Oversampling Unbalanced Data With SMOTE

Aaron Stearns

3/20/2019

This analysis aims to accurately predict whether a business will declare bankruptcy or not. The dataset is available here: https://www.kaggle.com/shebrahimi/financial-distress/data As we will see in this classification problem, the target variable is extremely unbalanced, and will therefore need to be oversampled. I will use the Synthetic Minority Over-sampling Technique (SMOTE) developed by Chawla et al. and published in the paper http://www.csee.usf.edu/~lohall/papers/smote.pdf

```
# I'll start by importing the necessary libraries and the data,
# then checking on the dimensions of the dataset:
library(caret)
library(mlbench)
library(DMwR)
library(e1071)
library(randomForest)
library(dplyr)
library(mltools)
library(data.table)

df <- read.csv("/Users/aaron/Documents/Financial-Distress.csv",
               stringsAsFactors=FALSE)

# check the dimensions of the dataset
dim(df)
```

```
## [1] 3672   86
```

```
# take a look at column data types
str(df)
```

```
## 'data.frame':    3672 obs. of  86 variables:
##  $ Company           : int  1 1 1 1 2 2 2 2 2 2 ...
##  $ Time              : int  1 2 3 4 1 2 3 4 5 6 ...
##  $ Financial.Distress: num  0.0106 -0.456 -0.3254 -0.5666 1.3573 ...
##  $ x1                : num  1.28 1.27 1.05 1.11 1.06 ...
##  $ x2                : num  0.02293 0.00645 -0.05938 -0.01523 0.10702 ...
##  $ x3                : num  0.875 0.821 0.922 0.859 0.815 ...
##  $ x4                : num  1.216 1.005 0.729 0.81 0.836 ...
##  $ x5                : num  0.0609 -0.0141 0.0205 0.076 0.2 ...
##  $ x6                : num  0.1883 0.181 0.0449 0.091 0.0478 ...
##  $ x7                : num  0.525 0.623 0.433 0.675 0.742 ...
##  $ x8                : num  0.01885 0.00642 -0.08142 -0.01881 0.12803 ...
##  $ x9                : num  0.183 0.036 -0.765 -0.108 0.577 ...
##  $ x10               : num  0.00645 0.0018 -0.05432 -0.06532 0.09408 ...
##  $ x11               : num  0.858 0.852 0.893 0.896 0.815 ...
##  $ x12               : num  2.006 -0.486 0.412 0.995 3.015 ...
##  $ x13               : num  0.1255 0.1793 0.0776 0.1411 0.1854 ...
##  $ x14               : num  6.97 4.58 11.89 6.09 4.39 ...
##  $ x15               : num  4.65 3.75 2.49 1.64 1.62 ...
##  $ x16               : num  0.0501 -0.014 0.0281 0.0939 0.2392 ...
##  $ x17               : num  2.2 2.46 1.4 2.06 3.03 ...
```

```
##  $ x18              : num  0.01826 0.02756 0.0126 0.0116 0.00681 ...
##  $ x19              : num  0.025 0.0288 0.0681 0.0944 0.0793 ...
##  $ x20              : num  0.02726 0.0411 0.01485 0.01442 0.00888 ...
##  $ x21              : num  1.417 1.18 0.817 0.904 1.025 ...
##  $ x22              : num  9.56 7.3 7.12 7.98 4.75 ...
##  $ x23              : num  0.1487 0.056 0.0652 0.1252 0.266 ...
##  $ x24              : num  0.67 0.67 0.848 0.805 0.768 ...
##  $ x25              : num  214.8 38.2 -498.4 -75.9 1423.1 ...
##  $ x26              : num  12.6 12.9 13.2 13.3 11.6 ...
##  $ x27              : num  6.46 5.55 16.25 8.89 17.49 ...
##  $ x28              : num  0.0438 0.2655 0.4166 0.0838 0.6208 ...
##  $ x29              : num  0.2046 0.1502 0.0741 0.0541 0.0469 ...
##  $ x30              : num  0.352 0.418 0.367 0.544 0.57 ...
##  $ x31              : num  8.32 9.53 9.35 7.09 9.49 ...
##  $ x32              : num  0.289 0.416 0.504 0.671 0.681 ...
