Automating Data Exploration with R

Modeling

To those that have made it this far, a big congratulations! You've covered a large part of the mechanical and boring aspects of my workday. Now that we have a lot of that automated, we can not only move on the funner stuff, but also get to the funner stuff much faster the next time around!

Here we're going to look at a handful of typical models - the common supervised models: `Random Forest, GBM, GLMNET` and an unsupervised one: `K-means` clustering.

Random Forest

Let's start with Random Forest from the randomForest (http://www.inside-r.org/packages/cran/randomforest/docs/randomforest) package. For the subsequent models, we'll use the tremendously helpful library `caret` , but right now we'll go directly with RF.

We're going to need three of our pipeline functions: `Binarize_Features, Impute_Features, Get_Free_Text_Measures` - so load them into memory if you haven't already done so. Keep in mind that these functions are not libraries, they're meant to be customized to your needs when appropriate. Matter of fact, I customized `Binarize_Features` to not only return the first word in the sentence, but also the second one which is the title of the name field.

```
# functions ----------------------------------------------------------------

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)
}

Impute_Features <- function(data_set, features_to_ignore=c(),
                            use_mean_instead_of_0=TRUE,
                            mark_NAs=FALSE,
                            remove_zero_variance=FALSE) {

     for (feature_name in setdiff(names(data_set), features_to_ignore)) {
          print(feature_name)
          # remove any fields with zero variance
          if (remove_zero_variance) {
               if (length(unique(data_set[, feature_name]))==1) {
                    data_set[, feature_name] <- NULL
                    next
               }
          }
```

```
          if (mark_NAs) {
               # note each field that contains missing or bad data
               if (any(is.na(data_set[,feature_name]))) {
                    # create binary column before imputing
                    newName <- paste0(feature_name, '_NA')
                    data_set[,newName] <- as.integer(ifelse(is.na(data_set[,featur
e_name]),1,0)) }

               if (any(is.infinite(data_set[,feature_name]))) {
                    newName <- paste0(feature_name, '_inf')
                    data_set[,newName] <- as.integer(ifelse(is.infinite(data_set[,
feature_name]),1,0)) }
          }

          if (use_mean_instead_of_0) {
               data_set[is.infinite(data_set[,feature_name]),feature_name] <- NA
               data_set[is.na(data_set[,feature_name]),feature_name] <- mean(data_
set[,feature_name], na.rm=TRUE)


          } else {
               data_set[is.na(data_set[,feature_name]),feature_name] <- 0
               data_set[is.infinite(data_set[,feature_name]),feature_name] <- 0
          }
     }
     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)
               data_set[,paste0(f_name, '_second_word')] <- sapply(strsplit(as.cha
racter(f_vector), " "), `[`, 2)
               # remove orginal field
               data_set[,f_name] <- NULL
```

```
        }
    }
    return(data_set)
}
```

Let's load the Titanic data set again. Take a quick peek at it before loading it in memory with `readLines`:

```
# data --------------------------------------------------------------------

# using dataset from the UCI Machine Learning Repository (http://archive.ics.uci.edu/ml/)
readLines('http://math.ucdenver.edu/RTutorial/titanic.txt', n=5)
```

```
## [1] "Name\tPClass\tAge\tSex\tSurvived"
## [2] "\"Allen, Miss Elisabeth Walton\"\t1st\t29\tfemale\t1"
## [3] "\"Allison, Miss Helen Loraine\"\t1st\t2\tfemale\t0"
## [4] "\"Allison, Mr Hudson Joshua Creighton\"\t1st\t30\tmale\t0"
## [5] "\"Allison, Mrs Hudson JC (Bessie Waldo Daniels)\"\t1st\t25\tfemale\t0"
```

```
# With readLines, we now know that the file has a header row and 5 columns separated by tabs.
titanicDF <- read.csv('http://math.ucdenver.edu/RTutorial/titanic.txt', sep='\t', header = TRUE)
head(titanicDF)
```

```
##                                             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
```

We have on free-form text field - `Name`. We'll transform it using `Get_Free_Text_Measures`. We'll binarize

