Homework10

N/A

2023-10-28

Question 14.1

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.

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.

I'm using R to read data from a text file named "breast-cancer-wisconsin.data.txt" and storing it in a variable I've called "data." I've set it up so that character columns are treated as characters, assuming there's no header row, and using commas as column separators. I've listed the descirptions of each variable here for conrtext.

1. Sample code number: id number

2. Clump Thickness: 1 - 10

3. Uniformity of Cell Size: 1 - 10

4. Uniformity of Cell Shape: 1 - 10

5. Marginal Adhesion: 1 - 10

6. Single Epithelial Cell Size: 1 - 10

7. Bare Nuclei: 1 - 10

8. Bland Chromatin: 1 - 10

9. Normal Nucleoli: 1 - 10

10. Mitoses: 1 - 10

11. Class: (2 for benign, 4 for malignant)

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

```
## Rows: 699 Columns: 11
## ── Column specification ────────────────────────────────────────────────────────
## Delimiter: ","
## dbl (11): X1, X2, X3, X4, X5, X6, X7, X8, X9, X10, X11
##
## ℹ Use `spec()` to retrieve the full column specification for this data.
## ℹ Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

I now run some summary statistics on it and explore it a little.

```
str(data)
```

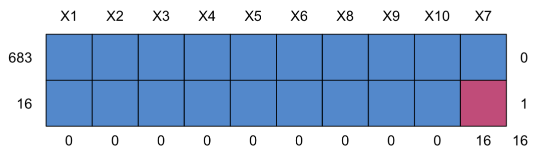
```
## 'data.frame':    699 obs. of  11 variables:
##  $ X1 : num  1000025 1002945 1015425 1016277 1017023 ...
##  $ X2 : num  5 5 3 6 4 8 1 2 2 4 ...
##  $ X3 : num  1 4 1 8 1 10 1 1 1 2 ...
##  $ X4 : num  1 4 1 8 1 10 1 2 1 1 ...
##  $ X5 : num  1 5 1 1 3 8 1 1 1 1 ...
##  $ X6 : num  2 7 2 3 2 7 2 2 2 2 ...
##  $ X7 : num  1 10 2 4 1 10 10 1 1 1 ...
##  $ X8 : num  3 3 3 3 3 9 3 3 1 2 ...
##  $ X9 : num  1 2 1 7 1 7 1 1 1 1 ...
##  $ X10: num  1 1 1 1 1 1 1 1 5 1 ...
##  $ X11: chr  "Benign" "Benign" "Benign" "Benign" ...
```

```
summary(data)
```

```
##        X1                 X2               X3               X4        
##  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  
##                                                                       
##        X5               X6               X7               X8        
##  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                       
##        X9              X10             X11           
##  Min.   : 1.000   Min.   : 1.000   Length:699        
##  1st Qu.: 1.000   1st Qu.: 1.000   Class :character  
##  Median : 1.000   Median : 1.000   Mode  :character  
##  Mean   : 2.867   Mean   : 1.589                     
##  3rd Qu.: 4.000   3rd Qu.: 1.000                     
##  Max.   :10.000   Max.   :10.000                     
##                                                      
```

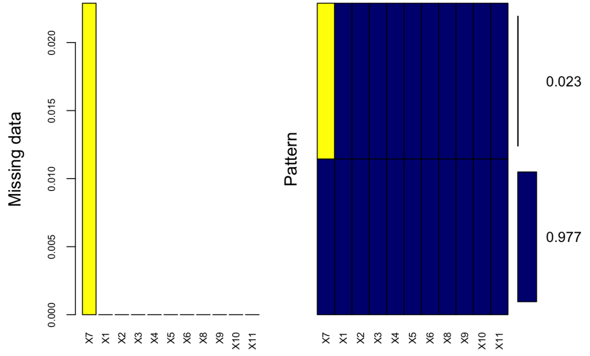
Now I attempt to find the missing data in our df. This R code uses the aggr function to create a visual plot that summarizes missing data patterns in a given dataset. It represents missing data in yellow and non-missing data in navy blue. The plot includes labels for variables, the number of missing and non-missing values, and sorts variables based on the percentage of missing data. This visual representation helps analysts quickly identify columns with missing values and their distribution within the dataset. We can see that the Bare Nuclei column has 16 missing values.

