Data Preprocessing

Yifei Sun

Contents

```
library(caret)
library(visdat)
library(gridExtra)
library(mvtnorm)
library(ISLR)
```

Transforming predictors

Although not always required, transforming the variables may lead to improvement in prediction, especially for parametric models.

For example, one may consider the Box-Cox transformation, which finds an appropriate transformation from a family of power transformation that will transform the variable as close as possible to a normal distribution. One may also consider the Yeo-Johnson transformation if the variables are not strictly positive.

```
gen_data <- function(N)
{
  X <- rmvnorm(N, mean = c(1,-1),
               sigma = matrix(c(1,.5,.5,1), ncol = 2))
  X1 <- exp(X[,1])
  X2 <- X[,2]
  X3 <- rep(1, N)
  eps <- rnorm(N, sd = .5)
  Y <- log(X1) + X2 + eps

  data.frame(Y = Y, X1 = X1, X2 = X2, X3 = X3)
}

set.seed(2021)
trainData <- gen_data(100)
testData <- gen_data(50)

x <- trainData[, -1]
y <- trainData[, 1]
x2 <- testData[, -1]
y2 <- testData[, 1]
```

preProcess in train()

```
fit.lm <- train(x, y,
                preProcess = c("BoxCox", "zv"),
                method = "lm",
                trControl = trainControl(method = "none"))

fit.lm
```

```
## Linear Regression
##
## 100 samples
##   3 predictor
```

```
##
## Pre-processing: Box-Cox transformation (1), remove (1)
## Resampling: None
```

```
pred.lm <- predict(fit.lm, newdata = x2)

fit.lm$preProcess$bc
```

```
## $X1
## Box-Cox Transformation
##
## 100 data points used to estimate Lambda
##
## Input data summary:
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##  0.1593  1.0890  2.5188  4.0904  4.3320 60.9414
##
## Largest/Smallest: 383
## Sample Skewness: 5.91
##
## Estimated Lambda: -0.1
## With fudge factor, Lambda = 0 will be used for transformations
```

preProcess()

The transformation is computed using the training data. Then it is applied to both training and test data.

```
pp <- preProcess(x, method = c("BoxCox", "zv"))

# transformed predictor matrix (training)
x_pp <- predict(pp, x)

head(x_pp)
```

```
##          X1          X2
## 1  1.024699 -0.49806285
## 2  1.429849 -0.56238480
## 3  1.369856 -2.62462616
## 4  1.489792 -0.04788637
## 5  1.461050  0.67458054
## 6 -0.115942 -1.54362411
```

```
# transformed predictor matrix (test)
x2_pp <- predict(pp, x2)

head(x2_pp)
```

