GT Introduction to Analytics Modeling - Week 6 HW

Robert Phillips

June 26, 2017

Question 1

In this problem you, can simulate a simplified airport security system at a busy airport. Passengers arrive according to a Poisson distribution with with a rate of 5 per minute (i.e., mean interarrival rate 0.2 minutes) to the ID/boarding-pass check queue, where there are several servers who each have exponential service time with mean rate 0.75 minutes. [Hint: model them as one block that has more than one resource.] After that, the passengers are assigned to the shortest of the several personal-check queues, where they go through the personal scanner (time is uniformly distributed between 0.5 minutes and 1 minute).

Use the Arena software (PC users) or Python with SimPy (Mac users) to build a simulation of the system, and then vary the number of ID/boarding-pass checkers and personal-check queues to determine how many are needed to keep average wait times below 15 minutes.

I used the following python code using simpy to perform the requested simulation. There are various ways to run the simulation to determine keep the average wait time per person to under 15 minutes. For example, 5 boarding pass queues and 5 will work.

```
import itertools
import simpy
import numpy as np

SIM_TIME = 60*60*1  # Simulation time in seconds

LAMBDA1 = 5. / 60.  # person arrival time per second
MU2 = .75 * 60  # boarding pass check time in seconds

UA = .5 * 60  # lower personal check time in seconds
UB = 1. * 60  # upper personal check time in seconds

wait_times = {}

def person(name, env, bp_queue, pc_queue):
    """A person arrives at the airport."""

    with bp_queue.request() as req:
        # Request one of boarding pass lines
        yield req

        # The boarding pass check process takes some (exp) time
        yield env.timeout(np.random.exponential(scale=MU2))

    with pc_queue.request() as req2:
        #request one of the personal check queues
        yield req2

        # The boarding pass check process takes some (uniform) time
        yield env.timeout(np.random.uniform(low=UA, high=UB))
```

```
    wait_times[name].append(env.now)

    print('%s finished airport queues in %.2f seconds.' % (name,
            env.now - wait_times[name][0]))

def person_generator(env, bp_queue, pc_queue):
    """Generate people that arrive at the airport."""
    for i in itertools.count():
        yield env.timeout(np.random.poisson(lam=LAMBDA1))
        name = 'Person %d' % i
        wait_times[name] = [env.now]
        env.process(person(name, env, bp_queue, pc_queue))

# Create environment and start processes
env = simpy.Environment()
boarding_pass_queue = simpy.Resource(env, 5)
personal_check_queue = simpy.Resource(env, 5)
env.process(person_generator(env, boarding_pass_queue, personal_check_queue))
env.run(until=SIM_TIME)
```

Question 2

The breast cancer data set at http://archive.ics.uci.edu/ml/machine-learning-databases/breast-cancerwisconsin/ has missing values.

1. *Use the mean/mode imputation method to impute values for the missing data.*
2. *Use regression to impute values for the missing data.*
3. *Use regression with perturbation to impute values for the missing data.*
4. *(Optional) Compare the results and quality of classification models (e.g., SVM, KNN) build using:*
    a. *the data sets from questions 1,2,3*
    b. *the data that remains after data points with missing values are removed*
    c. *the data set when a binary variable is introduced to indicate missing values.*

We first load the data, set the column names, and review a summary of the data. Note that we specify "?" as the expected representation of a missing value.

```
data = read.csv("breast-cancer-wisconsin.data.txt", header=F, na.strings="?")
colnames(data) = c("Id","ClumpSize","CellSize","CellShape","Adhesion",
                   "Epithelial","BareNuclei","Bland Chromatin","NormalNucleoli",
                   "Mitoses","Class")

summary(data)
```

```
##        Id              ClumpSize         CellSize        CellShape
##  Min.   :   61634   Min.   : 1.000   Min.   : 1.000   Min.   : 1.000
##  1st Qu.:  870688   1st Qu.: 2.000   1st Qu.: 1.000   1st Qu.: 1.000
##  Median : 1171710   Median : 4.000   Median : 1.000   Median : 1.000
##  Mean   : 1071704   Mean   : 4.418   Mean   : 3.134   Mean   : 3.207
##  3rd Qu.: 1238298   3rd Qu.: 6.000   3rd Qu.: 5.000   3rd Qu.: 5.000
##  Max.   :13454352   Max.   :10.000   Max.   :10.000   Max.   :10.000
##
##     Adhesion        Epithelial       BareNuclei     Bland Chromatin
##  Min.   : 1.000   Min.   : 1.000   Min.   : 1.000   Min.   : 1.000
##  1st Qu.: 1.000   1st Qu.: 2.000   1st Qu.: 1.000   1st Qu.: 2.000
##  Median : 1.000   Median : 2.000   Median : 1.000   Median : 3.000
```

```
##  Mean   : 2.807   Mean   : 3.216   Mean   : 3.545   Mean   : 3.438
##  3rd Qu.: 4.000   3rd Qu.: 4.000   3rd Qu.: 6.000   3rd Qu.: 5.000
##  Max.   :10.000   Max.   :10.000   Max.   :10.000   Max.   :10.000
##                                    NA's   :16
##  NormalNucleoli      Mitoses          Class
##  Min.   : 1.000   Min.   : 1.000   Min.   :2.00
##  1st Qu.: 1.000   1st Qu.: 1.000   1st Qu.:2.00
##  Median : 1.000   Median : 1.000   Median :2.00
##  Mean   : 2.867   Mean   : 1.589   Mean   :2.69
##  3rd Qu.: 4.000   3rd Qu.: 1.000   3rd Qu.:4.00
##  Max.   :10.000   Max.   :10.000   Max.   :4.00
##
```

