Homework 6

6/24/2020

Question 13.2

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 lambda = 50 to simulate a busier airport.]

I ran this simulation in SimPy - see attached file for code.

I programmed the simulation to run for 360 minutes and played around with different combinations of number of boarding pass checkers and personal scanners. I ran the simulation with the same parameters 25 times to get a good average wait time. Then I created a success rate that measured how many runs had an avg wait time of 15 mins or less.

I've summarized my tests in the table below.

Board Checkers	Personal Scanners	Success Rate	Avg Wait Time of 25 Runs
5	5	0.0%	156.351949
15	15	0.0%	108.630255
25	25	0.0%	61.102289
35	35	80.0%	14.283771
36	35	92.0%	13.111045
35	36	76.0%	12.95434
30	30	0.0%	37.761879
32	32	0.0%	28.236285
34	34	0.0%	18.482276
35	34	4.0%	17.17542
34	35	4.0%	18.023931

When I tested 35 board checkers and 35 personal scanners, the success rate was only 80% but the average wait time of the 25 runs was less than 15 mins so I decided to test some other combinations that were around 35.

It looks like the minimum number of boarding checkers and personal scanners to have average wait time less than 15 mins is 35 of each.

Question 14.1

The breast cancer data set has missing values. Use 1. mean/mode imputation method to impute values for missing data. 2. Use regression to impute values for 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.

1. Using mean/mode imputation to impute missing values.

Importing data, adding column names and changing the class variable so that 1 = malignant and 0 = benign.

```
data <- read.csv('http://archive.ics.uci.edu/ml/machine-learning-databases/breast-cancer-wisconsin/brea
colnames(data) = c("ID", "clump_thickness", "uniform_cell_size", "uniform_cell_shape", "marginal_adhesi

data$class[data$class == 2] = 0
data$class[data$class == 4] = 1
```

Let's see which columns have missing values:

```
data_mean<-data.frame(data)
sapply(data_mean, function(x) sum(is.na(x)))
```

```
##                 ID    clump_thickness  uniform_cell_size uniform_cell_shape
##                  0                  0                  0                  0
##  marginal_adhesion    single_epi_cell        bare_nuclei    bland_chromatin
##                  0                  0                 16                  0
##    normal_nucleoli            mitoses              class
##                  0                  0                  0
```

It looks like there are 16 missing values in the bare_nuclei column. Out of 699 data points, this is around 2.3% of the total data so we should be okay to impute the missing values without creating indicator variables.

Using mean to impute the missing values and checking there are still missing values after replacing them with the mean:

```
# Mean of bare nuclei before imputation
mean(data_mean$bare_nuclei, na.rm = TRUE)
```

```
## [1] 3.544656
```

```
data_mean$bare_nuclei[is.na(data_mean$bare_nuclei)] = round(mean(data_mean$bare_nuclei, na.rm = TRUE), 

# Checking that there are no more null values
sapply(data_mean, function(x) sum(is.na(x)))
```

```
##                 ID    clump_thickness  uniform_cell_size uniform_cell_shape
##                  0                  0                  0                  0
##  marginal_adhesion    single_epi_cell        bare_nuclei    bland_chromatin
##                  0                  0                  0                  0
##    normal_nucleoli            mitoses              class
##                  0                  0                  0
```

```
# Mean of bare nuclei after imputation
mean(data_mean$bare_nuclei, na.rm = TRUE)
```

```
## [1] 3.555079
```

2. Use regression to impute values for missing data

Using the mice library to generate imputed values using norm.predict as the method (it uses linear regression to impute the missing values) and use the complete function to get the completed data set back with the imputed values. Source: https://statisticsglobe.com/regression-imputation-stochastic-vs-deterministic/

```
imputed_vals <- mice(data, method = 'norm.predict', m = 1, seed = 8010)
```

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

```
data_reg <- complete(imputed_vals)

# Rounding to the nearest int
data_reg$bare_nuclei = round(data_reg$bare_nuclei, digits = 0)
summary(data_reg)
```

```
##        ID            clump_thickness  uniform_cell_size uniform_cell_shape
##  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
##  marginal_adhesion single_epi_cell   bare_nuclei     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   : 3.512   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
##  normal_nucleoli     mitoses           class
##  Min.   : 1.000   Min.   : 1.000   Min.   :0.0000
##  1st Qu.: 1.000   1st Qu.: 1.000   1st Qu.:0.0000
##  Median : 1.000   Median : 1.000   Median :0.0000
##  Mean   : 2.867   Mean   : 1.589   Mean   :0.3448
##  3rd Qu.: 4.000   3rd Qu.: 1.000   3rd Qu.:1.0000
##  Max.   :10.000   Max.   :10.000   Max.   :1.0000
```

```
# Checking that there are no more null values
sapply(data_reg, function(x) sum(is.na(x)))
```

```
##                 ID    clump_thickness  uniform_cell_size uniform_cell_shape
##                  0                  0                  0                  0
##  marginal_adhesion    single_epi_cell        bare_nuclei    bland_chromatin
##                  0                  0                  0                  0
##    normal_nucleoli            mitoses              class
##                  0                  0                  0
```

