AI Camp 2017 - TransferGO

G. Vilkelis

April 9, 2017

Summary

I've analysed customer data set provided by **transferGo**. The goal of analysis was to identify which factors (variables) influence the scoring of the customer.

The provided datased had very skewed distribution of (maximum) scores 10 (81% percent of all data). This means that a model always predicting value 10 would have a precision of 81%. This also means that any model which would have a precision rate below 81% would not be usefull at all.

I've trained a model using xgboost (using 80% random sample for training and 20% for testing). While the model showed that number of transactions and first seconds between booking and registration time were most important variables influencing the score, the model failed to generalize on unseen data.

Loading and cleaning data

The dataset was download from https://docs.google.com/spreadsheets/d/16kAUV7NqHHEEVAvSufmc7SZLCRCTthEis3IPKgr edit?usp=sharing on 8th of April, 2017.

```r
# dataset size
dim(d)
```

```
## [1] 7595    9
```

```r
# rename columns for easier coding
names(d)
```

```
## [1] "Score"
## [2] "country"
## [3] "User.s.lifetime.number.of.transactions"
## [4] "User.s.language"
## [5] "User.is.referred...1.True.0.False."
## [6] "Seconds.between.user.s.first.booking.date.and.created_at"
## [7] "User.s.address_status_id"
## [8] "USer.s.identity_status_id"
## [9] "User.s.lifetime.number.of.referrals"
```

```r
names(d) = c('score', 'country', 'transactions','language', 'is_referred',
             'first_booking', 'address_status', 'identity_status', 'referrals')
names(d)
```

```
## [1] "score"           "country"         "transactions"    "language"
## [5] "is_referred"     "first_booking"   "address_status"  "identity_status"
## [9] "referrals"
```

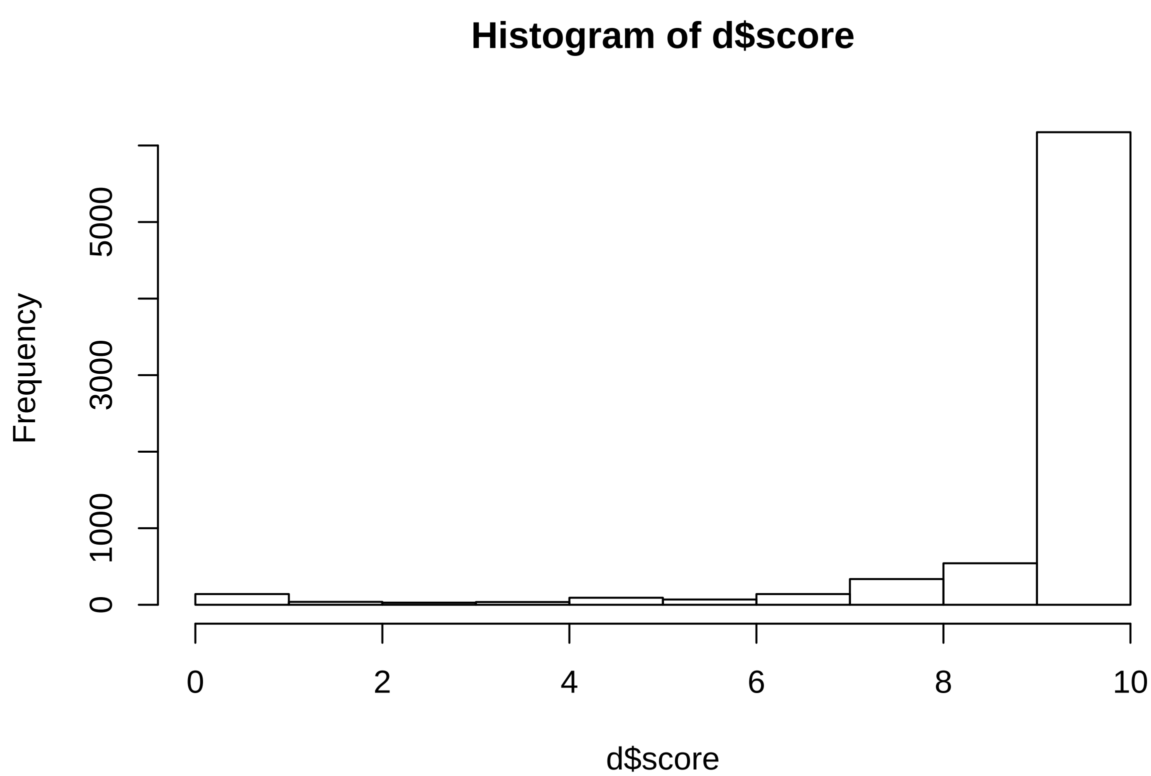
```r
# convert categorical values to factors
d$country = as.factor(d$country)
d$language = as.factor(d$language)
d$is_referred= as.factor(d$is_referred)
d$address_status = as.factor(d$address_status)
d$identity_status = as.factor(d$identity_status)
```

