Bank Marketing Data Set Model

Jose Arizpe

4 January 2020

Executive Summary

The purpose of this project is to create a machine learning algorithm using the Bank Marketing Data Set[1] from the UC Irving Machine Language Repository[2].

The data is related with direct marketing campaigns of a Portuguese banking institution. The marketing campaigns were based on phone calls. Often, more than one contact to the same client was required, in order to access if the product (bank term deposit) would be ('yes') or not ('no') subscribed.

Two data sets were used in this analysis:

1. bank-additional-full.csv with all examples (41188) and 20 inputs, ordered by date (from May 2008 to November 2010)
2. bank-additional.csv with 10% of the examples (4119), randomly selected from 1) above, and 20 inputs.

The classification goal is to predict if a client will subscribe (yes/no) a term deposit (variable y).

The particular purpose of this analysis is to test the effectiveness of Gradient Boosting Machines (GBM) methods, which I have been recommended to use on bank data, against common tree-based methods such as randomForest and rpart.

With this in mind, I have proceeded to analyse the Portuguese bank marketing data using the methods listed in the next section, with Accuracy being the chosen measure of performance for the analysis.

In general, I have found that the results in terms of Accuracy from GBM-based methods has been satisfactory and amongst the highest of all the methods tested. However, the best result of this analysis was consistently delivered by randomForest, with almost 1% higher Accuracy than the next best option. This is said taking into account that I have tested a couple of implementations of GBM, two different distributions for classification objectives, and also used the caret "train" function for fine-tuning parameters and scenario testing[3].

Having said this, the tests performed show that gbm results can improve considerably when fine-tuned, as evidenced by the 2% improvement in accuracy after tuning, and the many more tuning possibilities yet to be explored. That said, randomForest is still 1% ahead of gbm even after tuning, so a further increase of that magnitude does not seem like a quick win, and will be left as something to be explored in a future version of this analysis.

In the meantime, the conclusion of this particular analysis is that the best performing method for modelling the Portuguese Bank Marketing data is randomForest, which delivers an Accuracy of 0.93250.

Methods / Analysis

The present analysis includes the modelling of the Portuguese Bank data using the following methods:

1. randomForest

[1] [Moro et al., 2014] S. Moro, P. Cortez and P. Rita. A Data-Driven Approach to Predict the Success of Bank Telemarketing. Decision Support Systems, Elsevier, 62:22-31, June 2014)

[2] Dua, D. and Graff, C. (2019). UCI Machine Learning Repository [http://archive.ics.uci.edu/ml]. Irvine, CA: University of California, School of Information and Computer Science.

[3] Please note that both the embedded and the attached R code execute only the instructions within the application of each method that deliver the highest Accuracy result. The instructions for tuning and finding those best performing parameters have been commented-out in order to optimise the script performance.

2. gbm with bernoulli distribution and tuned with cross validation

3. gbm with adaboost distribution and tuned with cross validation

4. rpart

5. gbm - Gradient Boosting Machines

6. gbm_h2o, which is an h2o-based implementation of gbm

7. Rborist

The following general steps were performed to carry out this analysis:

1. Load and clean data
   i) Load data.

   ii) Clean/normalise data and data classes, cast variables as factors where possible to facilitate the analyses.

   iii) Remove na values and replace them with zero where relevant.

   iv) Assess the impact of variables with near zero values. Here I have found that the removal of these does not change the result significantly and moreover does not provide a better fit for any of the methods used.

2. Excute modelling work. Model the data using the methods mentioned above.

The steps' details follow:

