Automating Data Exploration with R

Categorical Data

Factors

We are now able to recognize the most common data types but the problem remains that in order to model data, everything needs to be in a numerical format. In the majority of cases you cannot turn text into factors and model off the level of that factor. This only works in rare cases with ordered categorical data and even then you have to be very careful. Imagine a survey question on satisfaction going from `very unhappy` to `very happy`. It would seem logical that we could use the index value of those categories instead of the text. And, in this case it works - if `happy` is 4 and `very happy` is 5, then 4.5 is somewhere in between. Nope, not if you use straight out-of-the-box factor levels:

```
survey <- data.frame(satisfaction=c('very unhappy','unhappy','neutral','happy','ve
ry happy'))
print(survey)
```

```
##   satisfaction
## 1 very unhappy
## 2      unhappy
## 3      neutral
## 4        happy
## 5   very happy
```

```
survey$satisfaction <- as.factor(survey$satisfaction)
survey$satisfaction_Level <- as.numeric(survey$satisfaction)
print(survey$satisfaction_Level)
```

```
## [1] 5 3 2 1 4
```

Unfortunately, you can't blindly rely on the `factor` level as it automatically assigns levels based on alphabetic order. So, to make sure that 1.5 is between `very unhappy` and `unhappy` you would need to customized the level correctly:

```
survey$satisfaction <- as.factor(survey$satisfaction)
levels(survey$satisfaction) <- list('very unhappy'=1,'unhappy'=2,'neutral'=3,'happ
y'=4,'very happy'=5)
survey$satisfaction_Level <- as.numeric(survey$satisfaction)
print(survey$satisfaction_Level)
```

```
## [1] 1 2 3 4 5
```

We can pull this off because it is an ordered categorical variable and because it is small. What about bigger categories and non-ordered ones? Like what to do with Zip codes that have over 43,000 levels where an intermediary value between two zip codes doesn't mean anything?

Binarizing Data - Making Dummy Features

You can create dummy variables manually, using base functions, such as matrix, or a packaged function like `dummyVar` from the `caret` package. Let's build our own function to create dummy variables so we can fully appreciate what it means.

Let's take a simple example using the Titanic data set:

```
Titanic_dataset <- read.table('http://math.ucdenver.edu/RTutorial/titanic.txt', se
p='\t', header=TRUE)
head(Titanic_dataset)
```

```
##                                             Name PClass   Age    Sex
## 1                   Allen, Miss Elisabeth Walton    1st 29.00 female
## 2                    Allison, Miss Helen Loraine    1st  2.00 female
## 3            Allison, Mr Hudson Joshua Creighton    1st 30.00   male
## 4 Allison, Mrs Hudson JC (Bessie Waldo Daniels)    1st 25.00 female
## 5                  Allison, Master Hudson Trevor    1st  0.92   male
## 6                             Anderson, Mr Harry    1st 47.00   male
##   Survived
## 1        1
## 2        0
## 3        0
## 4        0
## 5        1
## 6        1
```

`PClass` and `Sex` are candidates to be binarized:

```
dim(Titanic_dataset)
```

```
## [1] 1313    5
```

```
unique(Titanic_dataset$Sex)
```

```
## [1] female male
## Levels: female male
```

```
unique(Titanic_dataset$PClass)
```

```
## [1] 1st 2nd 3rd
## Levels: 1st 2nd 3rd
```

We could do it by hand such as:

```
Titanic_dataset_temp <- Titanic_dataset
Titanic_dataset_temp$Sex_Female <- ifelse(Titanic_dataset_temp$Sex=='female', 1,
0)
Titanic_dataset_temp$Sex_Male <- ifelse(Titanic_dataset_temp$Sex=='male', 1, 0)
head(Titanic_dataset_temp)
```

```
##                                             Name PClass   Age    Sex
## 1                  Allen, Miss Elisabeth Walton    1st 29.00 female
## 2                   Allison, Miss Helen Loraine    1st  2.00 female
## 3           Allison, Mr Hudson Joshua Creighton    1st 30.00   male
## 4 Allison, Mrs Hudson JC (Bessie Waldo Daniels)    1st 25.00 female
## 5                 Allison, Master Hudson Trevor    1st  0.92   male
## 6                           Anderson, Mr Harry    1st 47.00   male
##   Survived Sex_Female Sex_Male
## 1        1          1        0
## 2        0          1        0
## 3        0          0        1
## 4        0          1        0
## 5        1          0        1
## 6        1          0        1
```

