# ISYE 6501 Homework 6

Due: July 1, 2021

## 1 Simulation

### 1.1 Question 13.2

Using a Python simulation framework, SimPy, I could model a busy airports security system. For this problem I assume that passengers arrive to an airport's ID/boarding pass queue according to a Poisson distribution with $\lambda = 5$ per minute. Several workers can assist these passengers with a service time of $\mu = 0.75$ minutes. After this assistance, passengers are sent to a personal-check queue where they pass through a scanner in a time that is uniformly distributed between 0.5 and 1 minute. I arbitrarily chose to run the simulation for 2 hours and examined the results after varying the number of workers, number of personal check queues and arrival rate of passengers. The ultimate goal was to determine what combination of the number of workers and personal check queues results in the average wait time below 15 minutes. Results are summarized in Table 1 and my Python code which followed a similar method to [1] can be found in Appendix A.

| # of workers | # of personal check queues | Avg. wait time (mins) |
|---|---|---|
| 1 | 1 | 46.7 |
| 2 | 2 | 30.6 |
| 3 | 3 | 19.5 |
| 3 | 4 | 18.7 |
| 4 | 3 | 9.8 |
| 3 | 2 | 28.5 |
| 4 | 2 | 24.7 |
| 3 | 5 | 18.7 |
| 3 | 6 | 18.6 |

Table 1: Simulation results with arrival rate Poisson($\lambda = 5$)

It is apparent after a brief search that varying the number of workers has the most impact on decreasing the average wait time. This makes perfect sense because although the the workers take on average the same time as the personal check queues, the inter-arrival time is skewed towards the lower end and therefore longer assistance times (exponential vs. uniform distribution). One can think of the scenario as "you're only as strong as your weakest link" in a sense. The average wait time finally dips below 15 minutes when the number of workers is 4 and the number of queues is 3.

For comparison, I completed the same task but this time simulating a busier airport i.e. when the passenger arrival rate is Poisson($\lambda = 50$). The results are summarized in Table 2. As expected, it takes many more workers and queues to satisfy keeping the wait time below 15 minutes on average. It is work noting that there is certainly some trade off between these two parameters. For example there could be several combinations that satisfy the desired condition but one could turn this into an optimization problem and take the objective function or cost into consideration and choose the combination of parameters that costs the airport the least. This could also come at the cost of customer happiness however.

| # of workers | # of personal check queues | Avg. wait time (mins) |
|---|---|---|
| 5 | 5 | 53.5 |
| 10 | 10 | 45.9 |
| 20 | 20 | 28.7 |
| 30 | 30 | 12.9 |
| 30 | 20 | 27.8 |
| 35 | 20 | 27.9 |

Table 2: Simulation results with arrival rate Poisson($\lambda = 50$)

# 2 Missing Data

## 2.1 Question 14.1

Not all datasets are all pretty and groomed after acquisition. Often times there are several missing values or outliers in the data for various reasons. Thankfully there are many tactics that can be used to handle this issue. For this problem I used the $breast - cancer - wisconsin$ dataset, which has 16 missing values in the "Bare Nuclei" attribute. There are 699 total observations or data points so note that only $\approx 2.29\%$ of the data is missing so immediately it can be seen that changes to these missing points may have fairly minimal effects.

The first thing I did was explore and complete the different ways one can handles these missing values. See Appendix B for all code. The different methods are as follows 1) omit the data point as a whole when one attribute is missing 2) replace the missing value with the average of all other values of the predictor 3) use linear regression and all other attributes to predict the value for the missing point 4) perturb the previous value/prediction by some amount by randomly adding or subtracting some value drawn from a normal distribution. For the last method, I decided to use a normal distribution with mean and standard deviation equal to that of all the other values of the attribute in which there are missing points. Actually extracting and completing these task is not all too difficult by comparing the results is the real way to determine what method might appear best.

For this reason, I chose to construct a SVM model for each of these four missing data handling methods and use the accuracy metric on test data to determine which might be best. I chose a SVM model for each because the class of each data point in binary, the patient's sample is either benign or malignant. For the sake of comparison I did not alter the parameters of the SVM model and I chose a 70/30% training set to test set split for each model. The results that I obtained are summarized in Table 3 below.

| Missing data handling method | Accuracy on test set (%) |
|---|---|
| omit missing points | 98.1 |
| mean imputation | 96.3 |
| regression to impute missing value | 95.4 |
| regression + perturbation from normal dist. | 96.8 |

Table 3: Accuracy of SVM model using different techniques to handle missing data points

Although one could argue that omitting the points altogether is the best approach to this particular dataset, because there are so few missing data points, it is likely that the very small differences in accuracies are due to fitting random effects in the training set. My conclusion is that these varying methods become increasingly important to analyze once the number of missing data points exceeds a higher percentage of the overall number of data points.