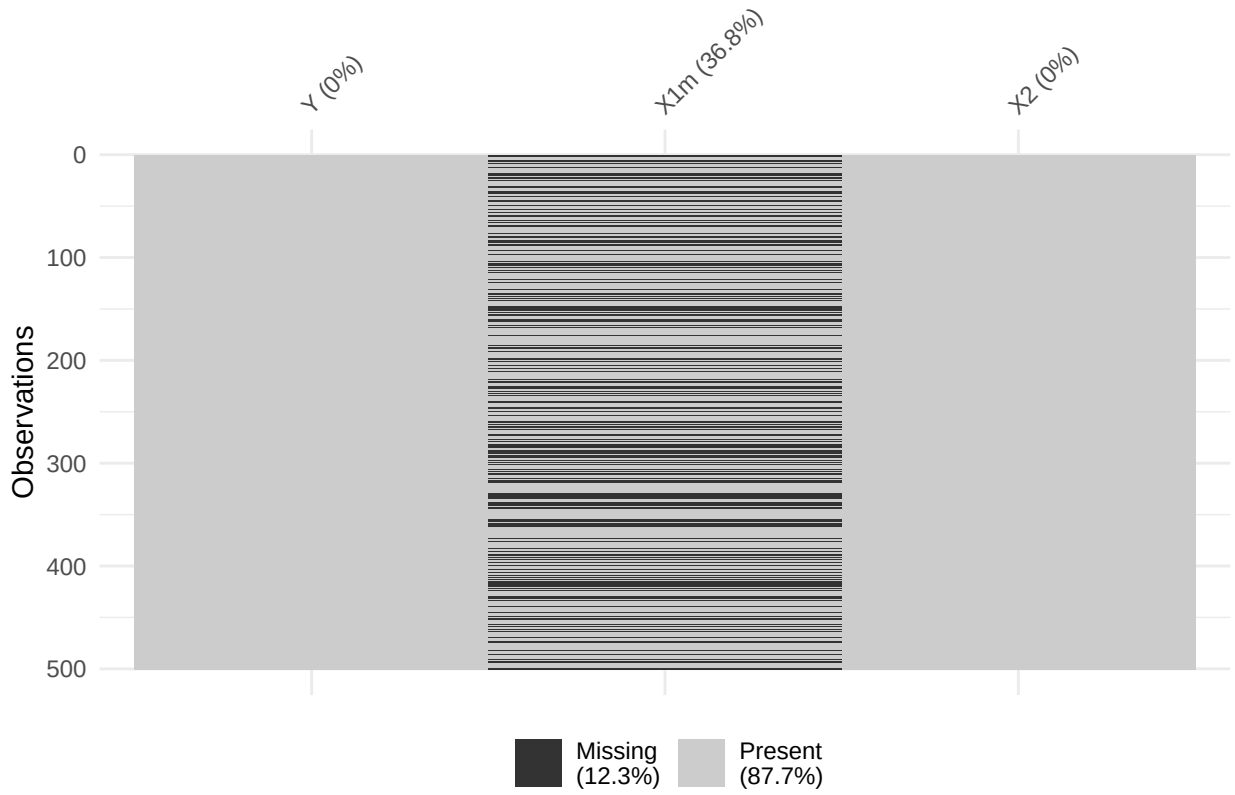
```
##           X1         X2
## 1 2.42225642  0.2723873
## 2 2.56224770 -0.3197342
## 3 1.35315271 -1.3903441
```

```
## 4 0.52508528 -0.5174462
## 5 0.82175069 -2.2899954
## 6 0.07559228 -0.9121611
```

Missing data

There are different mechanisms for missing data: missing completely at random (MCAR), missing at random (MAR), missing not at random (MNAR). MAR means that the missingness depends only on the observed data; MNAR means that the missingness further depends on the missing data. The missing data mechanism determines how you handle the missing data. For example, under MAR, you may consider imputation methods; under MNAR, you may consider treating missingness as an attribute.

```
gen_data <- function(N)
{
  X <- rmvnorm(N, mean = c(1,-1),
               sigma = matrix(c(1,0.5,0.5,1), ncol = 2))
  X1 <- X[,1]
  X2 <- X[,2]
  eps <- rnorm(N, sd = .5)
  Y <- X1 + X2 + eps

  # which X1 observations are missing
  ind_missing <- rbinom(N, size = 1, prob = exp(X2/2)/(1+exp(X2/2)))

  X1m <- X1
  X1m[ind_missing == 1] <- NA

  data.frame(Y = Y, X1m = X1m, X2 = X2, X1 = X1)
}

set.seed(2021)

dat <- gen_data(500)
dat2 <- gen_data(100)
trainData <- dat[,1:3]
testData <- dat2[,1:3]

vis_miss(trainData)
```



preProcess()

```
trainX <- trainData[,c(2:3)]
knnImp <- preProcess(trainX, method = "knnImpute", k = 3)
bagImp <- preProcess(trainX, method = "bagImpute")
medImp <- preProcess(trainX, method = "medianImpute")

trainX_knn <- predict(knnImp, trainX)
trainX_bag <- predict(bagImp, trainX)
trainX_med <- predict(medImp, trainX)

testData_knn <- predict(knnImp, testData)
testData_bag <- predict(bagImp, testData)
testData_med <- predict(medImp, testData)

head(trainX)
```

```
##        X1m          X2
## 1       NA -0.49806285
## 2       NA -0.56238480
## 3 1.369856 -2.62462616
## 4 1.489792 -0.04788637
## 5 1.461050  0.67458054
## 6       NA -1.54362411
```

```
head(trainX_med)
```

```
##         X1m          X2
## 1 0.9535893 -0.49806285
## 2 0.9535893 -0.56238480
## 3 1.3698556 -2.62462616
## 4 1.4897917 -0.04788637
## 5 1.4610501  0.67458054
## 6 0.9535893 -1.54362411
```

