homework6_isye6501

Zach Olivier

6/24/2018

Question 13.1

Question:

In this problem you, can simulate a simplified airport security system at a busy airport. Passengers arrive according to a Poisson distribution with λ1 = 5 per minute (i.e., mean interarrival rate μ1 = 0.2 minutes) to the ID/boarding-pass check queue, where there are several servers who each have exponential service time with mean rate μ2 = 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 (PC or 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. [If you're using SimPy, or if you have access to a non-student version of Arena, you can use λ1 = 50 to simulate a busier airport.]

Answer:

I attempted the simulation in Python using SimPy. Please see the text file attached for the simulation code. Overall, I was able to set up a decent simulation of the Airport queues between check-in and scan. I did struggle finding the correct way to keep track of the total number of passengers that entered my system for any given simulation of the airport. I ended up hard coding in an average completion count of 125 passengers entering the system to determine the various wait and system times. In practice, some simulations pushed through less or more than this - which would effect the overall wait time averages per person. Overall I simulated the entire system 100 times to get the estimates below.

In my attempt I saw that a combination of 25 scanners and 10 checkers was adequate to keep wait times for at most 125 people per hour to about 22 minutes of total wait time. This this example the check-in time accounted for an estimate of 18 minutes of the total wait time.

Question 14.1

Question:

The breast cancer data set breast-cancer-wisconsin.data.txt from http://archive.ics.uci.edu/ml/machine-learning-databases/breast-cancer-wisconsin/ (description at http://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+%28Original%29) 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 (1) the data sets from questions 1,2,3; (2) the data that remains after data points with missing values are removed; and (3) the data set when a binary variable is introduced to indicate missing values

Answer:

The code below shows each of the imputation techniques applied to the breast cancer dataset utilizing the mice package. Results for mean, regression, and regression with perturbation are all printed below. Overall there are 16 missing values to be imputed. Printed are the actual values for each imputation technique. **Mean imputation obviously has the most stability, while regression with pertubation has the most variability in its estimates of the missing data.**

Next I fit a random forest model to each dataset and the non-imputed (NAs omitted) dataset to see which technique performs best. **Overall the cross-validation accuracy for each dataset are extremely close. With my test and training splits and seed the mean imputation dataset performed the best with an accuracy of 2.86%. The regression imputation with no pertubation performed the worst with a cross validation accuracy of 3.81%.**

Overall it seems the imputation methods offer only marginal improvement over the base model omitting the 16 rows with NAs. I suspect this could be due to the low amount of missing values (2% of total data). Also using a non-parametric random forest may mask the effects of missing data. A parametric model may have been more sensitive to the missing data.

