Exercice pour MGF-Labs

Jason Xinghang DAI

25/09/2018

Objectives

- Toutes les hypothèses, ainsi que la façon dont vous mesurez l'efcacité de votre modèle prédictf
- Une descripton de votre approche et les types de modèles que vous avez envisagés
- Un bref résumé et une présentaton de vos conclusions
- Votre code, pour lequel vous pouvez utliser le langage de programmaton et les bibliothèques de votre choix

Le reste de ce document sera principalement rédigée en anglais, comme la pluspart de resource de recherche que j'ai utilisé pour reéaliser ce projet est en anglais

Background

Avant de commencer construire notre modeèle de machine learning, il faut comprendre le context de notre problématique. Il s'agit de prediction de la conversation (l'action de clicker sur un pub). Alors qu'est ce que c'est le RTB?

RTB

As an ad impression loads in a user's Web browser, information about the page it is on and the user viewing it is passed to an ad exchange, which auctions it off to the advertiser willing to pay the highest price for it. The winning bidder's ad is then loaded into the webpage nearly instantly; the whole process takes just milliseconds to complete. Advertisers typically use demand-side platforms to help them decide which ad impressions to purchase and how much to bid on them based on a variety of factors, **such as the sites they appear on and the previous behavior of the users loading them**. Zappos might recognize that a user has previously been on its site looking at a specific pair of shoes, for example, and therefore may be prepared to pay more than Amazon or Best Buy to serve ads to him. The price of impressions is determined in real time based on what buyers are willing to pay, hence the name "real-time bidding. After some research online, I assume that two general categories of factors influence the innitiation of a"conversation": user's behaviors, and websites situation.

Exploratory data analysis

Let's take a look at the data, first, we'll load all the library that we need in this project:

```r
library(dplyr)
library(ggplot2)
library(caret)
library(tidyr)
library(gridExtra)
library(pROC)
```

Import the dataset:

```r
#setwd("./documents") # set work path
data <- read.csv("test_campaign.csv")
```

Explore the data

```r
names(data) # show the feature names
```

```
##  [1] "fold_position"        "buyer_bid"            "geo_region"
##  [4] "operating_system"     "browser"              "advertiser_frequency"
##  [7] "advertiser_recency"   "campaign_id"          "creative_id"
## [10] "creative_freq"        "creative_rec"         "geo_dma"
## [13] "geo_city"             "device_type"          "geo_postal_code"
## [16] "click"
```

```r
summary(data) # show statistical summary of each feature
```

```
##  fold_position      buyer_bid       geo_region     operating_system
##  Min.   :0.0000   Min.   : 0.13   Min.   : 1.000   Min.   : 1.00
##  1st Qu.:1.0000   1st Qu.:13.00   1st Qu.: 2.000   1st Qu.:42.00
##  Median :1.0000   Median :13.00   Median : 6.000   Median :51.00
##  Mean   :0.8441   Mean   :14.35   Mean   : 5.939   Mean   :51.64
##  3rd Qu.:1.0000   3rd Qu.:13.70   3rd Qu.: 7.000   3rd Qu.:62.00
##  Max.   :2.0000   Max.   :40.00   Max.   :16.000   Max.   :89.00
##                   NA's   :75
##     browser       advertiser_frequency advertiser_recency  campaign_id
##  Min.   : 1.000   Min.   :  0.00       Min.   :    0      Min.   :1.000
##  1st Qu.: 6.000   1st Qu.:  0.00       1st Qu.:    0      1st Qu.:3.000
##  Median : 6.000   Median :  1.00       Median :    2      Median :3.000
##  Mean   : 7.416   Mean   :  1.35       Mean   : 2255      Mean   :2.751
##  3rd Qu.: 7.000   3rd Qu.:  2.00       3rd Qu.:   42      3rd Qu.:3.000
##  Max.   :25.000   Max.   :164.00       Max.   :64071      Max.   :6.000
##
##   creative_id creative_freq     creative_rec      geo_dma
##  Min.   :1    Min.   : 0.000   Min.   :    0   Min.   :1.000
##  1st Qu.:1    1st Qu.: 0.000   1st Qu.:    0   1st Qu.:2.000
##  Median :1    Median : 1.000   Median :    1   Median :3.000
##  Mean   :1    Mean   : 1.113   Mean   : 1891   Mean   :3.254
##  3rd Qu.:1    3rd Qu.: 2.000   3rd Qu.:   29   3rd Qu.:5.000
##  Max.   :1    Max.   :11.000   Max.   :63827   Max.   :8.000
##
##     geo_city     device_type    geo_postal_code     click
##  Min.   :   1   Min.   :1.000   Min.   : 1067   Min.   :      0
##  1st Qu.:2126   1st Qu.:1.000   1st Qu.:34125   1st Qu.:      0
##  Median :3581   Median :2.000   Median :51103   Median :      0
##  Mean   :3284   Mean   :1.746   Mean   :52257   Mean   :     10
##  3rd Qu.:3871   3rd Qu.:2.000   3rd Qu.:71067   3rd Qu.:      0
##  Max.   :7395   Max.   :7.000   Max.   :99994   Max.   :5748394
##
```