1- Load and clean data

```
#----------------------------------------------------------------------
# Load data
#
## Load train dataset
file_url <- "https://archive.ics.uci.edu/ml/machine-learning-databases/00222/bank-additional.zip"
temp <- tempfile()
download.file(file_url, temp)
unzip(temp, "bank-additional/bank-additional-full.csv")
train_set <- read_csv2("bank-additional/bank-additional-full.csv", col_names = TRUE)
```

```
## Using ',' as decimal and '.' as grouping mark. Use read_delim() for more control.
```

```
## Parsed with column specification:
## cols(
##   .default = col_character(),
##   age = col_double(),
##   duration = col_double(),
##   campaign = col_double(),
##   pdays = col_double(),
##   previous = col_double(),
##   emp.var.rate = col_number(),
##   cons.price.idx = col_number(),
##   cons.conf.idx = col_number(),
##   euribor3m = col_number(),
```

```
##   nr.employed = col_double()
## )
```

```
## See spec(...) for full column specifications.
```

```
## Load test dataset
unzip(temp, "bank-additional/bank-additional.csv")
test_set <- read_csv2("bank-additional/bank-additional.csv", col_names = TRUE)
```

```
## Using ',' as decimal and '.' as grouping mark. Use read_delim() for more control.
```

```
## Parsed with column specification:
## cols(
##   .default = col_character(),
##   age = col_double(),
##   duration = col_double(),
##   campaign = col_double(),
##   pdays = col_double(),
##   previous = col_double(),
##   emp.var.rate = col_number(),
##   cons.price.idx = col_number(),
##   cons.conf.idx = col_number(),
##   nr.employed = col_number()
## )
```

```
## See spec(...) for full column specifications.
```

```
#------------------------------------------------------------------------
# Clean/normalise data and data classes
#
train_set <- train_set %>% mutate(age = as.integer(age),
                                  job = as.factor(job),
                                  marital = as.factor(marital),
                                  education = as.factor(education),
                                  default = as.factor(default),
                                  housing = as.factor(housing),
                                  loan = as.factor(loan),
                                  contact = as.factor(contact),
                                  month = as.factor(month),
                                  day_of_week = as.factor(day_of_week),
                                  duration = as.integer(duration),
                                  campaign = as.integer(campaign),
                                  pdays = as.integer(pdays),
                                  previous = as.integer(previous),
                                  emp.var.rate = as.integer(emp.var.rate),
                                  cons.price.idx = as.integer(cons.price.idx),
                                  cons.conf.idx = as.integer(cons.conf.idx),
                                  euribor3m = as.integer(euribor3m),
                                  nr.employed = as.integer(nr.employed),
                                  poutcome = as.factor(poutcome),
                                  y = as.factor(y))
test_set <- test_set %>% mutate(age = as.integer(age),
                                  job = as.factor(job),
                                  marital = as.factor(marital),
                                  education = as.factor(education),
                                  default = as.factor(default),
                                  housing = as.factor(housing),
```

```
                                    loan = as.factor(loan),
                                    contact = as.factor(contact),
                                    month = as.factor(month),
                                    day_of_week = as.factor(day_of_week),
                                    duration = as.integer(duration),
                                    campaign = as.integer(campaign),
                                    pdays = as.integer(pdays),
                                    previous = as.integer(previous),
                                    emp.var.rate = as.integer(emp.var.rate),
                                    cons.price.idx = as.integer(cons.price.idx),
                                    cons.conf.idx = as.integer(cons.conf.idx),
                                    euribor3m = as.integer(euribor3m),
                                    nr.employed = as.integer(nr.employed),
                                    poutcome = as.factor(poutcome),
                                    y = as.factor(y))
train_set2 <- train_set
## Remove na's
train_set2$nr.employed[is.na(train_set2$nr.employed)] <- 0
```

2- Execute modelling work

A) Fit with randomForest

```
#-------------------------------------------------------------------------
# Fit with randomForest
#
set.seed(1)
ns <- 31 # Best tune for nodesize
fit_rf <- randomForest(y ~ ., data=train_set2, nodesize = ns)
y_hat <- predict(fit_rf, test_set)
#    Calculate performance metrics
acc <- confusionMatrix(y_hat, test_set$y)$overall["Accuracy"]
```

```
## Accuracy randomForest = 0.932507890264627
```

B) Fit with gbm / bernoulli distribution

```
#-------------------------------------------------------------------------
# Fit with gbm + bernoulli distribution with tuning
#
fit_gbm <- gbm(
  unclass(y)-1 ~ .,
  distribution = "bernoulli",
  data = train_set2,
  n.trees = 59994,         # Best tune for n.trees
  interaction.depth = 1,   # Best tune for interaction.depth
  shrinkage = 0.01,        # Best tune for shrinkage
  n.minobsinnode = 1,      # Best tune for n.minobsinnode
  n.cores = NULL,          # will use all cores by default
  verbose = FALSE
)
y_hat <- predict(fit_gbm, test_set, n.trees = 59994, type = "response")
y_hat <- factor(ifelse(y_hat>=0.5, "yes", "no"))
#    Calculate performance metrics
acc <- confusionMatrix(y_hat, test_set$y)$overall["Accuracy"]
```

```
## Accuracy tuned gbm/bernoulli = 0.917212915756252
```

```
## Accuracy pre-tuned gbm/bernoulli = 0.9072590
```

```
## Difference = 0.0099539
```

