First Thoughts on Missing Data

Introduction

We could spend at least a full semester talking about how to deal with missing data. Today I will just talk about a couple of first ideas.

Types of Missing Data

All descriptions/quotes below from https://en.wikipedia.org/wiki/Missing_data#Types_of_missing_data

- Missing completely at random (MCAR): "the events that lead to any particular data-item being missing are independent both of observable variables and of unobservable parameters of interest". In terms of statistical validity, it's ok to just drop the cases with missing data from our analysis.
- Missing at random(MAR): "missingness can be fully accounted for by variables where there is complete information... An example is that males are less likely to fill in a depression survey but this has nothing to do with their level of depression, after accounting for maleness." We have to be careful, but it's possible to analyze these data and obtain valid results.
- Not missing at random (NMAR): "the value of the variable that's missing is related to the reason it's missing... this would occur if men failed to fill in a depression survey *because* of their level of depression."

Running Data Set Example

Our data come from Elements of Statistical Learning. The following quote is from the documentation:

> A total of N=9409 questionnaires containg 502 questions were filled out by shopping mall customers in the San Francisco Bay area.
>
> The dataset income.data is an extract from this survey. It consists of 14 demographic attributes.

Our goal is to predict a household's annual income based on their responses to the other survey questions. As with basically all survey data, there is a lot of missing data.

```
library(readr)
library(dplyr)
library(ggplot2)
library(gridExtra)
library(purrr)
library(glmnet)
library(caret)
library(rpart)
library(tidyr)

# read in data
marketing <- read_table2("http://www.evanlray.com/data/ESL/marketing.data", col_names = FALSE)
```

```
## Warning in rbind(names(probs), probs_f): number of columns of result is not
## a multiple of vector length (arg 2)
```

```
## Warning: 1 parsing failure.
## row # A tibble: 1 x 5 col     row col   expected   actual   file                                  expec
```

```
names(marketing) <- c("household_income", "sex", "marital_status", "age", "education", "occupation", "time_in

# convert factors to factors
factor_vars <- c("sex", "marital_status", "age", "education", "occupation", "time_in_bay_area", "dual_income"
marketing <- marketing %>%
  mutate_at(.vars = factor_vars, factor)
```

```
# last row was all NAs, drop it
tail(marketing)
```

```
## # A tibble: 6 x 14
##   household_income sex   marital_status age   education occupation
##              <int> <fct> <fct>          <fct> <fct>     <fct>
## 1                1 2     5              1     1         2
## 2                2 1     5              2     4         1
## 3                1 2     5              1     2         1
## 4                4 1     1              6     4         3
## 5                6 1     5              3     4         1
## 6               NA <NA>  <NA>           <NA>  <NA>      <NA>
## # ... with 8 more variables: time_in_bay_area <fct>, dual_income <fct>,
## #   num_household_members <int>, num_household_children <int>,
## #   householder_status <fct>, home_type <fct>, ethnicity <fct>,
## #   home_language <fct>
```

```
marketing <- marketing %>% slice(-nrow(marketing))

# Initial train/test split ("estimation"/test) and cross-validation folds
set.seed(107847)
tt_inds <- caret::createDataPartition(marketing$household_income, p = 0.8)
train_set <- marketing %>% slice(tt_inds[[1]])
test_set <- marketing %>% slice(-tt_inds[[1]])

# how many missing values in each of train and test set?
count_missing <- function(x) {
  sum(is.na(x))
}

map_dbl(train_set, count_missing)
```

```
##       household_income                    sex         marital_status
##                      0                      0                    134
##                    age              education             occupation
##                      0                     66                    108
##       time_in_bay_area           dual_income  num_household_members
##                    746                      0                    298
## num_household_children     householder_status              home_type
##                      0                    198                    303
##              ethnicity          home_language
##                     51                    303
```

```
nrow(train_set)
```

```
## [1] 7196
```

```
map_dbl(test_set, count_missing)
```

```
##       household_income                    sex         marital_status
##                      0                      0                     26
##                    age              education             occupation
##                      0                     20                     28
##       time_in_bay_area           dual_income  num_household_members
##                    167                      0                     77
## num_household_children     householder_status              home_type
##                      0                     42                     54
##              ethnicity          home_language
##                     17                     56
```

```
nrow(test_set)
```

```
## [1] 1797
```

```
# Function to calculate error rate
calc_mse <- function(observed, predicted) {
  mean((observed - predicted)^2)
}
```