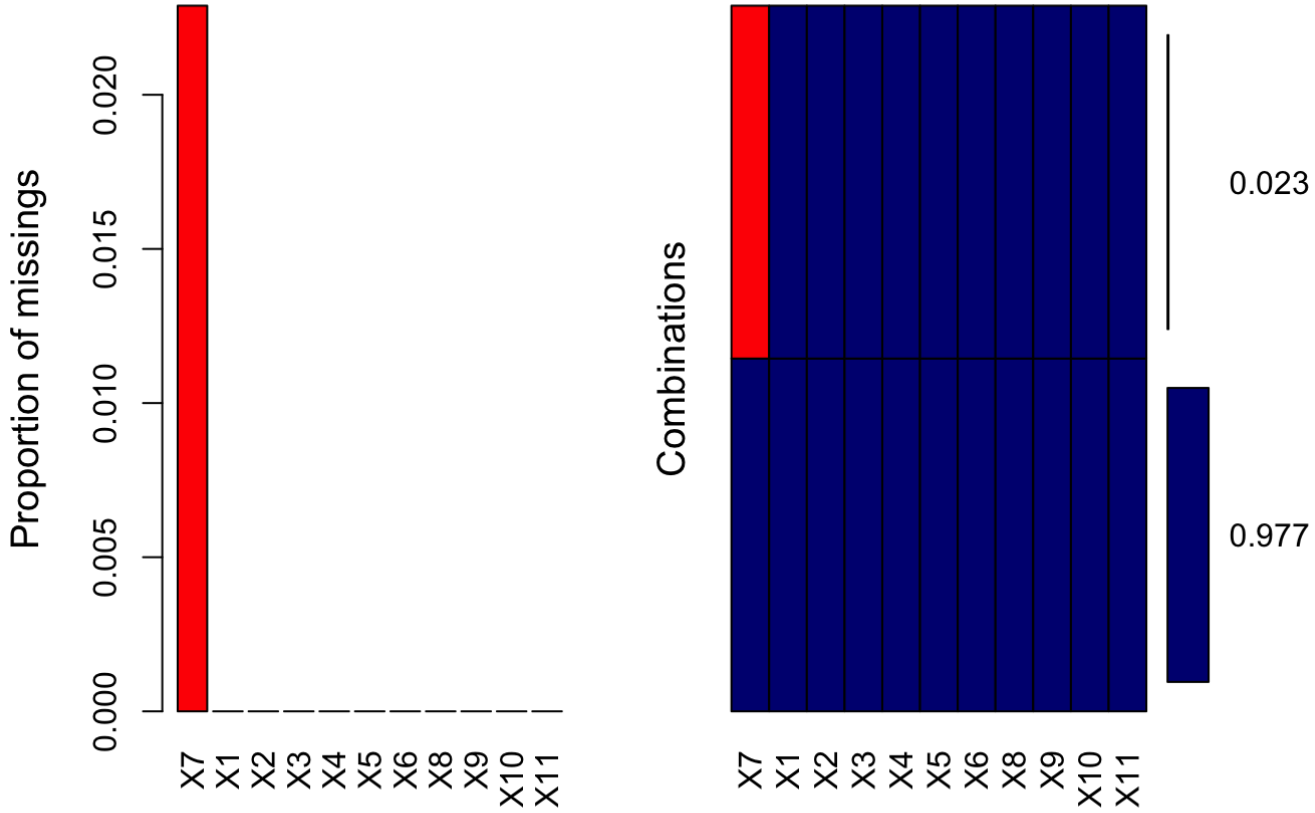
```
set.seed(110)


# read in the crime data
cancer_df = read_delim('breast-cancer-wisconsin.data.txt', delim = ',',
                       col_names = F, na = c('?')) %>%
        as.data.frame() %>%
        mutate(X11 = ifelse(X11 == 2, 'Benign', 'Malignant'))
```

```
# using mice to see missing data - including complete cases
md.pattern(cancer_df[,-11])
```

```
##     X1 X2 X3 X4 X5 X6 X8 X9 X10 X7
## 683  1  1  1  1  1  1  1  1   1  1  0
##  16  1  1  1  1  1  1  1  1   1  0  1
##      0  0  0  0  0  0  0  0   0 16 16
```

```
# plot the missing data
plot_missing <- aggr(cancer_df, col = c('navyblue', 'red'), numbers = T, sortVars = T)
```




```
##
##  Variables sorted by number of missings:
##  Variable      Count
##        X7 0.02288984
##        X1 0.00000000
##        X2 0.00000000
##        X3 0.00000000
##        X4 0.00000000
##        X5 0.00000000
```

```
##         X6 0.00000000
##         X8 0.00000000
##         X9 0.00000000
##        X10 0.00000000
##        X11 0.00000000
```

```r
# imputation using the mice package - mean imputation
mean_impute <- mice(cancer_df, m = 5, meth = 'mean' )
```

```
##
##  iter imp variable
##   1   1  X7
##   1   2  X7
##   1   3  X7
##   1   4  X7
##   1   5  X7
##   2   1  X7
##   2   2  X7
##   2   3  X7
##   2   4  X7
##   2   5  X7
##   3   1  X7
##   3   2  X7
##   3   3  X7
##   3   4  X7
##   3   5  X7
##   4   1  X7
##   4   2  X7
##   4   3  X7
##   4   4  X7
##   4   5  X7
##   5   1  X7
##   5   2  X7
##   5   3  X7
##   5   4  X7
##   5   5  X7
```

```
# look at the values
mean_impute$imp
```

```
## $X1
## NULL
##
## $X2
## NULL
##
## $X3
## NULL
##
## $X4
## NULL
##
## $X5
## NULL
##
## $X6
## NULL
##
## $X7
##            1        2        3        4        5
## 24  3.544656 3.544656 3.544656 3.544656 3.544656
## 41  3.544656 3.544656 3.544656 3.544656 3.544656
## 140 3.544656 3.544656 3.544656 3.544656 3.544656
## 146 3.544656 3.544656 3.544656 3.544656 3.544656
## 159 3.544656 3.544656 3.544656 3.544656 3.544656
## 165 3.544656 3.544656 3.544656 3.544656 3.544656
## 236 3.544656 3.544656 3.544656 3.544656 3.544656
## 250 3.544656 3.544656 3.544656 3.544656 3.544656
## 276 3.544656 3.544656 3.544656 3.544656 3.544656
## 293 3.544656 3.544656 3.544656 3.544656 3.544656
## 295 3.544656 3.544656 3.544656 3.544656 3.544656
## 298 3.544656 3.544656 3.544656 3.544656 3.544656
```

```
## 316 3.544656 3.544656 3.544656 3.544656 3.544656
## 322 3.544656 3.544656 3.544656 3.544656 3.544656
## 412 3.544656 3.544656 3.544656 3.544656 3.544656
## 618 3.544656 3.544656 3.544656 3.544656 3.544656
##
## $X8
## NULL
##
## $X9
## NULL
##
## $X10
## NULL
##
## $X11
## NULL
```

```
# imputation using the mice package - regression ignoring model error
regression_impute <- mice(cancer_df, m = 5, meth = 'norm.predict')
```

```
##
##  iter imp variable
##   1   1  X7
##   1   2  X7
##   1   3  X7
##   1   4  X7
##   1   5  X7
##   2   1  X7
##   2   2  X7
##   2   3  X7
##   2   4  X7
##   2   5  X7
##   3   1  X7
##   3   2  X7
##   3   3  X7
##   3   4  X7
```

```
##   3   5  X7
##   4   1  X7
##   4   2  X7
##   4   3  X7
##   4   4  X7
##   4   5  X7
##   5   1  X7
##   5   2  X7
##   5   3  X7
##   5   4  X7
##   5   5  X7
```

```
# look at the values
regression_impute$imp
```

```
