.1 ORIE 3120 Recitation 11

0.1 1. Introduction

Due on Friday, April 22

In this recitation we will learn how to employ the newsvendor problem to solve a scheduling problem. More specifically, we want to find the optimal number of nurses to be scheduled to work on the quarantine & isolation team at Cornell Health to handle positive COVID tests.

Whenever a student tests positive, this team of nurses calls that student to check on their symptoms and to move them to a room at a hotel. The nursing team also calls students identified as close contacts of the positive student, and who may have already been infected, to move them into quarantine at a hotel. This is done to protect against the possibility that others would be infected by coming into contact with someone who is infectious. In many cases this is straightforward but it sometimes requires a substantial amount of work: the person being contacted might have a medical situation that requires extra attention (either from COVID or another condition); might not answer their phone (requiring repeated calls); might be belligerent or misrepresent the truth (requiring a longer conversation and sometimes the use of additional resources); or might simply have a great number of questions.

This is a newsvendor problem because demand is stochastic and there is a cost both for scheduling too few nurses, and for scheduling too many. If we schedule too many nurses, we create a burden for them and risk burning out the nursing team. If we schedule too few, we can ask unscheduled nurses to come on and work at the last minute, but this forces them to drop their personal life and start working (creating a burden and risking burnout). Also, scheduling too few nurses creates a risk that we might not be able to contact and move to quarantine / isolation all of the students who are potentially infectious, risking the creation of new COVID cases.

We focus on the arrival period for the Spring 2021 semester. During this period, students arriving from elsewhere were testing positive due to infections acquired at home. The rapid increase in arriving students and the changing nature of nationwide and international prevalence relative to the Fall 2020 semester made it difficult to plan based on past experience. This made mathematical modeling extremely useful. Indeed, code close to this recitation was actually used to make staffing decisions at Cornell Health over this period.

[ ]:
```
# Importing basic libraries
import numpy as np
import pandas as pd
import numpy.random as rand
import matplotlib.pyplot as plt
```

```
import matplotlib as mpl
%matplotlib inline
import math
import time
from IPython import display

# Setting the default parameters for the plots produced by matplotlib
# Sets figure size. Width, height in inches.
plt.rcParams["figure.figsize"] = (10,10)
# Sets label font sizes
plt.rcParams['axes.labelsize'] = 14
plt.rcParams['xtick.labelsize'] = 12
plt.rcParams['ytick.labelsize'] = 12
```

0.2 2. Planning for a Single Day

We will start by creating a model that plans for a single day, finding the optimal number of nurses to schedule for that day.

First, we will model the probability of testing positive for an arriving student by specifying some constants.

We model students as arriving from one of the following location categories - 'New York State', 'Middle States', 'New England', 'Midwest', 'South', 'Southwest', 'West', 'Territories', 'USA, unknown', and 'International'.

If we know the distribution of the arriving Cornell student population over these location categories, and the COVID prevalence at each of these locations, we can estimate the probability that an arriving student will person testing postive if they arrive from 'X' location.

We can get the data we need from https://www.cornell.edu/about/facts.cfm

[ ]:
```
# Data from website
locations = ['New York State','Middle States','New England','Midwest','South',
             'Southwest','West','Territories','USA, unknown', 'International']
 ↪ # Location categories
pop_percent = [27,12,7,6,8,3,11,2,2,22]  # Probability distribution of the
 ↪origin locations for incoming Cornell students
 ↪
covid_prevalence = [1.3,1.4,1.6,1.9,2.0,1.7,1.8,2.4,2.5,3.1]  # the probability
 ↪that a person coming from each origin has COVID
# Both pop_percent and covid_percents are expressed as percentages between 0
 ↪and 100.

# Create a dataframe with the above information
data = pd.DataFrame(list(zip(locations, pop_percent, covid_prevalence)),
 ↪columns=['location','pop_percent','covid_prevalence'])

data.head()
```

Question 1: Write code that calculates the probability that an arriving student tests positive. This is

$$\sum_i P(\text{student origin is } i)P(\text{student is positive}|\text{origin is } i)$$

Store this in a variable `prevalence` as a probability between 0 and 1.

We will now run a simulation to generate many scenarios.

Each scenario will include the number of positive students arriving on the day we are planning for, which is a binomial(p,n) random variable where p is the prevalence you calculated in question 1 and n is the number of students that arrive on that day. We set the number of arriving students to 1000.

It will then calculate the number of minutes of work required to handle those positive students, assuming that each positive case requires 30 minutes, each of their contacts requires 20 minutes, and each positive case has 4 contacts.

It then calculates the number of nurses needed to do this work, assuming that each nurse works an 8 hour shift.