##  $ x33              : num  0.766 0.817 0.92 0.937 0.942 ...
##  $ x34              : num  2.58 2.6 1.49 2.35 4.13 ...
##  $ x35              : num  77.4 95.9 144.7 219.8 222.7 ...
##  $ x36              : num  0.02672 0.00758 -0.06648 -0.017 0.13123 ...
##  $ x37              : num  1.631 0.838 0.956 0.383 0.253 ...
##  $ x38              : num  0.01502 0.02743 0.01727 0.01433 0.00815 ...
##  $ x39              : num  0.00548 0.04543 0.02806 0.20337 0.35301 ...
##  $ x40              : num  0.127 0.138 0.102 0.101 0.176 ...
##  $ x41              : num  9.7 5.6 9.4 5.74 4.51 ...
##  $ x42              : num  -0.736 -0.644 -14.032 0.722 -0.113 ...
##  $ x43              : num  0.986 1.302 0.757 1.391 1.053 ...
##  $ x44              : num  0.1802 0.0469 -0.5798 -0.1501 0.6077 ...
##  $ x45              : num  1.501 1.01 0.578 0.645 0.258 ...
##  $ x46              : num  0.02622 0.00786 -0.06437 -0.01773 0.13138 ...
##  $ x47              : num  7.05 4.6 11.99 6.11 4.42 ...
##  $ x48              : num  1175 1062 651 703 2465 ...
##  $ x49              : num  5.34 3.74 10.93 5.7 4.14 ...
##  $ x50              : num  0.851 0.944 0.935 0.875 0.734 ...
##  $ x51              : num  12.8 12.9 12.9 13.1 11.4 ...
##  $ x52              : num  0.061737 -0.000565 0.041625 0.1084 0.25031 ...
##  $ x53              : num  0.1809 0.0563 0.0476 0.1013 0.2224 ...
##  $ x54              : num  210 250 281 414 315 ...
##  $ x55              : num  -0.5826 -0.4748 -1 0.565 -0.0601 ...
##  $ x56              : num  0.471 0.386 0.488 0.344 0.202 ...
##  $ x57              : num  0.1099 0.3693 0.0533 0.0734 1.2291 ...
##  $ x58              : num  0.00 0.00 3.79e-03 3.66e-05 -2.49e-03 ...
##  $ x59              : num  0.00 0.00 5.19e-03 4.53e-05 -2.98e-03 ...
##  $ x60              : num  0.22 0 0 0 0.227 ...
##  $ x61              : num  7.12 7.42 3.64 5.14 7.12 ...
##  $ x62              : num  15.38 7.11 7.02 9.91 15.38 ...
##  $ x63              : num  3.27 14.32 1.15 2.04 3.27 ...
##  $ x64              : num  17.87 18.77 9.9 -1.49 17.87 ...
##  $ x65              : num  34.69 124.76 6.45 -21.91 34.69 ...
##  $ x66              : num  30.1 26.1 30.2 34.3 30.1 ...
##  $ x67              : num  12.8 11.8 10.3 11.5 12.8 11.8 10.3 11.5 11.3 10.5 ...
##  $ x68              : num  7991 8323 8747 9042 7991 ...
##  $ x69              : num  364.95 0.19 11.95 -18.75 364.95 ...
##  $ x70              : num  15.8 15.6 15.2 10.4 15.8 15.6 15.2 10.4 11.9 18.4 ...
##  $ x71              : num  61.5 24.6 20.7 47.4 61.5 ...
```

```
##  $ x72               : num  4 0 0 4 4 0 0 4 4 2 ...
##  $ x73               : num  36 36 35 33 36 36 35 33 31 29 ...
##  $ x74               : num  85.4 107.1 120.9 54.8 85.4 ...
##  $ x75               : num  27.1 31.3 36.1 39.8 27.1 ...
##  $ x76               : num  26.1 30.2 35.3 38.4 26.1 ...
##  $ x77               : num  16 17 17 17.2 16 ...
##  $ x78               : num  16 16 15 16 16 16 15 16 14 12 ...
##  $ x79               : num  0.2 0.4 -0.2 5.6 0.2 0.4 -0.2 5.6 2.1 -6.4 ...
##  $ x80               : int  22 22 22 22 29 29 29 29 29 29 ...
##  $ x81               : num  0.0604 0.0106 -0.456 -0.3254 1.251 ...
##  $ x82               : int  30 31 32 33 7 8 9 10 11 12 ...
##  $ x83               : int  49 50 51 52 27 28 29 30 31 32 ...
```

```r
# check the columnwise sums of NA values:
table(is.na(df))
```

```
##
##  FALSE
## 315792
```