## $X1
## NULL
##
## $X2
## NULL
##
## $X3
## NULL
##
## $X4
## NULL
##
## $X5
## NULL
##
## $X6
## NULL
##
## $X7
##             1         2         3         4         5
## 24  5.3668444 5.3668444 5.3668444 5.3668444 5.3668444
```

```
## 41  8.1906545 8.1906545 8.1906545 8.1906545 8.1906545
## 140 0.8739048 0.8739048 0.8739048 0.8739048 0.8739048
## 146 1.6463451 1.6463451 1.6463451 1.6463451 1.6463451
## 159 1.0732097 1.0732097 1.0732097 1.0732097 1.0732097
## 165 2.1869997 2.1869997 2.1869997 2.1869997 2.1869997
## 236 2.7457909 2.7457909 2.7457909 2.7457909 2.7457909
## 250 2.0127204 2.0127204 2.0127204 2.0127204 2.0127204
## 276 2.3071413 2.3071413 2.3071413 2.3071413 2.3071413
## 293 5.9990541 5.9990541 5.9990541 5.9990541 5.9990541
## 295 1.1204949 1.1204949 1.1204949 1.1204949 1.1204949
## 298 2.6840419 2.6840419 2.6840419 2.6840419 2.6840419
## 316 5.6353998 5.6353998 5.6353998 5.6353998 5.6353998
## 322 1.8585043 1.8585043 1.8585043 1.8585043 1.8585043
## 412 0.8588140 0.8588140 0.8588140 0.8588140 0.8588140
## 618 0.5908040 0.5908040 0.5908040 0.5908040 0.5908040
##
## $X8
## NULL
##
## $X9
## NULL
##
## $X10
## NULL
##
## $X11
## NULL
```

```
# imputation using the mice package - perturbation impute
pert_impute <- mice(cancer_df, m = 5, meth = 'norm.nob')
```

```
##
##  iter imp variable
##   1   1  X7
##   1   2  X7
##   1   3  X7
```

```
##   1   4  X7
##   1   5  X7
##   2   1  X7
##   2   2  X7
##   2   3  X7
##   2   4  X7
##   2   5  X7
##   3   1  X7
##   3   2  X7
##   3   3  X7
##   3   4  X7
##   3   5  X7
##   4   1  X7
##   4   2  X7
##   4   3  X7
##   4   4  X7
##   4   5  X7
##   5   1  X7
##   5   2  X7
##   5   3  X7
##   5   4  X7
##   5   5  X7
```

```
# look at the values
pert_impute$imp
```

```
## $X1
## NULL
##
## $X2
## NULL
##
## $X3
## NULL
##
## $X4
```

```
## NULL
##
## $X5
## NULL
##
## $X6
## NULL
##
## $X7
##              1          2         3          4          5
## 24   5.7869179  4.7406402  2.302210  6.2524916  4.0709613
## 41   7.4748373  6.7262793  8.334035 11.6384786  8.7642454
## 140  3.9060596  2.4417125  4.363849  2.4925851 -0.6053397
## 146  0.8247748  0.6778350  4.556597  2.0040741  0.1009704
## 159  0.5154051 -0.5132453  2.566948  0.3977315 -0.3507075
## 165  0.3575915  1.3694505  2.667865  0.4161266 -0.4992850
## 236  2.5703282  2.0342014 -1.045230  6.1743108  6.8792285
## 250  1.3383906  1.4030520 -2.519233  1.0121447  2.4907062
## 276  5.0084405  1.3109643  2.387699  4.6027361 -2.1749981
## 293  7.4348592  4.4639509  6.707439  6.5828443  6.6087529
## 295  1.3109232  1.6467621  1.922924  3.7852507  1.2340365
## 298  3.3078760  0.1555507  3.083832 -2.6408875 -2.3108355
## 316  8.3256886  5.5347447  4.544594  6.3642593  8.8158951
## 322 -2.3027325 -0.6346507  5.067552  6.0890641  3.9478560
## 412 -1.3609379  0.3361175  3.658503 -1.1463636 -1.0062428
## 618 -3.1601130 -3.2200237  2.826626  4.6910916  4.2031726
##
## $X8
## NULL
##
## $X9
## NULL
##
## $X10
## NULL
##
```

```
## $X11
## NULL
```

```
# cancer with mean impute
cancer_mean_df <- complete(mean_impute)

# cancer with regression prediction impute
cancer_regression_df <- complete(regression_impute)

# cancer with pertubation impute
cancer_pert_df <- complete(pert_impute)




# set up test of random forest on each dataset to compare cross validation accuracy
datasets = list(cancer_df, cancer_mean_df, cancer_regression_df, cancer_pert_df)

