Automating Data Exploration with R

Pipeline Check

Let's load our pipeline functions:

```
Fix_Date_Features <- function(data_set) {
     text_features <- c(names(data_set[sapply(data_set, is.character)]), names(dat
a_set[sapply(data_set, is.factor)]))
     for (feature_name in text_features) {
          feature_vector <- as.character(data_set[,feature_name])
          # assuming date pattern: '01/11/2012'
          date_pattern <- '[0-9][0-9]/[0-9][0-9]/[0-9][0-9][0-9][0-9]'
          if (max(nchar(feature_vector)) == 10) {
               if (sum(grepl(date_pattern, feature_vector)) > 0) {
                    print(paste('Casting feature to date:',feature_name))
                    data_set[,feature_name] <-  as.Date(feature_vector, format="%
d/%m/%Y")
               }
          }
     }
     return (data_set)
}
```

```
Get_Free_Text_Measures <- function(data_set, minimum_unique_threshold=0.9, feature
s_to_ignore=c()) {

     # look for text entries that are mostly unique
     text_features <- c(names(data_set[sapply(data_set, is.character)]), names(dat
a_set[sapply(data_set, is.factor)]))
     for (f_name in setdiff(text_features, features_to_ignore)) {
          f_vector <- as.character(data_set[,f_name])

          # treat as raw text if data over minimum_precent_unique unique
          if (length(unique(as.character(f_vector))) > (nrow(data_set) * minimum_u
nique_threshold)) {
               data_set[,paste0(f_name, '_word_count')] <- sapply(strsplit(f_vecto
r, " "), length)
               data_set[,paste0(f_name, '_character_count')]  <-  nchar(as.charact
er(f_vector))
               data_set[,paste0(f_name, '_first_word')] <- sapply(strsplit(as.char
acter(f_vector), " "), `[`, 1)
               # remove orginal field
               data_set[,f_name] <- NULL
          }
     }
     return(data_set)
}
```

```
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)
}
```

We now have two functions to prepare our data for modeling, let's recap. I added the date transformation function just for illustration. We also need to go over raw text before categorical text as after cleaning up the raw text we end up with a new categorical field, the first word.

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

# fix date field if any
Titanic_dataset_temp <- Fix_Date_Features(data_set = Titanic_dataset_temp)

# extra quantative value out of text entires
Titanic_dataset_temp <- Get_Free_Text_Measures(data_set = Titanic_dataset_temp)

# binarize categories
Titanic_dataset_temp <- Binarize_Features(data_set = Titanic_dataset_temp, feature
s_to_ignore = c(), leave_out_one_level = TRUE)
```

Capping Categories

What to do when there are too many categories? Plenty of possibilities and here we'll go with the most popular categories. So if there are over 1000 categories, we'll find the top 20 (or whatever you choose) and neutralize all the other categories. Let's re-run the **pipeline check** up to `Get_Free_Text_Measures` and figure out the most popular entries:

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

# fix date field if any
Titanic_dataset_temp <- Fix_Date_Features(data_set = Titanic_dataset_temp)

# extra quantative value out of text entires
Titanic_dataset_temp <- Get_Free_Text_Measures(data_set = Titanic_dataset_temp)

# get the Name_first_word feature
temp_vect <- Titanic_dataset_temp$Name_first_word
# only give us the top 20 most popular categories
popularity_count <- 20

# install.packages('dplyr')
library(dplyr)
```

```
##
## Attaching package: 'dplyr'
##
## The following objects are masked from 'package:stats':
##
##     filter, lag
##
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
temp_vect <- data.frame(table(temp_vect)) %>% arrange(desc(Freq)) %>% head(popular
ity_count)
Titanic_dataset_temp$Name_first_word <- ifelse(Titanic_dataset_temp$Name_first_wor
d %in% temp_vect$temp_vect,
                                               Titanic_dataset_temp$Name_first_word, 'O
ther')
print(head(Titanic_dataset_temp$Name_first_word,40))
```

```
##  [1] "Other"  "Other"  "Other"  "Other"  "Other"  "Other"  "Other"
##  [8] "Other"  "Other"  "Other"  "Other"  "Other"  "Other"  "Other"
## [15] "Other"  "Other"  "Other"  "Other"  "Other"  "Other"  "Other"
## [22] "Other"  "Other"  "Other"  "Other"  "Other"  "Other"  "Other"
## [29] "Other"  "Other"  "Other"  "Other"  "Other"  "Other"  "Other"
## [36] "Other"  "Brown," "Brown," "Other"  "Other"
```

Then we can run through our last pipeline function `Binarize_Features` and look at our transformed data set:

```
# binarize categories
Titanic_dataset_temp <- Binarize_Features(data_set = Titanic_dataset_temp, feature
s_to_ignore = c(), leave_out_one_level = TRUE)
head(Titanic_dataset_temp, 2)
```

```
##   Age Survived Name_word_count Name_character_count PClass_2nd PClass_3rd
## 1  29        1               4                   28          0          0
## 2   2        0               4                   27          0          0
##   Sex_male Name_first_word_Brown, Name_first_word_Carlsson,
## 1        0                      0                         0
## 2        0                      0                         0
##   Name_first_word_Carter, Name_first_word_Fortune, Name_first_word_Van
## 1                       0                        0                   0
## 2                       0                        0                   0
##   Name_first_word_Williams, Name_first_word_Davies, Name_first_word_Kelly,
## 1                         0                        0                      0
## 2                         0                        0                      0
##   Name_first_word_Andersson, Name_first_word_Asplund,
## 1                          0                        0
## 2                          0                        0
##   Name_first_word_Ford, Name_first_word_Goodwin,
## 1                     0                        0
## 2                     0                        0
##   Name_first_word_Johansson, Name_first_word_Johnson,
## 1                          0                        0
## 2                          0                        0
##   Name_first_word_Kink, Name_first_word_Lefebre, Name_first_word_Panula,
## 1                     0                        0                       0
## 2                     0                        0                       0
##   Name_first_word_Rice, Name_first_word_Sage, Name_first_word_Skoog,
## 1                     0                     0                      0
## 2                     0                     0                      0
```

So, let's add this popularity feature to our `Binarize_Features` function by adding a new parameter `max_level_count` (20 is completely arbitrary here, if you have the computing muscle and the need, you could have categories in the 1000's):

```
Binarize_Features <- function(data_set, features_to_ignore=c(), leave_out_one_leve
l=FALSE, max_level_count=20) {
     require(dplyr)
     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'

          # only give us the top x most popular categories
          temp_vect <- data.frame(table(feature_vector)) %>% arrange(desc(Freq)) %
>% head(max_level_count)
          feature_vector <- ifelse(feature_vector %in% temp_vect$feature_vector, f
eature_vector, 'Other')

          # loop through each level of a feature and create a new column
          first_level=TRUE
          for (newcol in unique(feature_vector)) {
               if (leave_out_one_level & first_level) {
                    # avoid dummy trap and skip first level
                    first_level=FALSE
                    next
               }

               data_set[,paste0(feature_name,"_",newcol)] <- ifelse(feature_vector
==newcol,1,0)
          }
          # remove original feature
          data_set <- data_set[,setdiff(names(data_set),feature_name)]
     }
     return (data_set)
}
```