What if we just drop any training set observations with missing values?

```
train_set_none_missing <- train_set %>% drop_na()
map_dbl(train_set_none_missing, count_missing)
```

```
##       household_income                    sex         marital_status
##                      0                      0                      0
##                    age              education             occupation
##                      0                      0                      0
##       time_in_bay_area            dual_income  num_household_members
##                      0                      0                      0
## num_household_children     householder_status              home_type
##                      0                      0                      0
##              ethnicity          home_language
##                      0                      0
```

```
nrow(train_set_none_missing)
```

```
## [1] 5462
```

```
nrow(train_set_none_missing) / nrow(train_set)
```

```
## [1] 0.7590328
```

We just lost 24% of our data!! :(

```
rf_fit_none_missing <- train(
  form = household_income ~ .,
  data = train_set_none_missing,
  method = "rf",
  trControl = trainControl(method = "oob",
    returnResamp = "all",
    savePredictions = TRUE),
  tuneLength = 1
)
```

```
rf_preds_none_missing <- predict(rf_fit_none_missing, newdata = test_set)
length(rf_preds_none_missing)
```

```
## [1] 1414
```

```
nrow(test_set)
```

```
## [1] 1797
```

...and we can't even make predictions for everything in the test set. Test set MSE of ∞??

OK, but what about for the observations where we were able to make a prediction?

```
test_set_na_inds <- which(apply(test_set, 1, function(x) { any(is.na(x)) }))
test_set_none_missing <- test_set %>% slice(-test_set_na_inds)
nrow(test_set_none_missing)
```

```
## [1] 1414
```

```
calc_mse(rf_preds_none_missing, test_set_none_missing$household_income)
```

```
## [1] 3.841832
```

Impute missing values with median/most commonly occurring?

```
impute_missing_median <- function(x) {
  x[is.na(x)] <- median(x, na.rm = TRUE)
  return(x)
}

impute_missing_most_common <- function(x) {
  count_table <- table(x)
  x[is.na(x)] <- names(count_table)[which.max(count_table)]
  return(x)
}

marketing_median_impute <- marketing %>%
  mutate_at(factor_vars, impute_missing_most_common) %>%
  mutate_at(c("num_household_members", "num_household_children"), impute_missing_median)

train_set_median_impute <- marketing_median_impute %>% slice(tt_inds[[1]])
test_set_median_impute <- marketing_median_impute %>% slice(-tt_inds[[1]])

rf_fit_median_impute <- train(
  form = as.numeric(household_income) ~ .,
  data = train_set_median_impute,
  method = "rf",
  trControl = trainControl(method = "oob",
    returnResamp = "all",
    savePredictions = TRUE),
  tuneLength = 1
)
```

```
rf_preds_median_impute_all_cases <- predict(rf_fit_median_impute, newdata = test_set_median_impute)
rf_error_median_impute <- calc_mse(test_set$household_income, rf_preds_median_impute_all_cases)
rf_error_median_impute
```

```
## [1] 3.977617
```

```
calc_mse(test_set_none_missing$household_income,
         predict(rf_fit_median_impute, newdata = test_set_median_impute %>% slice(-test_set_na_inds)))
```

```
## [1] 3.829537
```

Impute by way of trees

```
marketing_x <- marketing %>% select(-household_income)

fit_one_tree_and_predict_target <- function(b, target_var) {
  fit_formula <- as.formula(paste0(target_var, " ~ ."))
  tree_fit <- train(fit_formula,
    data = marketing_x,
    control = rpart.control(xval = 0L, maxsurrogate = 3),
    method = "rpart",
    trControl = trainControl(method = "none"),
    na.action = na.rpart)

  marketing_target_imputed <- marketing_x[[target_var]]
  missing_inds <- which(is.na(marketing_target_imputed))
  preds <- predict(tree_fit, newdata = marketing_x %>% slice(missing_inds), na.action = na.rpart)
  marketing_target_imputed[missing_inds] <- preds

  result <- data.frame(
      x = marketing_target_imputed
  )
  names(result) <- target_var

  return(result)
}
```

```
marketing_x_imputed <- map_dfc(colnames(marketing_x), fit_one_tree_and_predict_target, b = 1L)
```

```
train_x_imputed <- marketing_x_imputed %>% slice(tt_inds[[1]])
test_x_imputed <- marketing_x_imputed %>% slice(-tt_inds[[1]])
```

```
y_train <- as.numeric(train_set$household_income)
rf_fit_tree_impute <- train(
  x = train_x_imputed,
  y = y_train,
  method = "rf",
  trControl = trainControl(method = "oob",
    returnResamp = "all",
    savePredictions = TRUE),
  tuneLength = 1
)
```

```
rf_error_tree_impute <- calc_mse(as.numeric(test_set$household_income), predict(rf_fit_tree_impute, newdata =
rf_error_tree_impute
```

```
## [1] 3.932311
```

```
calc_mse(test_set_none_missing$household_income,
  predict(rf_fit_tree_impute, newdata = test_set_median_impute %>% slice(-test_set_na_inds)))
```

```
## [1] 3.823088
```