```
md.pattern(data[,-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
```

```
mice_plot <- aggr(data, col=c('navyblue','yellow'),numbers=TRUE, sortVars=TRUE,
                  labels=names(data), cex.axis=.7,gap=3, ylab=c("Missing data","Pattern"))
```



```
##
##  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
```

Now I will perform the mean imputation using the mice package.In the "mice" package in R, the "mean impute" method is a straightforward approach used to handle missing values in a dataset. It involves replacing missing values in a variable with the mean (average) value of the observed data in the same variable. While this method is easy to apply and suitable for numeric variables, it has limitations when dealing with more complex missing data patterns or when imputed values are expected to represent population averages. The "mice" package provides a framework for multiple imputation, which offers a more robust and statistically rigorous approach to handling missing data.After using the mean method to impute data, I'm now going to use the mice package to incorporate regression to impute the values for missing data. Lastly I use the mice package to use regression with pertubation to impute the values for the missing data. In the "mice" package in R, the "regression with perturbation" method is used for imputing missing data in a dataset. This approach employs regression models to estimate missing values in a target variable while introducing randomness, or "perturbation," to the imputed values. This randomness accounts for the uncertainty associated with imputation, preventing overly deterministic imputations. The process involves selecting a target variable, constructing a regression model using other non-missing variables, introducing perturbation to the imputed values, and repeating this process in a multiple imputation framework. The result is a set of imputed datasets, each with perturbed values, offering a range of plausible imputations. This method is particularly valuable for handling complex data patterns and producing more realistic imputed data that considers imputation uncertainty.

```
mean_impute <- mice(data, 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
```

```
## Warning: Number of logged events: 1
```

```
regression_impute <- mice(data, 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
```

```
## Warning: Number of logged events: 1
```

```
regression_impute_pert <- mice(data, 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
```

```
## Warning: Number of logged events: 1
```

The provided code generates three distinct dataframes: mean_df, regression_df, and pert_df," each representing the "cancer" dataset with missing values imputed using different methods. mean_dfuses the "mean impute" method to fill missing values with column means, whileregression_dfemploys a "regression prediction impute" based on regression models, andpert_df` introduces perturbation into the regression-based imputation to address uncertainty. These dataframes now store the imputed data, facilitating subsequent analysis or comparisons to evaluate the impact of diverse imputation techniques on the dataset.

```
mean_df <- complete(mean_impute)
regression_df <- complete(regression_impute)
pert_df <- complete(regression_impute_pert)
```

Next I define a function to fit a random forest model and return the results. I also create an empty list to store the model results, define a list of datasets and then Loop through the datasets and fit random forests, storing results in the list

```
fit_random_forest <- function(data) {
  inTrain <- createDataPartition(data$X11, p = 0.75, list = FALSE)
  train_data <- data[inTrain, ] %>% na.omit()
  test_data <- data[-inTrain, ] %>% na.omit()

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

  return(rf_fit)
}

model_output <- list()

datasets = list(data, mean_df, regression_df, pert_df)



for (i in 1:length(datasets)) {
  model_output[[i]] <- fit_random_forest(datasets[[i]])
}
```

Lets take a look at each model and check the accuracy. It seems like all of the imputation methods are pretty solid. However I'm sure that they only really offer marginal improvement over the base model(before the 16 values were ommited.) There wasn't too much missing data, so I'm not sure how much we would have benefitted with these imputation methods. Nonetheless, it was good practice for datasets that may contain many more missing values.

```
results_dataset1 <- model_output[[1]]
results_dataset1
```

```
## Random Forest
##
## 512 samples
##  10 predictor
##   2 classes: 'Benign', 'Malignant'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 461, 461, 460, 461, 461, 461, ...
## Resampling results across tuning parameters:
##
##   mtry  Accuracy   Kappa
##    2    0.9765460  0.9487877
##    6    0.9706637  0.9353947
##   10    0.9687029  0.9307015
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was mtry = 2.
```

```
results_dataset2 <- model_output[[2]]
results_dataset2
```

```
## Random Forest
##
## 525 samples
##  10 predictor
##   2 classes: 'Benign', 'Malignant'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 473, 473, 472, 472, 473, 472, ...
## Resampling results across tuning parameters:
##
##   mtry  Accuracy   Kappa
##    2    0.9771771  0.9497576
##    6    0.9638244  0.9198634
##   10    0.9542453  0.8990431
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was mtry = 2.
```

```
results_dataset3 <- model_output[[3]]
results_dataset3
```

```
## Random Forest
##
## 525 samples
##  10 predictor
##   2 classes: 'Benign', 'Malignant'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 472, 472, 472, 472, 473, 473, ...
## Resampling results across tuning parameters:
##
##   mtry  Accuracy   Kappa
##    2    0.9713716  0.9367780
##    6    0.9656386  0.9241286
##   10    0.9618650  0.9161660
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was mtry = 2.
```

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?

ANS: During a global pandemic like COVID-19, optimizing the supply chain becomes paramount for efficiently distributing essential medical supplies, personal protective equipment, and vaccines to various regions. To achieve this, a wealth of data is essential. Demand forecasts help predict the need for medical resources, while supply availability data provides insights into manufacturing and inventory levels. Transportation data is crucial for planning efficient delivery routes, and geospatial information ensures the strategic positioning of healthcare facilities and distribution centers. Real-time inventory data allows for monitoring stock levels, and regulatory data ensures compliance with customs and trade regulations. Finally, cost data offers insights into the expenses involved in procurement and logistics. By employing optimization models that utilize this data, supply chain managers can make informed decisions, allocate resources effectively, minimize delays, and ensure that critical medical supplies reach areas with the greatest need, ultimately contributing to saving lives and bolstering healthcare systems during a pandemic.