In the data description page on Kaggle, the person who uploaded the dataset says that column 'x80' is categorical, so I'll take a look at it:

```r
table(df$x80)
```

```
##
##   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17  18
##  14  19  14  74   7  14  12  24 335  61  32  79   4 236 382   3  75 245
##  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36
## 169 104 119 157 110  27 371 274  47  47 360 146  18  65   4   4   3   3
##  37
##  14
```

It likes like there are 37 unique values for this variable, so I'll one-hot encode this feature and replace the column in the original data frame with this new matrix

```r
one_hot_vec <- function(x) {
    nc <- max(x)
    nr <- length(x)
    m <- integer(nr * nc)
    i <- (seq_len(nr) - 1) * nc + x
    m[i] <- 1L
    matrix(m, nrow = nr, ncol = nc, byrow = TRUE)
}

oneHot <- one_hot_vec(df$x80)

# inspect the one-hot matrix:
head(oneHot)
```

```
##      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10] [,11] [,12] [,13]
## [1,]    0    0    0    0    0    0    0    0    0     0     0     0     0
## [2,]    0    0    0    0    0    0    0    0    0     0     0     0     0
## [3,]    0    0    0    0    0    0    0    0    0     0     0     0     0
## [4,]    0    0    0    0    0    0    0    0    0     0     0     0     0
## [5,]    0    0    0    0    0    0    0    0    0     0     0     0     0
## [6,]    0    0    0    0    0    0    0    0    0     0     0     0     0
##      [,14] [,15] [,16] [,17] [,18] [,19] [,20] [,21] [,22] [,23] [,24]
```

```
## [1,]     0     0     0     0     0     0     0     0     1     0     0
## [2,]     0     0     0     0     0     0     0     0     1     0     0
## [3,]     0     0     0     0     0     0     0     0     1     0     0
## [4,]     0     0     0     0     0     0     0     0     1     0     0
## [5,]     0     0     0     0     0     0     0     0     0     0     0
## [6,]     0     0     0     0     0     0     0     0     0     0     0
##      [,25] [,26] [,27] [,28] [,29] [,30] [,31] [,32] [,33] [,34] [,35]
## [1,]     0     0     0     0     0     0     0     0     0     0     0
## [2,]     0     0     0     0     0     0     0     0     0     0     0
## [3,]     0     0     0     0     0     0     0     0     0     0     0
## [4,]     0     0     0     0     0     0     0     0     0     0     0
## [5,]     0     0     0     0     1     0     0     0     0     0     0
## [6,]     0     0     0     0     1     0     0     0     0     0     0
##      [,36] [,37]
## [1,]     0     0
## [2,]     0     0
## [3,]     0     0
## [4,]     0     0
## [5,]     0     0
## [6,]     0     0
```

```
# change column names in oneHot matrix
colnames(oneHot) <- paste0("x80_", 1:ncol(oneHot))

oneHot <- as.data.frame(oneHot)

df <- df %>%
      select(-x80)

# add oneHot matrix to original data frame
df <- cbind(df, oneHot)
```