Question 2: Fill in the two missing lines in the code below

[ ]:
```
# Defining constants
num_arrivals = 1000 # Number of arriving students
M = 1000 # Number of simulation replications

nurses_needed = []
num_pos = [0]*M
work = [0]*M

# Run simulation
for m in range(M):
    num_pos[m] = # Simulate a binomial(N, prevalence) random variable
    work[m] = # 30 min per positive, 20 min per contact, 4 contacts, work␣
 ↪defined in terms of minutes
    nurses_needed.append(int(work[m] / (8*60))) # nurse works 8h (8 x 60␣
 ↪minutes) shift

simulation_df = pd.DataFrame(nurses_needed, columns = ['nurses_needed'])
simulation_df.head()
```

To optimize for the number of nurses we bring in, we will write a function that calculates the number of extra nurses required given that `N` nurses are currently working using our simulated data.

As discussed at the beginning of this recitation, there are two bad things that can happen: * the number of nurses scheduled on the day we are planning for could be *strictly more* than the number needed to do the work necessary on this day. In this case, the difference between the number scheduled and the number needed is called *overage* * the number of nurses scheduled on the day we are planning for could be *strictly less* than the number needed to do the work necessary on this

day. In this case, the difference between the number needed and the number scheduled is called *underage*.

In particular, if the number of nurses scheduled is N and the number of nurses needed is D, then: * Overage = max(0,N-D) * Underage = max(0,D-N)

We estimate the expected overage and expected underage associated with a particular number of scheduled nurses, we calculate the overage and underage on each scenario and return the average over all scenarios.

Question 3: Fill in the two lines below to define a function that calculates the overage and underage from `sim_df`.

Hint: One way to do this very quickly is to use the `clip` function with the `lower` keyword. `df['x'].clip(lower=0)` returns the value of max(x,0) for each entry in `df['x']`.

[ ]:
```
def eval_N(nurses_needed, nurses_scheduled):
  overage = # FILL IN
  underage = # FILL IN
  return underage, overage

eval_N(sim_df['nurses_needed'],1)
```

Question 4: Using `eval_N`, plot the underage and overage vs. the number of nurses scheduled. Do this for `nurses_scheduled` ranging from 0 to 9 inclusive. You should see that the underage is high when not many nurses are scheduled (our planned resourcing is *under* demand) and overage is high when many nurses are scheduled (our planned resourcing is *over* demand).

[ ]:
```
# Your code below
```

Now, we want to define a function `optimize(df, alpha)`. It should return: * the optimal number of nurses to schedule such that we minimize the total cost (alpha*underage + overage).* the underage and overage at this optimal number of nurses to schedule

We will assume that we can schedule N = 0 to 9 (inclusive) nurses. For each N find the cost and return the N with least associated cost.

Question 5: A part of the function has been filled in for you. Please complete the function.

[ ]:
```
def optimize(nurses_needed, alpha):
  cost = [0]*10
  best_cost = np.inf
  opt_N = -1
  opt_underage = -1
  opt_overage = -1
  for N in range(10):
    # Your code here (fill in the for-loop)

  return opt_N, opt_underage, opt_overage
```

Question 6: Test your function by using it to find the optimal N, underage and overage at these

values: * alpha = 0 (you should see that the optimal overage is 0 and the optimal N is 0) * alpha = 1000 (you should see that the optimal underage is close to 0 and the optimal N is big) * alpha = 1 (you should see the optimal underage and overage are similar to each other, and the optimal N is roughly in the middle between 0 and 10)

Creating a Pareto Frontier

Unfortunately, it isn't easy for a human decision-maker to tell you their objective function. Instead, what we can do is to plot things that must be traded off in an optimal solution and ask them to choose the trade-off they like the best. This is called a "Pareto Frontier".

We can do this here by solving the optimization problem for many different values of alpha, calculating the underage and overage for each one, and then plotting underage vs. overage.

The nurse manager or medical director would then be able to select a point on this curve that they think will be best. We can then look up which value of alpha generated this point and what the optimal N was.

Question 7: Write code to create a Pareto frontier,

[ ]:
```
alpha = np.arange(0,2.5,.05)
overage = [0] * len(alpha)
underage = [0] * len(alpha)
opt_N = [0] * len(alpha)
for i in range(len(alpha)):
  # FILL IN YOUR CODE HERE

plt.plot(underage,overage,'.-')
plt.title('Plan for Monday Feb 4')
plt.xlabel('Underage Nurses')
plt.ylabel('Overage Nurses')
```

When we show this plot to management, they say that they like the point that has underage closest to 0.4 (roughly speaking this means that the probability is 40% that we'll need to call in an extra nurse).