everything left that isn't numeric with `Binarize_Features`, and impute any missing data with `Impute_Features`. I am throwing away `Name_first_word` as I don't believe it brings any value to the model.

```
titanicDF <- Get_Free_Text_Measures(titanicDF)
titanicDF$Name_first_word <- NULL
titanicDF <- Binarize_Features(titanicDF, leave_out_one_level = TRUE)
titanicDF <- Impute_Features(titanicDF, use_mean_instead_of_0 = FALSE)
```

```
## [1] "Age"
## [1] "Survived"
## [1] "Name_word_count"
## [1] "Name_character_count"
## [1] "Name_second_word_Mr"
## [1] "Name_second_word_Mrs"
## [1] "Name_second_word_Master"
## [1] "Name_second_word_Colonel"
## [1] "Name_second_word_Dr"
## [1] "Name_second_word_Major"
## [1] "Name_second_word_(Bowerman),"
## [1] "Name_second_word_Captain"
## [1] "Name_second_word_Villiers,"
## [1] "Name_second_word_Gordon,"
## [1] "Name_second_word_y"
## [1] "Name_second_word_Jonkheer"
## [1] "Name_second_word_(Russell),"
## [1] "Name_second_word_the"
## [1] "Name_second_word_Col"
## [1] "Name_second_word_Derhoef,"
## [1] "Name_second_word_Ms"
## [1] "Name_second_word_(Icabod),"
## [1] "Name_second_word_Mlle"
## [1] "Name_second_word_Rev"
## [1] "Name_second_word_Brito,"
## [1] "Name_second_word_Carlo,"
## [1] "Name_second_word_(?Douton),"
## [1] "Name_second_word_(Nasrallah),"
## [1] "Name_second_word_(Schmidt),"
## [1] "Name_second_word_(Kalil),"
## [1] "Name_second_word_Ernst"
## [1] "Name_second_word_(Kareem),"
## [1] "Name_second_word_Messemaeker,"
## [1] "Name_second_word_Mulder,"
## [1] "Name_second_word_Thomas"
## [1] "Name_second_word_Hilda"
## [1] "Name_second_word_Delia"
## [1] "Name_second_word_Jenny"
## [1] "Name_second_word_Oscar"
## [1] "Name_second_word_Nils"
## [1] "Name_second_word_Eino"
## [1] "Name_second_word_(Borak),"
## [1] "Name_second_word_Albert"
## [1] "Name_second_word_W"
## [1] "Name_second_word_Sander"
## [1] "Name_second_word_Richard"
## [1] "Name_second_word_Mansouer"
## [1] "Name_second_word_Nikolai"
## [1] "Name_second_word_(Joseph),"
## [1] "Name_second_word_(Trembisky),"
```

```
## [1] "Name_second_word_Khalil"
## [1] "Name_second_word_Simon"
## [1] "Name_second_word_William"
## [1] "Name_second_word_(Sitik),"
## [1] "Name_second_word_(Thomas),"
## [1] "Name_second_word_Billiard,"
## [1] "Name_second_word_der"
## [1] "Name_second_word_de"
## [1] "Name_second_word_Impe,"
## [1] "Name_second_word_Leo"
## [1] "PClass_2nd"
## [1] "PClass_3rd"
## [1] "Sex_male"
```

We split the data set three ways - `training, validation and live` . The validation set will be used to tune the RF model.

```
# split data set
set.seed(1234)
random_splits <- runif(nrow(titanicDF))
train_data <- titanicDF[random_splits < .5,]
tune_data <- titanicDF[random_splits >= .5 & random_splits < .8,]
test_data <- titanicDF[random_splits >= .8,]
```

`randomForest` 's `tuneRF` will give us the optimal `mtry` setting to use. Here we dedicate our tune_data set for this task:

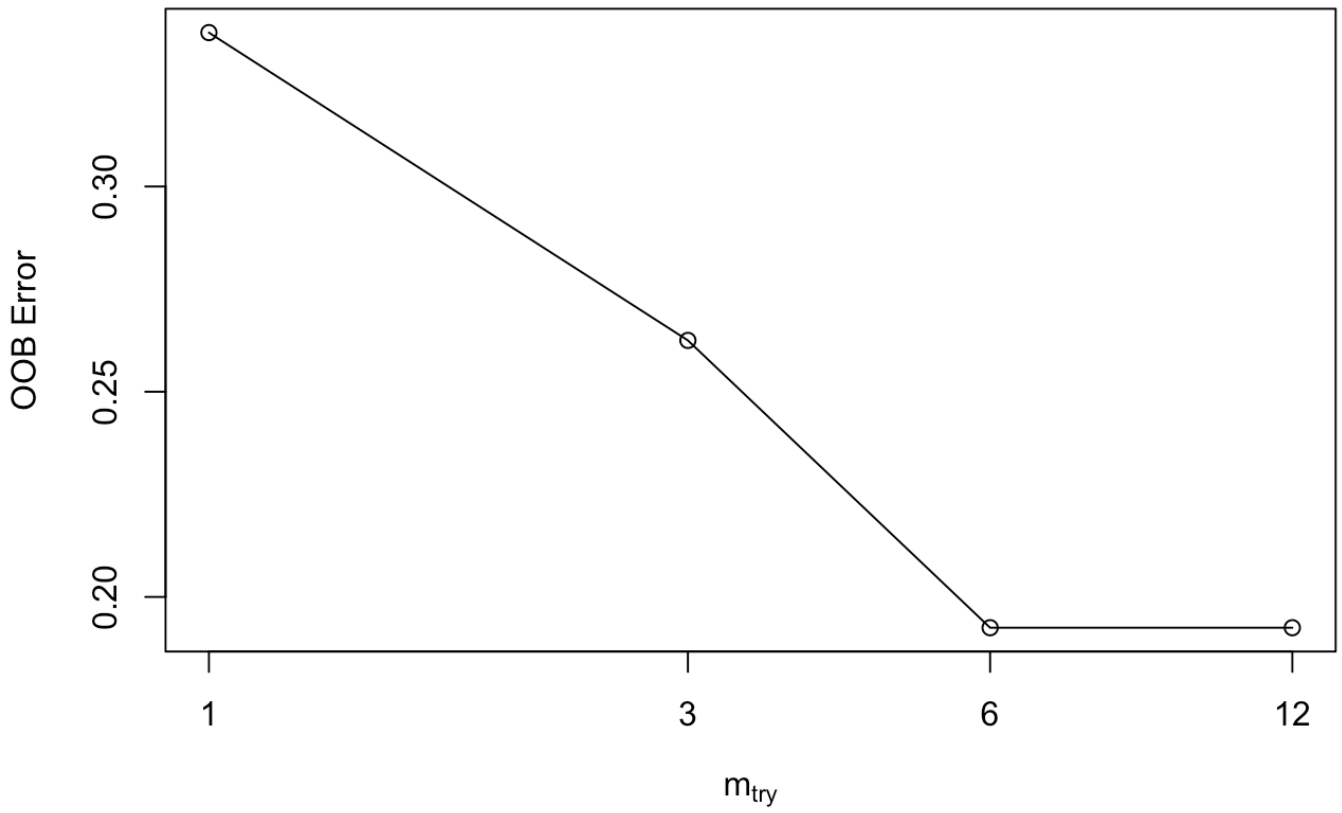
```
# install.packages('randomForest')
library(randomForest)
```

```
## randomForest 4.6-12
## Type rfNews() to see new features/changes/bug fixes.
```

```
set.seed(1234)
outcome_name <- 'Survived'
feature_names <- setdiff(names(train_data), outcome_name)

tnRF <- tuneRF(x=tune_data[,feature_names],
       y = as.factor(tune_data[,outcome_name]),
       mtryStart = 3, stepFactor = 0.5)
```

```
## mtry = 3  OOB error = 26.25%
## Searching left ...
## mtry = 6     OOB error = 19.25%
## 0.2666667 0.05
## mtry = 12    OOB error = 19.25%
## 0 0.05
## Searching right ...
## mtry = 1     OOB error = 33.75%
## -0.7532468 0.05
```




```
best_mtry <- tnRF[tnRF[, 2] == min(tnRF[, 2]), 1][[1]]
print(best_mtry)
```

```
## [1] 6
```

We now can call the RF model using the optimal `mtry` setting. We also set `importance` to true (http://www.inside-r.org/packages/cran/randomforest/docs/importance (http://www.inside-r.org/packages/cran/randomforest/docs/importance)) to access variable importance according to RF:

```
rf_model <- randomForest(x=train_data[,feature_names],
                         y=as.factor(train_data[,outcome_name]),
                         importance=TRUE, ntree=100, mtry = best_mtry)

print(importance(rf_model, type=1)[importance(rf_model, type=1)!=0,])
```

```
##                      Age         Name_word_count    Name_character_count
##                 4.472367                4.586581                3.786784
##     Name_second_word_Mr    Name_second_word_Mrs Name_second_word_Master
##                 6.880982                6.093252                1.222808
##     Name_second_word_Dr      Name_second_word_y     Name_second_word_Ms
##                 2.447301                2.817508                1.202244
##   Name_second_word_Mlle    Name_second_word_Rev              PClass_2nd
##                -1.145309                4.133762                3.677996
##              PClass_3rd                Sex_male
##                 8.137837                7.619451
```

Let's test the model on our test_data and use the `pROC` library to get an `AUC` score:

```
predictions <- predict(rf_model, newdata=test_data[,feature_names], type="prob")

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

```
## Type 'citation("pROC")' for a citation.
##
## Attaching package: 'pROC'
##
## The following objects are masked from 'package:stats':
##
##     cov, smooth, var
```

```
print(roc(response = test_data[,outcome_name], predictor = predictions[,2]))
```

```
##
## Call:
## roc.default(response = test_data[, outcome_name], predictor = predictions[,
2])
##
## Data: predictions[, 2] in 174 controls (test_data[, outcome_name] 0) < 92 cases
(test_data[, outcome_name] 1).
## Area under the curve: 0.8815
```