This is a large dataset, and we there are missing values(NA). Due to the large volume of this dataset, we'll simply delete rows with NA values ($n = 74$) instead of imputing them.

```r
data_clean <- na.omit(data)
```

Another problem shown in the summary is that *click* have a quite abnormal max value(5748394), it's apparently an error, we should delete it.

```r
data_clean <- filter(data_clean, !grepl(5748394, click))
data_clean$click <- as.factor(data_clean$click) # transform the class into factors
```

Feature selection

It's a rather large dataset, in order to avoid over-fitting or excessive computation, we are going to select relevant features. According to my research, geographic information is supposed to have little influence on user's conversation with a pub. I assume that the distribution of geographic information is almost identical between click = 1 and click = 2. Let's have a look:

```r
# build a dataset with only geographic information and click
geo <- select(data_clean,geo_dma, geo_region, geo_postal_code, geo_city, click)
# group the dataset by click
geo <- group_by(geo, geo$click)
# Calculate the mean of each feature
summarise(geo, mean(geo_dma), mean(geo_region), mean(geo_postal_code), mean(geo_city))
```

```
## # A tibble: 2 x 5
##   `geo$click` `mean(geo_dma)` `mean(geo_regio~ `mean(geo_posta~
##   <fct>                 <dbl>            <dbl>            <dbl>
## 1 0                      3.26             5.95           52206.
## 2 1                      3.14             5.82           52842.
## # ... with 1 more variable: `mean(geo_city)` <dbl>
```

```r
# Calculate the standard deviation of each feature
summarise(geo, sd(geo_dma), sd(geo_region), sd(geo_postal_code), sd(geo_city))
```

