Homework Week 10

Chris Messer

2022-10-29

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. 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.

> See analysis below for answers to 1 through 4.

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?

> Let's say I am currently building a machine that can draw things with different color pens. It needs to draw a complete picture using only dots spaced at different intervals, and it needs to do that moving using two arms that can move across two axis, up and down and left and right.
>
> What if I needed to do this while moving the arms as little as possible to prevent wear and tear on the gears that move the arms? We could have this machine draw one dot at a time, starting on one end, making a dot, then going to somewhere random, and making another dot, etc. but that would ean the arms are moving up and down for no reason.
>
> We can't just move in a grid pattern systematically, going left to right, then down a row, and repeating, because we would have to switch the pen way too frequently. So we need a way to optimize the path the pen takes, with one pen color at a time.
>
> So we would want to optimize the order of the coordinates that each dot is placed, so that it does it one color at a time, and moves the arms as little as possible.
>
> We would need to know the dimensions of the drawing, the number of colors, and the coordinates of the dots, and the color of each dot.
>
> Our limitations would be things like the machine has to draw on the paper, the machine can only use one color at a time, and the machine must place all of the dots.
>
> We would be minimizing the distance the mechanical arms travel.

Analysis

```
data <- read.csv('breast-cancer-wisconsin.data.txt')
str(data)
```

```
## 'data.frame':    698 obs. of  11 variables:
##  $ X1000025: int  1002945 1015425 1016277 1017023 1017122 1018099 1018561 1033078 1033078 1035283 ...
##  $ X5      : int  5 3 6 4 8 1 2 2 4 1 ...
##  $ X1      : int  4 1 8 1 10 1 1 1 2 1 ...
##  $ X1.1    : int  4 1 8 1 10 1 2 1 1 1 ...
##  $ X1.2    : int  5 1 1 3 8 1 1 1 1 1 ...
##  $ X2      : int  7 2 3 2 7 2 2 2 2 1 ...
##  $ X1.3    : chr  "10" "2" "4" "1" ...
##  $ X3      : int  3 3 3 3 9 3 3 1 2 3 ...
##  $ X1.4    : int  2 1 7 1 7 1 1 1 1 1 ...
##  $ X1.5    : int  1 1 1 1 1 1 1 5 1 1 ...
##  $ X2.1    : int  2 2 2 2 4 2 2 2 2 2 ...
```

Looks like our column X1.3 is type chr, but the data description says this column should be continuous integer from 1-10, so lets convert this to an integer.

```
data$X1.3 <- as.integer(data$X1.3)
```

```
## Warning: NAs introduced by coercion
```

```
data$X2.1 <- as.factor(data$X2.1)

#Now lets call summary() on the new data frame to see if we have any n/a fields
summary(data)
```

```
##     X1000025             X5               X1              X1.1       
##  Min.   :   61634   Min.   : 1.000   Min.   : 1.000   Min.   : 1.000  
##  1st Qu.:  870258   1st Qu.: 2.000   1st Qu.: 1.000   1st Qu.: 1.000  
##  Median : 1171710   Median : 4.000   Median : 1.000   Median : 1.000  
##  Mean   : 1071807   Mean   : 4.417   Mean   : 3.138   Mean   : 3.211  
##  3rd Qu.: 1238354   3rd Qu.: 6.000   3rd Qu.: 5.000   3rd Qu.: 5.000  
##  Max.   :13454352   Max.   :10.000   Max.   :10.000   Max.   :10.000  
##                                                                        
##       X1.2              X2              X1.3              X3        
##  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   : 2.809   Mean   : 3.218   Mean   : 3.548   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                        
##       X1.4            X1.5        X2.1   
##  Min.   : 1.00   Min.   : 1.00   2:457  
##  1st Qu.: 1.00   1st Qu.: 1.00   4:241  
##  Median : 1.00   Median : 1.00          
##  Mean   : 2.87   Mean   : 1.59          
##  3rd Qu.: 4.00   3rd Qu.: 1.00          
##  Max.   :10.00   Max.   :10.00          
## 
```

Looks like the column we converted from a chr to an int value contains some non-integer characters, which R has determined are NA values. Lets check those out.

```
data.na_rows <- which(is.na(data$X1.3))
data[data.na_rows,]
```

```
##     X1000025 X5 X1 X1.1 X1.2 X2 X1.3 X3 X1.4 X1.5 X2.1
## 23   1057013  8  4    5    1  2   NA  7    3    1    4
## 40   1096800  6  6    6    9  6   NA  7    8    1    2
## 139  1183246  1  1    1    1  1   NA  2    1    1    2
## 145  1184840  1  1    3    1  2   NA  2    1    1    2
## 158  1193683  1  1    2    1  3   NA  1    1    1    2
## 164  1197510  5  1    1    1  2   NA  3    1    1    2
## 235  1241232  3  1    4    1  2   NA  3    1    1    2
## 249   169356  3  1    1    1  2   NA  3    1    1    2
## 275   432809  3  1    3    1  2   NA  2    1    1    2
## 292   563649  8  8    8    1  2   NA  6   10    1    4
## 294   606140  1  1    1    1  2   NA  2    1    1    2
## 297    61634  5  4    3    1  2   NA  2    3    1    2
## 315   704168  4  6    5    6  7   NA  4    9    1    2
## 321   733639  3  1    1    1  2   NA  3    1    1    2
## 411  1238464  1  1    1    1  1   NA  2    1    1    2
## 617  1057067  1  1    1    1  1   NA  1    1    1    2
```

