Missing Value Treatment

Missing values in data is a common phenomenon in real world problems. Knowing how to handle missing values effectively is a required step to reduce bias and to produce powerful models. Let's explore various options of how to deal with missing values and how to implement them.

Data prep and pattern

Lets use the BostonHousing dataset in mlbench package to discuss the various approaches to treating missing values. Though the original BostonHousing data doesn't have missing values, I am going to randomly introduce missing values. This way, we can validate the imputed missing values against the actuals, so that we know how effective are the approaches in reproducing the actual data. Lets begin by importing the data from mlbench pkg and randomly insert missing values (NA).

```
data ("BostonHousing", package="mlbench")  # initialize the data  # load the data
original <- BostonHousing  # backup original data
# Introduce missing values
set.seed(100)
BostonHousing[sample(1:nrow(BostonHousing), 40), "rad"] <- NA
BostonHousing[sample(1:nrow(BostonHousing), 40), "ptratio"] <- NA
head(BostonHousing)
#>      crim zn indus chas   nox    rm  age    dis rad tax ptratio      b lstat medv
#> 1 0.00632 18  2.31    0 0.538 6.575 65.2 4.0900   1 296    15.3 396.90  4.98 24.0
#> 2 0.02731  0  7.07    0 0.469 6.421 78.9 4.9671   2 242    17.8 396.90  9.14 21.6
#> 3 0.02729  0  7.07    0 0.469 7.185 61.1 4.9671   2 242    17.8 392.83  4.03 34.7
#> 4 0.03237  0  2.18    0 0.458 6.998 45.8 6.0622   3 222    18.7 394.63  2.94 33.4
#> 5 0.06905  0  2.18    0 0.458 7.147 54.2 6.0622   3 222    18.7 396.90  5.33 36.2
#> 6 0.02985  0  2.18    0 0.458 6.430 58.7 6.0622   3 222    18.7 394.12  5.21 28.7
```

The missing values have been injected. Though we know where the missings are, lets quickly check the 'missings' pattern using mice::md.pattern.

```
library(mice)
md.pattern(BostonHousing)  # pattern or missing values in data.
#>     crim zn indus chas nox rm age dis tax b lstat medv rad ptratio
#> 431    1  1     1    1   1  1   1   1   1  1     1    1   1       1  0
```

```
#> 35  1 1   1  1 1 1  1  1 1 1   1  1  0    1 1
#> 35  1 1   1  1 1 1  1  1 1 1   1  1  1    0 1
#>  5  1 1   1  1 1 1  1  1 1 1   1  1  0    0 2
#>     0 0   0  0 0 0  0  0 0 0   0  0 40   40 80
```

There are really four ways you can handle missing values:

1. Deleting the observations

If you have large number of observations in your dataset, where all the classes to be predicted are sufficiently represented in the training data, then try deleting (or not to include missing values while model building, for example by setting na.action=na.omit) those observations (rows) that contain missing values. Make sure after deleting the observations, you have:

1. Have sufficent data points, so the model doesn't lose power.
2. Not to introduce bias (meaning, disproportionate or non-representation of classes).

```
# Example
lm(medv ~ ptratio + rad, data=BostonHousing, na.action=na.omit)  # though na.omit is default in lm()
```

2. Deleting the variable

If a paricular variable is having more missing values that rest of the variables in the dataset, and, if by removing that one variable you can save many observations, then you are better off without that variable unless it is a really important predictor that makes a lot of business sense. It is a matter of deciding between the importance of the variable and losing out on a number of observations.

3. Imputation with mean / median / mode

Replacing the missing values with the mean / median / mode is a crude way of treating missing values. Depending on the context, like if the variation is low or if the variable has low leverage over the response, such a rough approximation is acceptable and could possibly give satisfactory results.

```
library(Hmisc)
impute(BostonHousing$ptratio, mean)  # replace with mean
impute(BostonHousing$ptratio, median)  # median
impute(BostonHousing$ptratio, 20)  # replace specific number
```

```
# or if you want to impute manually

BostonHousing$ptratio[is.na(BostonHousing$ptratio)] <- mean(BostonHousing$ptratio, na.rm = T)  # not run
```

Lets compute the accuracy when it is imputed with mean

```
library(DMwR)

actuals <- original$ptratio[is.na(BostonHousing$ptratio)]

predicteds <- rep(mean(BostonHousing$ptratio, na.rm=T), length(actuals))

regr.eval(actuals, predicteds)

#>       mae       mse      rmse      mape

#> 1.62324034 4.19306071 2.04769644 0.09545664
```

4. Prediction

4.1. kNN Imputation

DMwR::knnImputation uses k-Nearest Neighbours approach to impute missing values. What kNN imputation does in simpler terms is as follows: For every observation to be imputed, it identifies ‘k’ closest observations based on the euclidean distance and computes the weighted average (weighted based on distance) of these ‘k’ obs.