# 3 Optimization

## 3.1 Question 15.1

A particular problem where I see optimization as a fantastic solution is the scheduling of a conference/symposium. Let's say that there is only one room/stage available for a speaker and that there have several submissions from grad students, post-docs, and principal investigators. These data points or submissions come from anyone who wants to present at a particular time for a specific window of time (maybe 15 min. increments). For added complexity, say each speaker has an assigned priority value (1-3) where the lowest priority speaker is a grad student and the highest is the PI. My job is to optimize the tradeoff between number of presentations as well as maximizing the total priorities of presenters. Additionally, there is the constraint that there can only be one speaker speaking at a time. So for this example: Variables: presenter request start time, presenter request present length, presenter's priority. Constraints: only one person can speak on the stage at any given time, the presentations can go from 9am to 5pm. Objective function: Maximize the total number of presentations balanced with maximizing the sum of priorities controlled by some tune able parameter for a tradeoff between the two.

# References

[1] https://realpython.com/simpy-simulating-with-python/#how-to-get-started-with-simpy

# Appendix A Simulation code

```
import simpy
import random
import statistics
import numpy

wait_times = []

class Airport(object):
    def __init__(self, env, num_workers, num_queues):
        self.env = env
        self.worker = simpy.Resource(env, num_workers)
        self.queue = simpy.Resource(env, num_queues)

    def id_check(self, passenger):
        yield self.env.timeout(random.expovariate(1.33))

    def personal_check(self, passenger):
        yield self.env.timeout(numpy.random.uniform(0.5, 1.0))

def go_to_airport(env, passenger, airport):
    arrival_time = env.now

    with airport.worker.request() as request:
        yield request
        yield env.process(airport.id_check(passenger))

    with airport.queue.request() as request:
        yield request
        yield env.process(airport.personal_check(passenger))
```

```
    wait_times.append(env.now - arrival_time)

def run_airport(env, num_workers, num_queues):
    airport = Airport(env, num_workers, num_queues)

    for passenger in range(1):
        env.process(go_to_airport(env, passenger, airport))

    while True:
        yield env.timeout(random.expovariate(50))
        passenger += 1
        env.process(go_to_airport(env, passenger, airport))

def avg_wait_time(wait_times):
    average_wait = statistics.mean(wait_times)
    return average_wait

def get_user_input():
    num_workers = input("Input # of workers: ")
    num_queues = input("Input # of queues: ")

    params = [num_workers, num_queues]
    if all(str(i).isdigit() for i in params):  # Check input is valid
        params = [int(x) for x in params]
    else:
        print(
            "Could not parse input. The simulation will use default values:",
            "\n1 cashier, 1 server, 1 usher.",
        )
        params = [1, 1]
    return params

def main():
    # Setup
    random.seed(42)
    num_workers, num_queues = get_user_input()

    # Run the simulation
    env = simpy.Environment()
    env.process(run_airport(env, num_workers, num_queues))
    env.run(until=120)

    # View the results
    mins = avg_wait_time(wait_times)
    print(
      "Running simulation...",
      f"\nThe average wait time is {mins} minutes.",
    )

main()
```