Now that we know what rows have N/A's, and that they are in the column, we can start using various methods to impute. Lets start with mean and median imputing. We can do this one of two ways, either remove the values manually, or remove them with a built in function of mean(), na.rm=T/F.

```
#manual
data.mean <- mean(data[-data.na_rows,]$X1.3)
#built in function
data.mean <- mean(data$X1.3, na.rm=T)


#first make a copy of the dataset
data.mean_set <- data
data.mean_set[data.na_rows,]$X1.3 <- round(data.mean)

data.mean_set$X1.3 <- as.integer(data.mean_set$X1.3)
```

Now for the mode, there is no built in function for mode, so let's write one. I am adding an _ after it, because mode() is a function, it just is not what we think it is.

```
mode_ <- function(v) {
   uniqv <- unique(v)
   uniqv[which.max(tabulate(match(v, uniqv)))]
}

#store the mode
data.mode <- mode_(data$X1.3)

#first make a copy of the dataset
data.mode_set <- data
data.mode_set[data.na_rows,]$X1.3 <- data.mode
```

Regression Imputation

For regression imputation, we want to build a regression model that predicts our missing values, based on the existing values as the response. We have a couple of data preparation steps to make before we start.

```
#We want to take out the original response column and the label column. We also want to take out the rows of miss
ing data, as that is the data set we want to predict on.
model_set <- data[-data.na_rows, 2:10]
library(dplyr)
```

```
## 
## Attaching package: 'dplyr'
```

```
## The following objects are masked from 'package:stats':
## 
##     filter, lag
```

```
## The following objects are masked from 'package:base':
## 
##     intersect, setdiff, setequal, union
```

```
#move response variable to end, will make leaps easier to read
model_set <- model_set %>% relocate(X1.3, .after = X1.5)
```

Now what values do we want to use as the predictors? We have learned several ways to do this by now. stepwise, backwards elimination, forward elimination, lasso, ridge, elastic-net, oh my! I'm going to cheat a little bit here since that is not the focus of this assignment. I will use the leaps package which exhaustively tries all combination of predicting features and returns the best combination. I like this package because in my experience, it returns the best model without having to cross validate and compare different variable selection approaches.

Plus, I get to introduce the grader to a new approach!

```
library(leaps)
leaps.regression <- leaps(model_set[,-9],model_set$X1.3,nbest = 1)
which.min(leaps.regression$Cp)
```

```
## [1] 4
```

```
leaps.regression$which[4,]
```

```
##     1     2     3     4     5     6     7     8 
##  TRUE FALSE  TRUE  TRUE FALSE  TRUE FALSE FALSE
```

Just wanted to show you an example output of leaps. It is telling us that 4 predictors is the best combination, with those four being columns 1, 3, 4, 6 from the (reordered, with X1.3 at the end, and the data label dropped) data set.

Now lets set up our linear model, and cross validate. We don't really nee to cross validate, since we are not comparing this model against any other. Typically we would just separate the data set into train and test data, but we can still use cross validation and just report the cross validation score as the accuracy score, since we are not comparing it to any other models.

```
model_set.best <- cbind(model_set[,which(leaps.regression$which[4,] == T)], X1.3 = model_set$X1.3)

data.m1 <- lm(X1.3~., data = model_set.best)
summary(data.m1)
```

```
## 
## Call:
## lm(formula = X1.3 ~ ., data = model_set.best)
## 
## Residuals:
##     Min      1Q  Median      3Q     Max 
## -9.8105 -0.9436 -0.3130  0.6869  8.6870 
## 
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)    
## (Intercept) -0.53519    0.17524  -3.054  0.00235 ** 
## X5           0.22688    0.04125   5.500 5.40e-08 ***
## X1.1         0.31628    0.05092   6.212 9.15e-10 ***
## X1.2         0.33190    0.04434   7.485 2.23e-13 ***
## X3           0.32438    0.05611   5.782 1.13e-08 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
## 
## Residual standard error: 2.275 on 677 degrees of freedom
## Multiple R-squared:  0.6128, Adjusted R-squared:  0.6105 
## F-statistic: 267.9 on 4 and 677 DF,  p-value: < 2.2e-16
```

```
library(DAAG)
#cross validate and calculate the r^2 value for
data.m1.cv <- cv.lm(model_set.best, data.m1, m=5, seed=10, printit=F, plotit =F)
SSres1 <- attr(data.m1.cv,"ms")*nrow(model_set.best)
SStotal1 <- sum((model_set.best$X1.3 - mean(model_set.best$X1.3))^2)
rs1 <- 1- SSres1/SStotal1
rs1
```

```
## [1] 0.608062
```