Or we can automate the process by building a simple loop to break each variable by unique values and creating a new column for each:

```
Titanic_dataset_temp <- Titanic_dataset

for (newcol in unique(Titanic_dataset_temp$PClass)) {
     feature_name <- 'PClass'
     Titanic_dataset_temp[,paste0(feature_name,"_",newcol)] <- ifelse(Titanic_data
set_temp[,feature_name]==newcol,1,0)
}

head(Titanic_dataset_temp)
```

```
##                                             Name PClass   Age    Sex
## 1                   Allen, Miss Elisabeth Walton    1st 29.00 female
## 2                    Allison, Miss Helen Loraine    1st  2.00 female
## 3            Allison, Mr Hudson Joshua Creighton    1st 30.00   male
## 4 Allison, Mrs Hudson JC (Bessie Waldo Daniels)    1st 25.00 female
## 5                  Allison, Master Hudson Trevor    1st  0.92   male
## 6                              Anderson, Mr Harry    1st 47.00   male
##   Survived PClass_1st PClass_2nd PClass_3rd
## 1        1          1          0          0
## 2        0          1          0          0
## 3        0          1          0          0
## 4        0          1          0          0
## 5        1          1          0          0
## 6        1          1          0          0
```

We successfully binarized a categorical variable with two lines of code. Let's build a real function to handle everything automatically. One word of caution, especially when running linear models, is the **dummy trap**. The general rule for creating dummy variables is to have one less variable than the number of categories present to avoid perfect collinearity. This isn't an issue for tree-based classifying algorithms.

Here is our full-fledged function that can handle missing variables and the **dummy trap**.

```
Binarize_Features <- function(data_set, features_to_ignore=c(), leave_out_one_leve
l=FALSE) {

     text_features <- c(names(data_set[sapply(data_set, is.character)]), names(dat
a_set[sapply(data_set, is.factor)]))
     for (feature_name in setdiff(text_features, features_to_ignore)) {
          feature_vector <- as.character(data_set[,feature_name])

          # check that data has more than one level
          if (length(unique(feature_vector)) == 1)
               next

          # We set any non-data to text
          feature_vector[is.na(feature_vector)] <- 'NA'
          feature_vector[is.infinite(feature_vector)] <- 'INF'
          feature_vector[is.nan(feature_vector)] <- 'NAN'

          # loop through each level of a feature and create a new column
          first_level=TRUE
          for (newcol in unique(feature_vector)) {
               if (first_level && leave_out_one_level) {
                    # avoid dummy trap and skip first level
                    first_level=FALSE
               } else {
                    data_set[,paste0(feature_name,"_",newcol)] <- ifelse(feature_v
ector==newcol,1,0)
               }
          }
          # remove original feature
          data_set <- data_set[,setdiff(names(data_set),feature_name)]
     }
     return (data_set)
}

Titanic_dataset_temp <- Binarize_Features(data_set = Titanic_dataset, features_to_
ignore = c('Name'))

str(Titanic_dataset_temp)
```

```
## 'data.frame':    1313 obs. of  8 variables:
##  $ Name      : Factor w/ 1310 levels "Abbing, Mr Anthony",..: 22 25 26 27 24 31
45 46 50 54 ...
##  $ Age       : num  29 2 30 25 0.92 47 63 39 58 71 ...
##  $ Survived  : int  1 0 0 0 1 1 1 0 1 0 ...
##  $ PClass_1st: num  1 1 1 1 1 1 1 1 1 1 ...
##  $ PClass_2nd: num  0 0 0 0 0 0 0 0 0 0 ...
##  $ PClass_3rd: num  0 0 0 0 0 0 0 0 0 0 ...
##  $ Sex_female: num  1 1 0 1 0 0 1 0 1 0 ...
##  $ Sex_male  : num  0 0 1 0 1 1 0 1 0 1 ...
```

If you need to use linear or logistic regression models, check out the dummyVars (http://www.inside-r.org/packages/cran/caret/docs/dummyVars) function from the caret (http://topepo.github.io/caret/index.html) package. It offers a lot more bells and whistles than our basic function.