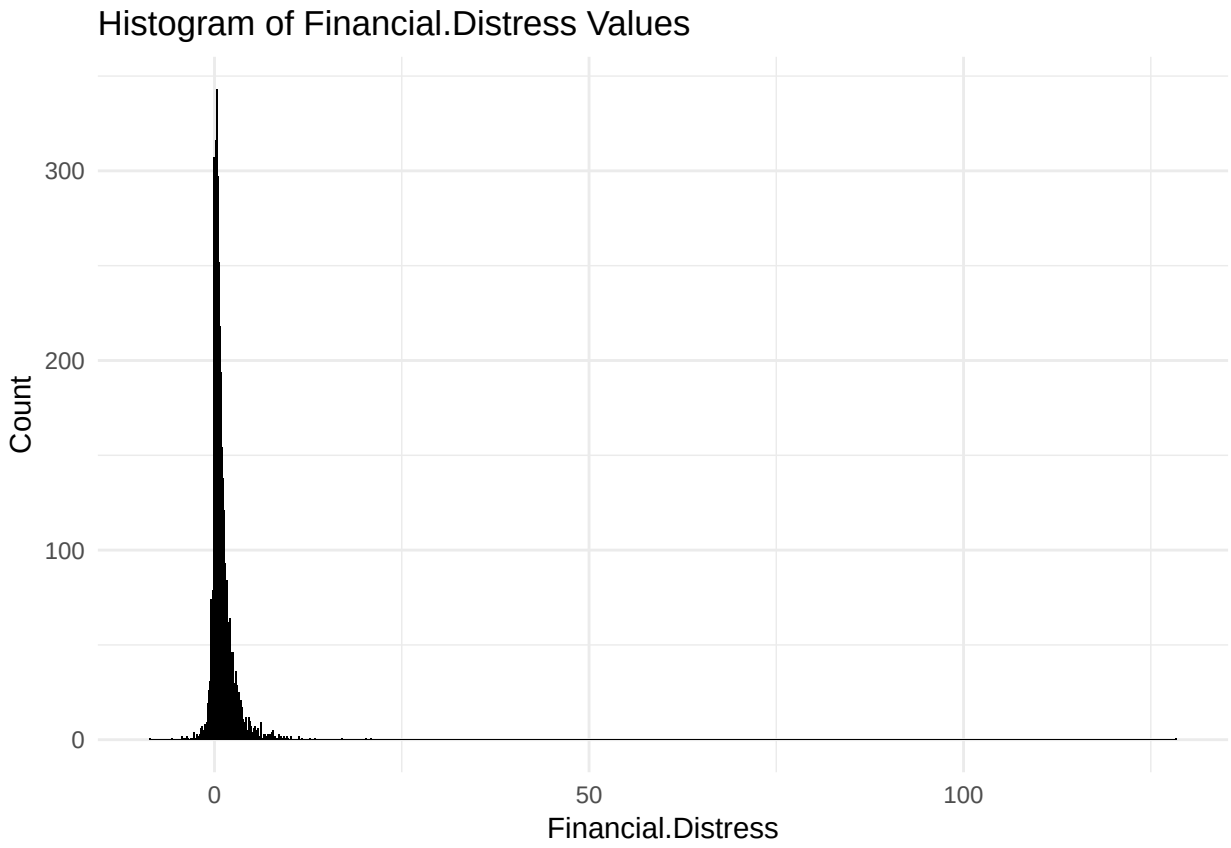
The dependent variable, "Financial.Distress" seems to be continuous rather than categorical, with some extreme outliers, as seen in the five number summary of the variable below:

```
fivenum(df$Financial.Distress)
```

```
## [1]  -8.631700   0.172130   0.583805   1.352300 128.400000
```

I'll visualize the Financial.Distress values to see how many outliers there are:

```
ggplot(df, aes(Financial.Distress)) +
  geom_histogram(bins = 1000, fill = "black") +
  labs(x="Financial.Distress", y="Count") +
  ggtitle("Histogram of Financial.Distress Values") +
  theme_minimal()
```



So it looks like there are just a few outliers, as indicated before in the five number summary.