Great, our model has an R^2 value of .61. Lets now use our model to predict and impute the missing values.

```
# Create a copy of the data set
data.regression_set <- data
#predict the missing values and round them to the nearest integer
predicted_values <- (round(predict(data.m1, data[data.na_rows,])))
#store the new values in the df
data.regression_set[data.na_rows,]$X1.3 <- as.integer(predicted_values)
#convert the new values back into integer values
data.regression_set$X1.3 <- as.integer(data.regression_set$X1.3)
```

Perturbation

```
perturb_values <- rnorm(nrow(data[data.na_rows,]), predicted_values, sd(predicted_values))
perturb_values
```

```
##  [1]  4.74777589 10.58987851  0.06488335  2.13952342 -2.84820513  2.71984795
##  [7]  5.18190974  2.52514087  2.41000924  6.13543423 -0.24434204  5.59407254
## [13]  6.05605407  0.55729515 -1.61013367  0.07600197
```

Some of our values are negative, so lets round them all to the nearest integer and then push them back to the range of 1-10.

```
perturb_values <- round(perturb_values)
perturb_values[perturb_values < 1] <- 1
perturb_values[perturb_values >10] <- 10
```

Now store in the new data frame.

```
data.perturb_set <- data
data.perturb_set[data.na_rows,]$X1.3 <-perturb_values
data.perturb_set$X1.3 <- as.integer(data.perturb_set$X1.3)
```

Comparing different models

First lets take care of creating the two extra data sets this question asks about, the data set with no missing values, and the dataset with an interaction factor

```
data.no_missing <- data[-data.na_rows,] #create dataset with no missing values


data.binary<- data #make a copy of the original dataset
data.binary$X1.3a <- 1 #default all values to 1
data.binary[data.na_rows,]$X1.3a <- 0 #replace the value for rows with missing values in X1.3 with 1

#replace the original values in the column that were na with 0
data.binary[data.na_rows,]$X1.3 <- 0
```

Now lets take all of our datasets and store them into a list to loop over

```
data_sets <- list(mean_set = data.mean_set,
                  mode_set = data.mode_set,
                  regression_set = data.regression_set,
                  perturb_set = data.perturb_set,
                  binary = data.binary,
                  no_missing = data.no_missing)
```

Ok, now this part looks crazy, but it's just looping over every different data set, and for each data set, cross validate a model to determine the best value of K in a KNN model, then use that best model to get a percent accuracy from a set of test data. If this is unclear, I would revisit homework 2, this is nothing new!

```
library(kknn)
model_accuracy <- c()
data_set_names <- names(data_sets)


i = 1
for (new_data in data_sets){

  #create the train/test data
  sample_size <- floor(.8 * nrow(new_data))
  set.seed(1)
  train_ind <- sample(seq_len(nrow(new_data)), size = sample_size)

  k_train <-new_data[train_ind,]
  k_test <- new_data[-train_ind,]

  k_list <- as.list(c(1:10))
  k_accuracy <- c()

  pred <- rep(0, nrow(k_train)) # create a vector of numbers for as many rows in the data set
  knn_vector <- vector("numeric") #create an empty vector to store the accuracy of the model
  set.seed(1)
  k =1
  for (K in seq_along(k_list)){ #loop over k to find the accuracy

    k_model <-  cv.kknn(X2.1~.,
                        k_train,
                        kcv = 10,
                        k = K,
                        scale = TRUE)



    pred <- as.data.frame(round(k_model[[1]][,2]))
    actual <-as.data.frame(k_model[[1]][,1])
    k_accuracy <- sum(pred == actual) / nrow(pred) #calc how many predictions were correct

    knn_vector <- c(knn_vector, k_accuracy) #store the models accuracy for each value of k
  }

  best_k_model <-   kknn(k_train$X2.1~.,
                k_train,
                k_test,
                k = which.max(knn_vector),
                scale = TRUE)
  best_pred <- (fitted(best_k_model))
  model_accuracy$data_set[i] <- data_set_names[i]
  model_accuracy$accuracy[i] <- sum(best_pred == k_test$X2.1)/nrow(k_test)
  i <- i+1

}
as.data.frame(model_accuracy)
```

```
##         data_set  accuracy
## 1       mean_set 0.9571429
## 2       mode_set 0.9571429
## 3 regression_set 0.9571429
## 4    perturb_set 0.9500000
## 5         binary 0.9428571
## 6     no_missing 0.9781022
```

And now we can see clearly, regardless of what imputation method we used, it had hardly any impact on our final model. In fact, just ignoring the records with missing values entirely resulted in the highest accuracy!

This isn't too shoking, considering there was only a small percentage of records with missing values.

Another thing to note- on the binary dataset where we used an interaction factor to impute the missing data, this is often not a good approach. 1. As mentioned in the lecture video, imputing with a value of 0 (even though we have the interaction factor) results in the model pulling values away from what the value actually may have been inappropriately. 2. Adding in extra features like the interactino factor increases our dimensionality.

This isn't a huge issue in this data points given we have a large number of rows compared to the number of features (columns) but what if we only had 30 rows, and we wanted to add an interaction factor for ever column? We would need to consider a way to reduce dimensions back down to an appropriate level so that the model can run. For example, we could use PCA.