# set up list to store model results into
model_output = list(no_imputation = NULL, mean_imputation = NULL, regression_imputation = NULL,
                    pert_imputation = NULL)


# model for loop - fit random forest to each dataset and then extract cv results
for (i in seq_along(datasets)) {

        df = datasets[[i]]

        set.seed(45)

        # non-imputation classification results
        inTrain <- createDataPartition(df$X11, p = .75, list = F)

        # training and test splits
        train <- df[inTrain, ] %>% na.omit()
        test <- df[-inTrain,] %>% na.omit()
```

```
        # fit model
        rf_fit <- train(
                X11 ~ .,
                method = 'rf',
                data = train,
                metric = 'Accuracy',
                trControl = trainControl(
                        method = 'cv',
                        number = 10
                )
                )

        model_output[[i]] = rf_fit

}


# view the results of each model - compare cross validation accuracy
model_output$no_imputation$finalModel
```

```
##
## Call:
##  randomForest(x = x, y = y, mtry = param$mtry)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 2
##
##         OOB estimate of  error rate: 2.92%
## Confusion matrix:
##           Benign Malignant class.error
## Benign       326         9  0.02686567
## Malignant      6       173  0.03351955
```

```
model_output$mean_imputation$finalModel
```

```
##
## Call:
##  randomForest(x = x, y = y, mtry = param$mtry)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 2
##
##         OOB estimate of  error rate: 2.86%
## Confusion matrix:
##           Benign Malignant class.error
## Benign       333        11  0.03197674
## Malignant      4       177  0.02209945
```

```
model_output$regression_imputation$finalModel
```

```
##
## Call:
##  randomForest(x = x, y = y, mtry = param$mtry)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 6
##
##         OOB estimate of  error rate: 3.81%
## Confusion matrix:
##           Benign Malignant class.error
## Benign       332        12  0.03488372
## Malignant      8       173  0.04419890
```

```
model_output$pert_imputation$finalModel
```

```
##
## Call:
##  randomForest(x = x, y = y, mtry = param$mtry)
##                Type of random forest: classification
```

```
##                     Number of trees: 500
## No. of variables tried at each split: 2
##
##         OOB estimate of  error rate: 3.05%
## Confusion matrix:
##           Benign Malignant class.error
## Benign       333        11  0.03197674
## Malignant      5       176  0.02762431
```

Question 15.1

Question:

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:

Developing an optimization model to balance marketing budget, incentive budget, volume requirements, and transaction price requirements would be an excellent application for my current workplace. Many times we ask - how can we support this sales objective without eroding our transaction price?

Having a model that encompasses all these different factors would help us focus on what objectives we need for the current month, quarter or year. This optimization model would likely need to sit on top of other statistical models that provide the inputs to the optimization models. We would need data to support:

```
    - marketing efficiency
    - incentive effiiciency
    - sales volume estimates
    - transaction price estimates
```