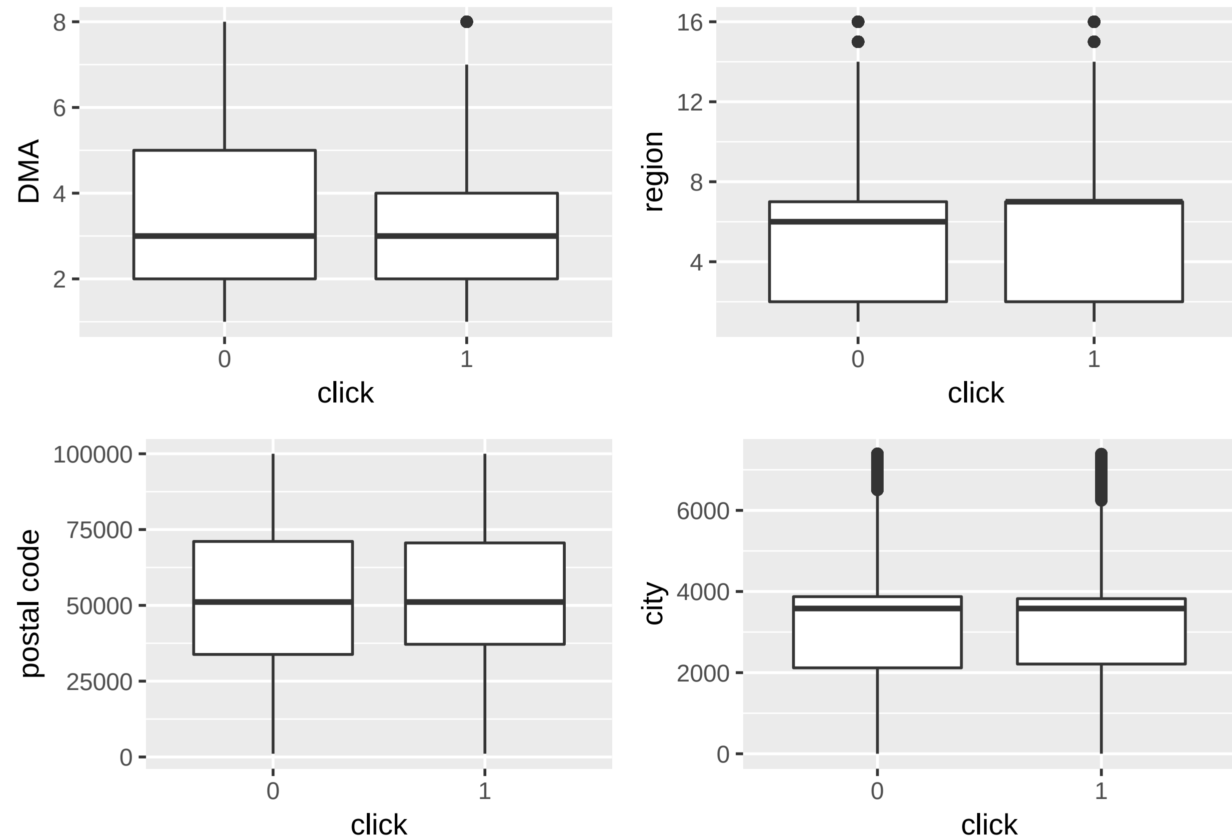
```
## # A tibble: 2 x 5
##   `geo$click` `sd(geo_dma)` `sd(geo_region)` `sd(geo_postal_~
##   <fct>               <dbl>            <dbl>            <dbl>
## 1 0                    1.78             3.87           24037.
## 2 1                    1.66             3.68           22914.
## # ... with 1 more variable: `sd(geo_city)` <dbl>
```

Judging from the result, I assume that the geographic information is not relevant for prediction, since they are practically the same between click = 0/1 Now let's visualize those features to verify this hypothesis

```r
p_dma <- ggplot(data = geo) +
    geom_boxplot(mapping = aes(y = geo$geo_dma, x = geo$click)) +
    labs(x = "click", y = "DMA")
p_region <- ggplot(data = geo) +
    geom_boxplot(mapping = aes(y = geo$geo_region, x = geo$click))+
    labs(x = "click", y = "region")
p_postalcode <- ggplot(data = geo) +
    geom_boxplot(mapping = aes(y = geo$geo_postal_code, x = geo$click))+
    labs(x = "click", y = "postal code")
p_city <- ggplot(data = geo) +
    geom_boxplot(mapping = aes(y = geo$geo_city, x = geo$click))+
    labs(x = "click", y = "city")
```

Now here is the plot:

```r
grid.arrange(p_dma, p_region, p_postalcode, p_city)
```



We can see that there are some outliers in dma, region and city. The boxplot shows that the action click doesn't really have an inlfuence on the distribution of geographic information, therefore they should be dropped.

```
data_clean <- select(data_clean, -starts_with("geo"))
```

Now,let's eliminate features that shows trival statistical significance. When we look at other features, we also need to look out for near zero variables, they will interfere with our model prediction

```
nearZeroVar(data_clean, saveMetrics = TRUE)
```

```
##                       freqRatio percentUnique zeroVar   nzv
## fold_position          5.297894  0.0005208686   FALSE FALSE
## buyer_bid              2.826950  0.0418431109   FALSE FALSE
## operating_system       1.285952  0.0154524351   FALSE FALSE
## browser                6.339880  0.0043405717   FALSE FALSE
## advertiser_frequency   1.902525  0.0133689607   FALSE FALSE
## advertiser_recency    17.883151  5.5722522879   FALSE FALSE
## campaign_id            5.385553  0.0010417372   FALSE FALSE
## creative_id            0.000000  0.0001736229    TRUE  TRUE
## creative_freq          2.079151  0.0020834744   FALSE FALSE
## creative_rec          19.355051  5.0531199161   FALSE  TRUE
## device_type            1.165310  0.0010417372   FALSE FALSE
## click                 11.439762  0.0003472457   FALSE FALSE
```