The advantage is that you could impute all the missing values in all variables with one call to the function. It takes the whole data frame as the argument and you don’t even have to specify which variabe you want to impute. But be cautious not to include the response variable while imputing, because, when imputing in test/production environment, if your data contains missing values, you won’t be able to use the unknown response variable at that time.

```
library(DMwR)

knnOutput <- knnImputation(BostonHousing[, !names(BostonHousing) %in% "medv"])  # perform knn imputation.

anyNA(knnOutput)

#> FALSE
```

Lets compute the accuracy.

```
actuals <- original$ptratio[is.na(BostonHousing$ptratio)]

predicteds <- knnOutput[is.na(BostonHousing$ptratio), "ptratio"]

regr.eval(actuals, predicteds)

#>       mae       mse      rmse      mape
```

```
#> 1.00188715 1.97910183 1.40680554 0.05859526
```

The mean absolute percentage error (mape) has improved by ~ 39% compared to the imputation by mean. Good.

4.2 rpart

The limitation with DMwR::knnImputation is that it sometimes may not be appropriate to use when the missing value comes from a factor variable. Both rpart and mice has flexibility to handle that scenario. The advantage with rpart is that you just need only one of the variables to be non NA in the predictor fields.

The idea here is we are going to use rpart to predict the missing values instead of kNN. To handle factor variable, we can set the method=class while calling rpart(). For numerics, we use, method=anova. Here again, we need to make sure not to train rpart on response variable (medv).

```
library(rpart)

class_mod <- rpart(rad ~ . - medv, data=BostonHousing[!is.na(BostonHousing$rad), ], method="class", na.action=na.omit)  # since rad is a factor

anova_mod <- rpart(ptratio ~ . - medv, data=BostonHousing[!is.na(BostonHousing$ptratio), ], method="anova", na.action=na.omit)  # since ptratio is numeric.

rad_pred <- predict(class_mod, BostonHousing[is.na(BostonHousing$rad), ])

ptratio_pred <- predict(anova_mod, BostonHousing[is.na(BostonHousing$ptratio), ])
```

Lets compute the accuracy for ptratio

```
actuals <- original$ptratio[is.na(BostonHousing$ptratio)]

predicteds <- ptratio_pred

regr.eval(actuals, predicteds)

#>       mae       mse      rmse      mape

#> 0.71061673 0.99693845 0.99846805 0.04099908
```

The mean absolute percentage error (mape) has improved additionally by another ~ 30% compared to the knnImputation. Very Good.

Accuracy for rad

```
actuals <- original$rad[is.na(BostonHousing$rad)]

predicteds <- as.numeric(colnames(rad_pred)[apply(rad_pred, 1, which.max)])

mean(actuals != predicteds)  # compute misclass error.

#> 0.25
```

This yields a mis-classification error of 25%. Not bad for a factor variable!

4.3 mice

mice short for Multivariate Imputation by Chained Equations is an R package that provides advanced features for missing value treatment. It uses a slightly uncommon way of implementing the imputation in 2-steps, using mice() to build the model and complete() to generate the completed data. The mice(df)function produces multiple complete copies of df, each with different imputations of the missing data.
The complete() function returns one or several of these data sets, with the default being the first. Lets see how to impute 'rad' and 'ptratio':

```
library(mice)
miceMod <- mice(BostonHousing[, !names(BostonHousing) %in% "medv"], method="rf")  # perform mice imputation, based on random forests.
miceOutput <- complete(miceMod)  # generate the completed data.
anyNA(miceOutput)
#> FALSE
```

Lets compute the accuracy of ptratio.

```
actuals <- original$ptratio[is.na(BostonHousing$ptratio)]
predicteds <- miceOutput[is.na(BostonHousing$ptratio), "ptratio"]
regr.eval(actuals, predicteds)
#>       mae       mse      rmse      mape
#> 0.36500000 0.78100000 0.88374204 0.02121326
```

The mean absolute percentage error (mape) has improved additionally by ~ 48% compared to the rpart. Excellent!.

Lets compute the accuracy of rad

```
actuals <- original$rad[is.na(BostonHousing$rad)]
predicteds <- miceOutput[is.na(BostonHousing$rad), "rad"]
mean(actuals != predicteds)  # compute misclass error.
#> 0.15
```

The mis-classification error reduced to 15%, which is 6 out of 40 observations. This is a good improvement compared to rpart's 25%.

If you'd like to dig in deeper, here is the manual.

Though we have an idea of how each method performs, there is not enough evidence to conclude which method is better or worse. But these are definitely worth testing out the next time you impute missing values.