Question 8:
What is the number of nurses that we should schedule to achieve this tradeoff between overage and underage? What is the value of alpha that gave us this optimal number of nurses?

0.3 3. Planning for Multiple Days

0.3.1 Setup

Our current model only simulates a single day. However, we would like to have an optimal schedule for the number of nurses to employ for an entire period, say 3 weeks. We could do this by making 21 plots using the code above, one for each day, using the number of students scheduled to arrive for that day. We could then show each of those 21 plots to management and have them pick out a point on each plot. This would be a pain, however. It is better to show them a single plot for the entire 3-week period and have them select a point.

In addition, we would like to introduce the possibility that a student does not test positive immediately upon arriving and thus contacts multiple students during the time he/she goes undetected. For simplicity, we do not assume the contacted students spread the virus to other contacts.

For example, a student arriving on day 0 may not test positive until day 4, and he/she may contact 6 students. We will implement this improved model below.

Before we do so, note that our objective is essentially still the same as above: for a particular α optimally choose the number of nurses to schedule to minimize the expected total cost. However, our choice of nurses to schedule is no longer a scalar, since we are considering *many* days as opposed to a single day. Rather, it is a vector (say, $\vec{N}$) whose length is the number of days in the period we are considering, and whose jth entry is our choice of the number of nurses to employ on day j.

1 Simulation

Here we'll simulate scenarios, where each scenarios consists of the amount of work that needs to be done on each day during the first 3 weeks of the semester

[ ]:
```
num_days = 21
num_scenarios = 20

# This is the number of people who arrive on each day.
# It was generated using num_arrivals = np.ceil(2000*rand.
 ↪uniform(size=num_days))
num_arrivals = [1076., 1752., 1783., 1326., 1385., 1454., 1230.,  689., 1190.,
       1853., 1716.,  405.,   70., 1097., 1504.,  634., 1407., 1222.,
       1140.,  404., 1703.]
```

[ ]:
```
# Create a dataframe containing all 0's where each row will be a scenario
# and each column will be a simulated day.  Once we run the code below, the
# values in the dataframe will be the number of nurses needed on each day in
# each scenario
df_sim = pd.DataFrame(0, index=range(num_scenarios), columns=range(num_days))

for i in range(num_scenarios):
  for d in range(num_days): # Iterate over days in our simulation
    # Simulate the students who arrive on this day
    infected_no = rand.binomial(num_arrivals[d],prevalence)
    for j in range(infected_no):
      days_infectious = rand.geometric(0.3)
      day_tested_positive = d + days_infectious
      if day_tested_positive < num_days:
        contacts = 4
        df_sim.loc[i,day_tested_positive] += (30 + 20 * contacts)/(8*60)
```

[ ]:
```
df_sim
```

Question 9: Change the above code so that the number of contacts depends on the number of days

the person is infectious. Let the number of contacts be 1.5 times the number of days infectious. This models the fact that a person who has been circulating in the campus population is going to have more local contacts.

Question 10: Plot the average amount of work per day vs. day, averaging across all of the scenarios. You will probably see that the amount of work is larger for later days, even though the distribution over the number of arriving students and the prevalence was the same on every day. Why is that?

2 Optimization

For each value of alpha, we will now find the optimal schedule of nurses. This is just like we did above, except now we will take multiple days into account. Fortunately, the optimal schedule over multiple days for a given alpha can be optimized separately for each day.

To optimize for a given value of alpha, we need to use the `optimize` function we wrote above. We use this to figure out the optimal number of nurses to schedule on that day, treating that day separately from all other days. This also tells us the optimal underage and overage on that day.

We can then add up the overage and underage costs across all days to get the total underage and overage.

Question 11 Fill in the missing line of code below using your `optimize` function to complete this.

[ ]:
```
best = pd.DataFrame()
best['alpha'] = np.arange(0,5,.25)
best['underage'] = 0
best['overage'] = 0
```

[ ]:
```
for index, row in best.iterrows():
  for d in df_sim.columns:
    opt_N, underage, overage = # FILL IN
    # These are the total values, across all days
    best.loc[index,'underage'] += underage
    best.loc[index,'overage'] += overage
```

[ ]:
```
best
```

[ ]:
```
plt.plot(best['underage'],best['overage'],'.-')
plt.xlabel('Underage')
plt.ylabel('Overage')
```

Question 12 Suppose that leadership says that they prefer the solution whose total underage over the 21 day period is closest to 10. What value (or values) of alpha that you searched does this correspond to? How many nurses are scheduled on each day under the optimal schedule for this choice for alpha? (If multiple values of alpha produce underage costs that are closes to 10, pick one.)