So we need to consider to drop creative_id to improve our prediction **However, maybe the advertisement identity is important information, this need to be discussed with clients** for computational reasons, i'll drop them in this project.

```
data_clean <- select(data_clean, -creative_id)
```

Now let's look for correlations among these features, if yes, maybe they should be dropped

```
cor <- cor(data_clean[,1:10])
# the cutoff is set at 0.8, relatively high
hcor <- findCorrelation(cor, cutoff=0.8, verbose = TRUE)
```

```
## Compare row 9  and column  6 with corr  0.803
##   Means:  0.178 vs 0.116 so flagging column 9
## All correlations <= 0.8
```

So what's these features are about? I did a brief reasearch online, and here are their explications: *creative_freq:times the user has seen this creative by this advertiser* advertiser_recency:how long it has been since the user saw an ad from this advertiser Judging from the code-book, these two features are clearly quite similar Therefore column 9 will be dropped

```
data_clean <- data_clean[, -hcor]
```

Finally names of the class levels should be converted to valid names

```
levels(data_clean$click) <- make.names(levels(factor(data_clean$click)))
```

features can still be dropped during the training process, with varImp* function* # Build prediction model Now let's build our prediction model This is clearly a binary classification problem, try to predict two classes click = "0" or "1"

```
prop.table(table(data_clean$click))
```

```
##
##         X0         X1
## 0.91961261 0.08038739
```

Highly imbalanced dataset

The result shows that this is a highly imbalanced dataset. Therefore "accuracy" can not be used as a error measure, we need other performance matrix and adapted algorithms. Tree algorithms are usually better suited for imbalanced data classification, and we can easily explain the classification mechanism to our clients. As for imbalanced binary classification, we can use down sampling and cost sensitive learning, or we can use ROC as the performance matrix to maximise the distinction between the two classes.

Now let's look at the data distribution

```
sapply(data_clean,sd)
```

```
##        fold_position            buyer_bid     operating_system
##            0.3695277            5.7402655           17.3105674
##              browser advertiser_frequency   advertiser_recency
##            3.5209551            2.1270070         7108.4073647
##          campaign_id        creative_freq          device_type
##            0.8126310            1.3863796            0.7575740
##                click
##            0.2718922
```

```
sapply(data_clean,class)
```

```
##        fold_position            buyer_bid     operating_system
##            "integer"            "numeric"            "integer"
```

```
##              browser advertiser_frequency   advertiser_recency
##            "integer"            "integer"            "integer"
##          campaign_id         creative_freq          device_type
##            "integer"            "integer"            "integer"
##                click
##             "factor"
```

Some columns are quite skewed, we need to do some standarization. ## Split the dataset We are going to split the data into two parts, train dataset (70%) and test dataset(30%). In the train dataset, we'll use k folds cross validation.

```r
set.seed(665)
inTrain <- createDataPartition(y = data_clean$click, p = 0.7, list = FALSE )
training <- data_clean[inTrain,]; testing <- data_clean[-inTrain,]
```

Choose error measurement and sampling control

Now let's define two suammary functions, one is used to train models with a maximum ROC, one is used for cost sensitive training.

```r
#For accuracy, Kappa, the area under the ROC curve,sensitivity and specificity:
fiveStats <- function(...)
    c(twoClassSummary(...),
      defaultSummary(...))
# Everything but the area under the ROC curve:
fourStats <- function (data, lev = levels(data$obs), model = NULL)
{
       accKapp <- postResample(data[, "pred"], data[, "obs"])
       out <- c(accKapp,
         sensitivity(data[, "pred"], data[, "obs"], lev[1]),
         specificity(data[, "pred"], data[, "obs"], lev[2]))
         names(out)[3:4] <- c("Sens", "Spec")
        out
}
```

We are going to use two control method, "down" method is chosen becasue **my computer(macbookair) is too slow** for "smote"or "rose"

```r
control_down <- trainControl(method = "cv",
                        number = 10,
                        sampling = "down",
                        verboseIter = FALSE,
                        summaryFunction = fiveStats,
                        classProbs = TRUE)

control_noprob <- trainControl(method = "cv",
                              number = 10,
                              sampling = "down",
                              verboseIter = FALSE,
                              summaryFunction = fourStats,
                              classProbs = FALSE)
```