I'll check what percentage of the total rows in the data have a "Financial.Distress" value greater than 5

```
df %>%
  filter(Financial.Distress > 5) %>%
  count() / nrow(df) * 100
```

```
##          n
## 1 2.750545
```

Only 2.75 percent. I'll remove those rows.

```
df <- df %>%
      filter(Financial.Distress < 5)
```

Now I'll convert the dependent variable to a categorical variable following the instructions in the kaggle kernel (pasted in the comment below)

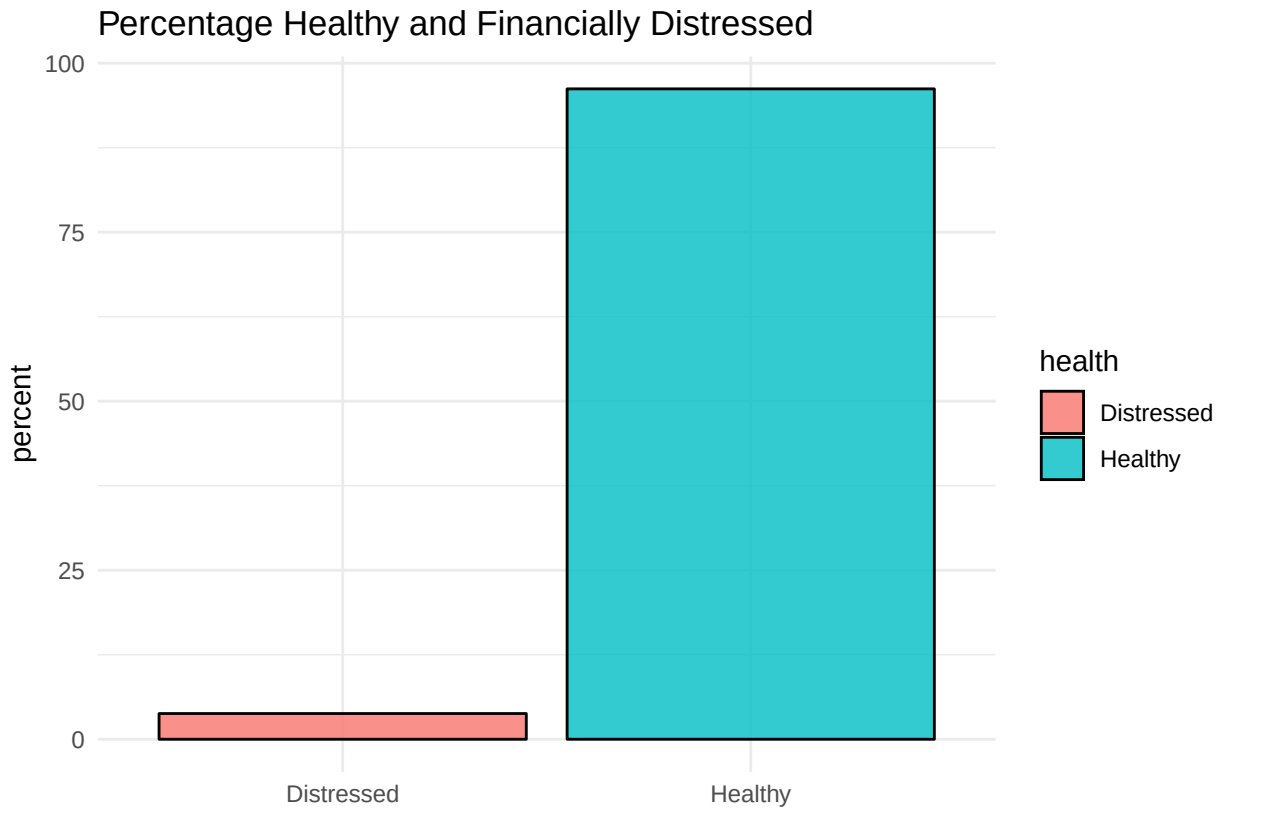
```
# The target variable is denoted by "Financial Distress" if it
# is greater than -0.50 the company should be considered as healthy (0).
# Otherwise, it would be regarded as financially distressed (1).
df$Financial.Distress  <- ifelse(df$Financial.Distress > -0.5, 0, 1)

df$Financial.Distress <- as.factor(df$Financial.Distress)
```

```
percs <- df %>%
  group_by(Financial.Distress) %>%
  summarise(n = n()) %>%
  mutate(perc = n/nrow(df) * 100) %>%
  mutate(perc = round(perc, digits = 1))
```

```r
percs$health <- ifelse(percs$Financial.Distress == 0, "Healthy", "Distressed")

ggplot(data = percs, aes(x = health, y = perc, fill = health)) +
  geom_bar(stat = "identity", alpha = 0.8, color = "black") +
  labs(x = "", y = "percent") +
  ggtitle("Percentage Healthy and Financially Distressed") +
  theme_minimal()
```