C) Fit with gbm / adaboost distribution

```
#-----------------------------------------------------------------------
# Fit with gbm + adaboost distribution with tuning
#
set.seed(1)
fit_gbm <- gbm(
  unclass(y)-1 ~ .,
  distribution = "adaboost",
  data = train_set2,
  n.trees = 59972,         # Best tune for n.trees
  interaction.depth = 3,   # Best tune for interaction.depth
  shrinkage = 0.01,        # Best tune for shrinkage
  n.minobsinnode = 1,      # Best tune for n.minobsinnode
  n.cores = NULL,          # will use all cores by default
  verbose = FALSE
)
y_hat <- predict(fit_gbm, test_set, n.trees = 59972)
y_hat <- plogis(2*y_hat) # Convert predictions from logit scale to probability
y_hat <- factor(ifelse(y_hat>=0.5, "yes", "no"))
#    Calculate performance metrics
acc <- confusionMatrix(y_hat, test_set$y)$overall["Accuracy"]
```

```
## Accuracy tuned gbm/adaboost = 0.922796795338674
```

```
## Accuracy pre-tuned gbm/adaboost = 0.9126001
```

```
## Difference = 0.0101967
```

D) Fit with rpart

```
#-----------------------------------------------------------------------
# Fit with rpart
#
set.seed(1)
cp <- 0.002083333 # fit$bestTune
fit_rpart <- rpart(y~., data = train_set, cp = cp)
y_hat <- predict(fit_rpart, test_set)
y_hat <- ifelse(y_hat[,1]>y_hat[,2],"no", "yes")
y_hat <- factor(y_hat)
#    Calculate performance metrics
acc <- confusionMatrix(y_hat, test_set$y)$overall["Accuracy"]
```

```
## Accuracy rpart = 0.914056809905317
```

E) Fit with gbm

```
#-----------------------------------------------------------------------
# Fit with gbm
#
```

```
set.seed(1)
fit <- train(y~., method = "gbm", data = train_set2, verbose = FALSE)
y_hat <- predict(fit, test_set)
#    Calculate performance metrics
acc <- confusionMatrix(y_hat, test_set$y)$overall["Accuracy"]
```

```
## Accuracy gbm = 0.91138625880068
```

F) Fit with gbm_h20

```
#------------------------------------------------------------------------
# Fit with gbm_h2o
#
set.seed(1)
fit <- train(y~., method = "gbm_h2o", data = train_set2, verbose = FALSE)
y_hat <- predict(fit, test_set)
#    Calculate performance metrics
acc <- confusionMatrix(y_hat, test_set$y)$overall["Accuracy"]
```

```
## Accuracy gbm_h2o = 0.897305171158048
```

G) Fit with Rborist

```
#------------------------------------------------------------------------
# Fit with Rborist
#
set.seed(1)
pf <- 20 # fit$bestTune$predFixed
mn <- 12 # fit$bestTune$minNode
fit_rf <- Rborist(x = select(train_set2, -y), y = train_set2$y, type="class", nTree=1000, minNode=mn, pr
y_hat <- predict(fit_rf, select(test_set, -y))
#    Calculate performance metrics
acc <- confusionMatrix(y_hat$yPred, test_set$y)$overall["Accuracy"]
```

```
## Accuracy Rborist = 0.885894634620053
```

Results

The following table shows the summary of the results:

```
##                 method  accuracy
## 1         randomForest 0.9325079
## 2  tuned gbm/adaboost 0.9227968
## 3 tuned gbm/bernoulli 0.9172129
## 4                rpart 0.9140568
## 5                  gbm 0.9113863
## 6              gbm_h2o 0.8973052
## 7              Rborist 0.8858946
```

```
## The highest accuracy is delivered by randomForest at 0.932507890264627
```

Conclusion

The purpose of the analysis presented in this report is to create a machine learning algorithm that predicts if a client will subscribe (yes/no) a term deposit (variable y), using the Portuguese Bank Marketing Data Set from the UC Irving Machine Language Repository. As part of this effort, various methods based on

Gradient Boosting Machines (GBM, which have been recommended for use on bank data) have been tested for Accuracy alongside common tree-based methods such as randomForest, Rborist and rpart.

As a result of this analysis I have found that the overall performance of the GBM-based methods is generally satisfactory on the Portuguese bank data. However, I have found that the best performance is delivered by randomForest with an Accuracy of 0.9325079.

Having said this, the tests performed show complexity in gbm which (potentially) if managed could lead to better results which have been obtained here. The gains of 2% in accuracy after tuning point in this direction, as well as the breadth of tuning possibilities which are yet to be explored. Nevertheless, plugging the 1% remaining gap against randomForest accuracy does not seem to be a quick win so this is something to be explored in a future version of this analysis.