Using regression to impute the missing values, it looks like the avg of the populated bare_nuclei column has decreased slightly to 3.512 vs 3.555079, the avg of the column when missing values were imputed using the mean

3. Use regression with perturbation to impute values for the missing data.

Using the same mice library, this time using the norm.nob method (Linear regression ignoring model error):

```
imputed_vals2 <- mice(data, method = 'norm.nob', m = 1, seed = 8010)
```

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

```
data_pert <- complete(imputed_vals2)

# Rounding to the nearest int
data_pert$bare_nuclei = round(data_pert$bare_nuclei, digits = 0)
summary(data_pert)
```

```
##        ID           clump_thickness  uniform_cell_size uniform_cell_shape
##  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    
##  marginal_adhesion single_epi_cell   bare_nuclei     bland_chromatin 
##  Min.   : 1.000    Min.   : 1.000   Min.   :-3.000   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   : 3.501   Mean   : 3.438  
##  3rd Qu.: 4.000    3rd Qu.: 4.000   3rd Qu.: 5.500   3rd Qu.: 5.000  
##  Max.   :10.000    Max.   :10.000   Max.   :10.000   Max.   :10.000  
##  normal_nucleoli     mitoses           class       
##  Min.   : 1.000   Min.   : 1.000   Min.   :0.0000  
##  1st Qu.: 1.000   1st Qu.: 1.000   1st Qu.:0.0000  
##  Median : 1.000   Median : 1.000   Median :0.0000  
##  Mean   : 2.867   Mean   : 1.589   Mean   :0.3448  
##  3rd Qu.: 4.000   3rd Qu.: 1.000   3rd Qu.:1.0000  
##  Max.   :10.000   Max.   :10.000   Max.   :1.0000
```

```
# Checking that there are no more null values
sapply(data_reg, function(x) sum(is.na(x)))
```

```
##                  ID    clump_thickness  uniform_cell_size uniform_cell_shape 
##                   0                  0                  0                  0 
##  marginal_adhesion    single_epi_cell        bare_nuclei    bland_chromatin
##                  0                  0                  0                  0
##    normal_nucleoli            mitoses              class
##                  0                  0                  0
```

Looking at the summary for bare_nuclei, it looks like I have to do some adjustments to make sure the values are between 1 and 10. If the value is negative, I'll set it to 1.

```
data_pert$bare_nuclei[data_pert$bare_nuclei <= 0] = 1
summary(data_pert)
```

```
##        ID           clump_thickness  uniform_cell_size uniform_cell_shape
##  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
##  marginal_adhesion single_epi_cell   bare_nuclei     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   : 3.515   Mean   : 3.438
##  3rd Qu.: 4.000    3rd Qu.: 4.000   3rd Qu.: 5.500   3rd Qu.: 5.000
##  Max.   :10.000    Max.   :10.000   Max.   :10.000   Max.   :10.000
##  normal_nucleoli     mitoses           class
##  Min.   : 1.000   Min.   : 1.000   Min.   :0.0000
##  1st Qu.: 1.000   1st Qu.: 1.000   1st Qu.:0.0000
##  Median : 1.000   Median : 1.000   Median :0.0000
##  Mean   : 2.867   Mean   : 1.589   Mean   :0.3448
##  3rd Qu.: 4.000   3rd Qu.: 1.000   3rd Qu.:1.0000
##  Max.   :10.000   Max.   :10.000   Max.   :1.0000
```

Similar to the regression imputation, the avg for the completed bare_nuclei column is lower than the avg of the column when missing values are imputed using mean.

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.

To classify the tumors, I'll use KNN and the different data sets to generate classification models. I'll use the classification accuracy to compare the effectiveness of the models.