The summary indicates that we have 16 missing values for Bare Nuclei, which is consistent with the dataset documentation. Therefore we'll setup 3 data sets to answer parts 1 - 3 of this question.

```
#convert the "Class" column to a factor since it only has to values
data$Class = as.factor(data$Class)
#we don't need the id column for our models
data$Id = NULL

#create a vector indicating location of missing values
data.nas = is.na(data$BareNuclei)

#mean value
nas.mean = mean(data$BareNuclei, na.rm=T)
data.mean = data
data.mean$BareNuclei[data.nas] = nas.mean
data.mean$BareNuclei[data.nas]
```

```
##  [1] 3.544656 3.544656 3.544656 3.544656 3.544656 3.544656 3.544656
##  [8] 3.544656 3.544656 3.544656 3.544656 3.544656 3.544656 3.544656
## [15] 3.544656 3.544656
```

```
#regression value, note that we only use the rows with values
data.reg = data
fit.reg = lm(BareNuclei ~ ., data = data.reg[!data.nas,])
data.reg$BareNuclei[data.nas] = predict(fit.reg, data.reg[data.nas,])
data.reg$BareNuclei[data.nas]
```

```
##  [1] 7.201509 3.412194 1.200127 1.588095 1.271663 1.444743 1.960806
##  [8] 1.407689 1.625150 6.343076 1.219350 1.000995 2.005965 1.407689
## [15] 1.200127 1.048844
```

```
#regression value with pertubation, use mean and sd of residuals to draw from a normal dist
set.seed(1)
draw = rnorm(16, mean = mean(fit.reg$residuals), sd = sd(fit.reg$residuals))

data.reg2 = data.reg
data.reg2$BareNuclei[data.nas] = data.reg2$BareNuclei[data.nas] + draw
data.reg2$BareNuclei[data.nas]
```

```
##  [1]  5.9568289  3.7770684 -0.4601551  4.7577044  1.9263509 -0.1854172
##  [7]  2.9292622  2.8746407  2.7691506  5.7363098  4.2230568  1.7755616
## [13]  0.7716431 -2.9926230  3.4352137  0.9595665
```

We can now use our 3 data sets to test performance using ksvm from the kernlab package. As in a previous

assignment, we'll use the 'vanilladot' kernel with C = 100. We also set the type to 'C-svc' since this is a classification exercise. Note that ksvm will automatically omit rows with missing values.

```
require(kernlab)
```

We output a confusion matrix for each model to show how it performs on the training data.

```
#mean based data set
fit1 = ksvm(Class ~ ., data = data.mean, scaled = T, kernal='vanilladot', type='C-svc', C=100)
table(predict(fit1, data.mean) == data.mean$Class)
```

```
##
## TRUE
##  699
```

```
#regression based data set
fit2 = ksvm(Class ~ ., data = data.reg, scaled = T, kernal='vanilladot', type='C-svc', C=100)
table(predict(fit2, data.reg) == data.reg$Class)
```

```
##
## TRUE
##  699
```

```
#regression with pertubtaion based data set
fit3 = ksvm(Class ~ ., data = data.reg2, scaled = T, kernal='vanilladot', type='C-svc', C=100)
table(predict(fit3, data.reg2) == data.reg2$Class)
```

```
##
## TRUE
##  699
```

```
#original data
fit4 = ksvm(Class ~ ., data = data, scaled = T, kernal='vanilladot', type='C-svc', C=100)
table(predict(fit4, data) == data$Class[!data.nas])
```

```
##
## TRUE
##  683
```

```
#boolean value, add new column, remove other one
data.bool = data
data.bool$HasBareNuclei = data.nas
data.bool$BareNuclei = NULL

fit5 = ksvm(Class ~ ., data = data.bool, scaled = T, kernal='vanilladot', type='C-svc', C=100)
table(predict(fit5, data.bool) == data.bool$Class)
```

```
##
## FALSE  TRUE
##     1   698
```

Question 3

Describe a situation or problem from your job, everyday life, current events, etc., for which optimization would be appropriate. What data would you need?

Answer: A popular web site incurs a lot of traffic during some parts of the day and lower traffic at other parts of the day. Traffic also varies based on other factors such as day of the week, seasonal events, and special promotions. A properly performing site is critical to sales and customer satisfaction. However, there

is also a cost to running the infrastructure. In order to optimize this cost, we could first build a regression model to forecast traffic and response time. We would use previously captured log data for this model.

We could then apply this model to determine the cost of the required resources on various cloud platforms along with different approaches to utilizing these platforms. For example, our optimzation model could have variables that represent differnt types of computing power. Our goal would be to minimize the cost (via optimal selction of computer power types) while also maintianing minimal levels needed resources.