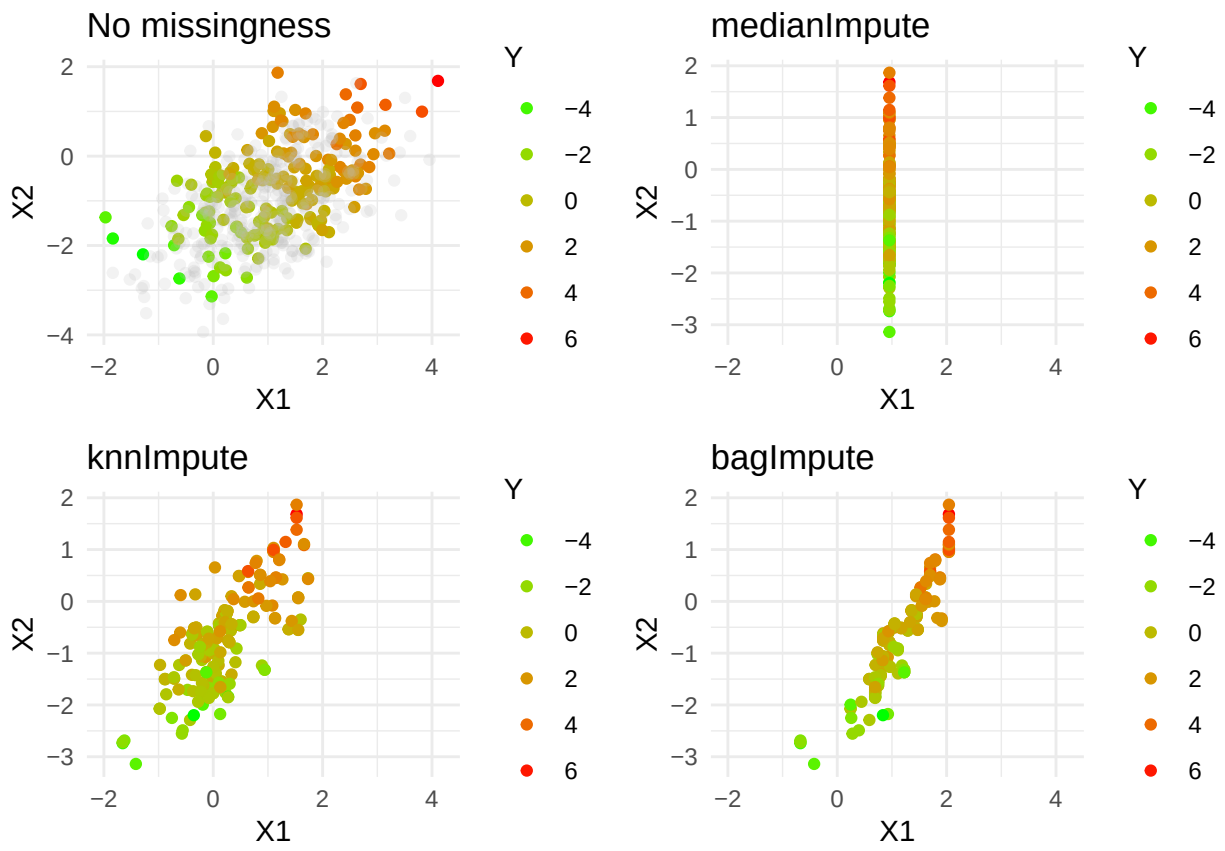
```
head(trainX_knn)
```

```
##         X1m         X2
## 1 0.1152723  0.4707506
## 2 0.3195374  0.4075333
## 3 0.4518912 -1.6192909
## 4 0.5702574  0.9131957
## 5 0.5418920  1.6232549
## 6 0.1535531 -0.5568541
```

```
head(trainX_bag)
```

```
##         X1m          X2
## 1 1.0483302 -0.49806285
## 2 1.2900352 -0.56238480
## 3 1.3698556 -2.62462616
## 4 1.4897917 -0.04788637
## 5 1.4610501  0.67458054
## 6 0.7468201 -1.54362411
```

```
## Warning: Removed 1 rows containing missing values (geom_point).
```



preProcess in train()

```
fit.lm <- train(x = trainData[,c(2,3)],
                y = trainData$Y,
                preProcess = c("knnImpute"), # bagImpute/medianImpute
                method = "lm",
                trControl = trainControl(method = "none",
                                         preProcOptions = list(k = 5)))

pred.lm <- predict(fit.lm, newdata = testData)

mean((testData$Y - pred.lm)^2)
```

```
## [1] 0.6347831
```