Choose algorithms

For this kind of binary classification problem, expecially with a imbalanced dataset Tree classification is usually good, notably random forest. But I'll not use it, because it takes more than 5 hours to train the model on my macbook air. I think SVM is also quite good for cost sensitive learing, but it's takes too much time to train, and impossible to explain to our clients.
Therefore i'll demonstrate here one linear algorithm LDA(linear discriminant analysis), one tree algorithm RPART(recursive partitioning), and a boosting algorithm GBM (Stochastic Gradient Boosting). These algorithms will first be trained with a ROC performance matrix, then rpart will be trained with cost sensitive learning method. Finally we'll compare the result. First, train three models with ROC as performance matrix

```
#LDA
set.seed(666)
fit.lda <- train(click ~., data = training, method = "lda",
                   preProcess = c("center", "scale"),
                   metric = "ROC",
                   tuneLength = 5,
                   trControl = control_down )
pred_lda <- predict(fit.lda, testing)
con_lda <- confusionMatrix(pred_lda, testing$click, positive = "X1")
# RPART
set.seed(666)
fit.rpart <- train(click ~., data = training, method = "rpart",
                   preProcess = c("center", "scale"),
                   metric = "ROC",
                   tuneLength = 5,
                   trControl = control_down)
pred_rpart <- predict(fit.rpart, testing)
con_rpart <- confusionMatrix(pred_rpart, testing$click, positive = "X1")
```

```
#GBM
set.seed(666)
fit.gbm <- train(click ~., data = training, method = "gbm",
                   preProcess = c("center", "scale"),
                   metric = "ROC",
                   tuneLength = 5,
                 verbose = 0,
                   trControl = control_down)
pred_gbm <- predict(fit.rpart, testing)
con_gbm <- confusionMatrix(pred_gbm, testing$click, positive = "X1")
```

Now let's try cost sensitive learning. It's not possible to define a cost matrix without inputs from our business clients. Here I try to use a cost matrix to just show this method.

```
#rpartcost
set.seed(666)
fit.rpartcost <- train(click ~., data = training,
                method = "rpart",
                preProcess = c("center", "scale"),
                metric = "Kappa",
                parms = list(loss = matrix(c(0,1,2,0), nrow = 2)),
                trControl = control_noprob)
pred_rpartcost <- predict(fit.rpartcost, testing)
con_rpartcost <- confusionMatrix(pred_rpartcost, testing$click, positive = "X1")
```

Result comparison

Now let's look at their results.

```
# for the training result
lda = fit.lda$results
gbm = fit.gbm$results
rpart = fit.rpart$results
rpartcost = fit.rpartcost$results
# for the testing result, transpose the metrix
result <- t(data.frame(lda = con_lda$byClass, gbm = con_gbm$byClass,
                     rpart = con_rpart$byClass, rpartcost = con_rpartcost$byClass))
result
```

```
##           Sensitivity Specificity Pos Pred Value Neg Pred Value Precision
## lda         0.8106551   0.5822981      0.1450433      0.9723612 0.1450433
## gbm         0.7658747   0.6384536      0.1562413      0.9689401 0.1562413
## rpart       0.7658747   0.6384536      0.1562413      0.9689401 0.1562413
## rpartcost   0.5675306   0.7643646      0.1739217      0.9528726 0.1739217
##              Recall        F1 Prevalence Detection Rate
## lda       0.8106551 0.2460611 0.08038753     0.06516656
## gbm       0.7658747 0.2595362 0.08038753     0.06156678
## rpart     0.7658747 0.2595362 0.08038753     0.06156678
## rpartcost 0.5675306 0.2662501 0.08038753     0.04562238
##           Detection Prevalence Balanced Accuracy
## lda                  0.4492905         0.6964766
## gbm                  0.3940494         0.7021642
## rpart                0.3940494         0.7021642
## rpartcost            0.2623157         0.6659476
```

Conclusion

From the results, we can conclude that rpart model is very good at distinguishing the target classes, with a relatively high AUC. The cost sensitive learning model also give some good result, but the definitive cost matrix needs to be decided with our clients. Therefore, within the context of this project, I'll conclude with the fit.rpart prediction model.