## Appendix B Design of experiments code

```
data <- read.table("breast-cancer-wisconsin.data.txt", header=FALSE, sep = ",")
data <- data[,-1]
print(data)

miss_val <- c()
for (i in 1:nrow(data)) {
  if (data[i,6] == '?') {
    print(i)
    miss_val <- c(miss_val, i)
  }
}
#print(miss_val)
#print(as.numeric(data[-miss_val,7]))
V7_keep <- as.numeric(data[-miss_val,6])
print(mean(V7_keep))
print(sd(V7_keep))

model_missing <- glm(as.numeric(V7) ~ V2+V3+V4+V5+V6+V8+V9+V10+V11,
                     data=data[-miss_val,])
summary(model_missing)
pred <- predict(model_missing, data[miss_val,-6])
print(pred)

perturb <- rnorm(683, mean = 3.544, sd = 3.643)
plot(perturb)
rand_i <- sample(1:683, 16)
rand_oper <- sample(1:2, 1)
perturb_keep <- perturb[rand_i]

perturb_imputed <- c()
for (i in 1:16) {
  r = sample(1:2, 1)
  if (r == 1) {
    perturb_imputed <- c(perturb_imputed, pred[i] + perturb_keep[i])
  } else {
    perturb_imputed <- c(perturb_imputed, pred[i] - perturb_keep[i])
  }
}

##############################################################################

# SVM using removed data
data1 <- data[-miss_val,]
data1 <- transform(data1, V7 = as.integer(V7))
data1 <- as.matrix(data1)
set.seed(123)
n1 <- nrow(data1)
train_idx1 <- sample(1:n1, size=round(n1*0.7), replace = FALSE)
train1 <- data[train_idx1,]
test1 <- data[-train_idx1,]

model1 <- ksvm(data1[train_idx1,-10], data1[train_idx1,10], type="C-svc",
```

```
                kernel="vanilladot", C=1)
a_vector <- colSums(model1@xmatrix[[1]] * model1@coef[[1]])
a0 <- model1@b
pred1 <- predict(model1, data1[-train_idx1,-10])

# frac of model predictions match actual classification
acc1 <- sum(pred1 == data1[-train_idx1,10]) / (n1*0.3)

##############################################################################

# SVM using mean data
data2 <- data
for (i in 1:nrow(data2)) {
  if (data2[i,6] == '?') {
    data2[i,6] = 3.544656
  }
}

data2 <- transform(data2, V7 = as.double(V7))
data2 <- as.matrix(data2)
set.seed(123)
n2 <- nrow(data2)
train_idx2 <- sample(1:n2, size=round(n2*0.7), replace = FALSE)
train2 <- data[train_idx2,]
test2 <- data[-train_idx2,]

model2 <- ksvm(data2[train_idx2,-10], data2[train_idx2,10], type="C-svc",
                kernel="vanilladot", C=1)
a_vector <- colSums(model2@xmatrix[[1]] * model2@coef[[1]])
a0 <- model2@b
pred2 <- predict(model2, data2[-train_idx2,-10])

# frac of model predictions match actual classification
acc2 <- sum(pred2 == data2[-train_idx2,10]) / (n2*0.3)

##############################################################################

# SVM using regression data
data3 <- data
count = 1
for (i in 1:nrow(data3)) {
  if (data3[i,6] == '?') {
    data3[i,6] = pred[count]
    count = count + 1
    print(count)
  }
}
#print(data3)
data3 <- transform(data3, V7 = as.double(V7))
data3 <- as.matrix(data3)
set.seed(123)
n3 <- nrow(data3)
train_idx3 <- sample(1:n3, size=round(n3*0.7), replace = FALSE)
train3 <- data[train_idx3,]
```

```
test3 <- data[-train_idx3,]

model3 <- ksvm(data3[train_idx3,-10], data3[train_idx3,10], type="C-svc",
               kernel="vanilladot", C=1)
a_vector <- colSums(model3@xmatrix[[1]] * model3@coef[[1]])
a0 <- model3@b
pred3 <- predict(model3, data3[-train_idx3,-10])

# frac of model predictions match actual classification
acc3 <- sum(pred3 == data3[-train_idx3,10]) / (n3*0.3)

###############################################################################

# SVM using regression data
data4 <- data
count = 1
for (i in 1:nrow(data4)) {
  if (data4[i,6] == '?') {
    data4[i,6] = perturb_imputed[count]
    count = count + 1
    print(count)
  }
}

data4 <- transform(data4, V7 = as.double(V7))
data4 <- as.matrix(data4)
set.seed(123)
n4 <- nrow(data4)
train_idx4 <- sample(1:n4, size=round(n4*0.7), replace = FALSE)
train4 <- data[train_idx4,]
test4 <- data[-train_idx4,]

model4 <- ksvm(data4[train_idx4,-10], data4[train_idx4,10], type="C-svc",
               kernel="vanilladot", C=1)
a_vector <- colSums(model4@xmatrix[[1]] * model4@coef[[1]])
a0 <- model4@b
pred4 <- predict(model4, data4[-train_idx4,-10])

# frac of model predictions match actual classification
acc4 <- sum(pred4 == data4[-train_idx4,10]) / (n4*0.3)
```