1. Mean Imputation

```
# Creating train and test sets
in_train = createDataPartition(y = data_mean$class, p = 0.8, list = FALSE)
train = data_mean[in_train,]
test = data_mean[-in_train,]

#Creating the kknn model where k = 2
set.seed(8010)
```

```r
mean_knn <- kknn(class~., train, test, k=2, scale=TRUE)

#Predicted values and checking for accurary
mean_yhat <- round(fitted(mean_knn))
mean_acc <- sum(mean_yhat == test$class) / nrow(test)
```

2. Regression

```r
# Creating train and test sets
in_train = createDataPartition(y = data_reg$class, p = 0.8, list = FALSE)
train = data_reg[in_train,]
test = data_reg[-in_train,]

#Creating the kknn model where k = 2
set.seed(8010)
reg_knn <- kknn(class~., train, test, k=2, scale=TRUE)

#Predicted values and checking for accurary
reg_yhat <- round(fitted(reg_knn))
reg_acc <- sum(reg_yhat == test$class) / nrow(test)
```

3. Regression with Pertubation

```r
# Creating train and test sets
in_train = createDataPartition(y = data_pert$class, p = 0.8, list = FALSE)
train = data_pert[in_train,]
test = data_pert[-in_train,]

#Creating the kknn model where k = 2
set.seed(8010)
pert_knn <- kknn(class~., train, test, k=2, scale=TRUE)

#Predicted values and checking for accurary
pert_yhat <- round(fitted(pert_knn))
pert_acc <- sum(pert_yhat == test$class) / nrow(test)

sum(pert_yhat == test$class)
```

```
## [1] 132
```

4. Original Data where Missing Values are Removed

```r
data_removed <- na.omit(data)

# Creating train and test sets
in_train = createDataPartition(y = data_removed$class, p = 0.8, list = FALSE)
train = data_removed[in_train,]
test = data_removed[-in_train,]

#Creating the kknn model where k = 2
```

```r
set.seed(8010)
removed_knn <- kknn(class~., train, test, k=2, scale=TRUE)

#Predicted values and checking for accurary
removed_yhat <- round(fitted(removed_knn))
removed_acc <- sum(removed_yhat == test$class) / nrow(test)
```

5. Data where a binary variable is introduced to indicate missing values

```r
#Creating indicator flag
data_bin <- data
data_bin$bare_nuclei_ind <- is.na(data_bin$bare_nuclei)
data_bin$bare_nuclei_ind[data_bin$bare_nuclei_ind == TRUE] = 1
data_bin$bare_nuclei_ind[data_bin$bare_nuclei_ind == FALSE] = 0

# Creating train and test sets
in_train = createDataPartition(y = data_bin$class, p = 0.8, list = FALSE)
train = data_bin[in_train,]
test = data_bin[-in_train,]

#Creating the kknn model where k = 2
set.seed(8010)
binary_knn <- kknn(class~., train, test, k=2, scale=TRUE)

#Predicted values and checking for accurary
binary_yhat <- round(fitted(binary_knn))
binary_acc <- sum(binary_yhat == test$class) / nrow(test)
```

Comparing the results:

```r
print("Accurary Results")
```

```
## [1] "Accurary Results"
```

```r
print(paste0("Mean Imputation: ", mean_acc))
```

```
## [1] "Mean Imputation: 0.956834532374101"
```

```r
print(paste0("Regression Imputation: ", reg_acc))
```

```
## [1] "Regression Imputation: 0.949640287769784"
```

```r
print(paste0("Regression with Pertubation Imputation: ", pert_acc))
```

```
## [1] "Regression with Pertubation Imputation: 0.949640287769784"
```

```r
print(paste0("Data with missing data points removed: ", removed_acc))
```

```
## [1] "Data with missing data points removed: 0.955882352941177"
```

```r
print(paste0("Data with missing data indicator: ", binary_acc))
```

```
## [1] "Data with missing data indicator: 0.669064748201439"
```

It looks like all data sets generated similar accuracies, with the exception of the data set with the indicator variable. The data that produces the best accuracy is the data where the mean is used to impute the data. The data with the missing points removed is second best. Looking at how close the accuracies are, I would feel comfortable using the data with the missing points removed as it's the simplest to code and explain and I wouldn't be sacrificing alot of accuracy.

Question 15.1

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

In my job as a data analyst for a loyalty company, I can use optimization to determine the number of emails to send out in order to maxmize the number of survey completions, given a member's propensity to complete the survey. Variables = Number of Emails Constraints = Cost of Email Deployment have to be less than or equal to budget, Number of Emails > 0 Objective Function = maximize the number of survey completions within the constraints