We can see that the classes are very unbalanced, with businesses in financial distress only accounting for 3.8 percent of observations in the data. Let's first see how a model performs without oversampling the minority class in the training data:

```r
n <- nrow(df)

trainIndex <- sample(1:n, size = round(0.7*n), replace=FALSE)

train <- df[trainIndex, 3:82]

test <- df[-trainIndex, 3:82]

set.seed(7)

trainControl <- trainControl(method="cv", number=5)

# Random forest classifier with 5-fold cross validation
fit.rf <- train(Financial.Distress ~ .,
                data = train,
                method="rf",
```

```
                trControl=trainControl)

p <- predict(fit.rf, test)

confusion <- as.data.frame(cbind(test$Financial.Distress, p))

compare <- ifelse(confusion$V1 == 1 & confusion$p == 2, "falsePositive",
             ifelse(confusion$V1 == 2 & confusion$p == 1, "falseNegative",
              ifelse(confusion$V1 == 1 & confusion$p == 1, "trueNegative",
                ifelse(confusion$V1 == 2 & confusion$p == 2, "truePositive", "Neither"))))

table(compare)
```

```
## compare
## falseNegative falsePositive  trueNegative  truePositive
##            37             6          1023             5
```

Here we can see that although the overall accuracy is high, there is a very high false negative rate.

Now, I will oversample the minority class by 200 percent and undersample the majority class by 100 percent in the training set using the SMOTE function in the DMwR package:

```
n <- nrow(df)

trainIndex <- sample(1:n, size = round(0.7*n), replace=FALSE)

train <- df[trainIndex, 3:ncol(df)]

test <- df[-trainIndex, 3:ncol(df)]

# Oversampling:
trainSplit <- SMOTE(Financial.Distress ~ .,
                    train,
                    perc.over = 200,
                    perc.under = 100)

set.seed(7)

trainControl <- trainControl(method="cv", number=5)

# Same random forest model with 5-fold cross validation as before
fit.rf <- train(Financial.Distress ~ .,
                data = trainSplit,
                method="rf",
                trControl=trainControl)

p <- predict(fit.rf, test)

confusion <- as.data.frame(cbind(test$Financial.Distress, p))

compare <- ifelse(confusion$V1 == 1 & confusion$p == 2, "falsePositive",
             ifelse(confusion$V1 == 2 & confusion$p == 1, "falseNegative",
              ifelse(confusion$V1 == 1 & confusion$p == 1, "trueNegative",
                ifelse(confusion$V1 == 2 & confusion$p == 2, "truePositive", "Neither"))))
```

```r
table(compare)
```

```
## compare
## falseNegative falsePositive  trueNegative  truePositive
##             7           140           889            35
```

Here we can see that although the overall accuracy has dropped and the false positive rate his risen, the true positive rate has risen significantly, with the false negative rate dropping drastically.