Examining the Score values distribution

```r
summary(d)
```

```
##      score            country      transactions   language is_referred
##  Min.   : 0.000   3      :6010   Min.   :  0.0   1:2443   0:6136
##  1st Qu.:10.000   4      : 481   1st Qu.:  5.0   2:2486   1:1459
##  Median :10.000   6      : 319   Median : 11.0   3:1559
##  Mean   : 9.417   7      : 265   Mean   : 18.8   4: 885
##  3rd Qu.:10.000   15     : 134   3rd Qu.: 24.0   5: 184
##  Max.   :10.000   (Other): 380   Max.   :411.0   6:  38
##                   NA's   :   6
##  first_booking        address_status identity_status   referrals
##  Min.   :       118   1:4940         1:   7          Min.   :   0.000
##  1st Qu.:       601   2:  89         2:  64          1st Qu.:   0.000
##  Median :      1648   3:2566         3:7524          Median :   0.000
##  Mean   :   2466367                                  Mean   :   1.466
##  3rd Qu.:    386294                                  3rd Qu.:   0.000
##  Max.   :108804750                                   Max.   :1833.000
##  NA's   :7
```

```r
hist (d$score)
```




```r
# Percentage of rows with score 10
sum (d$score == 10)/nrow(d)
```

```
## [1] 0.8129032
```

As we can see above, 81% of data rows have a score value of 10. **It means that a 'dumb' predictor which always predicts 10, would have precission of 81%.**

Examinining missing values

There are only a few missing values with no pattern:

```
md.pattern(d)
```

```
##      score transactions language is_referred address_status
## 7586     1            1        1           1              1
##    2     1            1        1           1              1
##    3     1            1        1           1              1
##    4     1            1        1           1              1
##           0            0        0           0              0
##      identity_status referrals country first_booking
## 7586               1         1       1             1  0
##    2               1         1       0             1  1
##    3               1         1       1             0  1
##    4               1         1       0             0  2
##                    0         0       6             7 13
```

```
# only 6 country and 7 first_booking values are missing. We will remove missing values
d <- d[complete.cases(d),]
```

Creating a model (xgboost)

```
# convert all categorical input variables to binary representation
sparse_matrix <- sparse.model.matrix(~., data = d[,2:9])

# split 20% test / 80% training
set.seed(2)
h=sample(c(0,1),prob=c(0.2,0.8),nrow(sparse_matrix),replace=T)
train=sparse_matrix[h==1,]
test=sparse_matrix[h==0,]

train_labels = d$score[h==1]
test_labels = d$score[h==0]

bst <- xgboost(data = train, label = train_labels, max.depth = 15,
               eta = 1, nthread = 1, nround = 30,objective = "multi:softmax", num_class = 11)
```

```
## [1]  train-merror:0.186888
## [2]  train-merror:0.220581
## [3]  train-merror:0.149544
## [4]  train-merror:0.128299
## [5]  train-merror:0.112531
## [6]  train-merror:0.096929
## [7]  train-merror:0.076515
## [8]  train-merror:0.059585
## [9]  train-merror:0.043154
## [10] train-merror:0.032199
## [11] train-merror:0.023402
```

```
## [12] train-merror:0.018257
## [13] train-merror:0.012282
## [14] train-merror:0.008963
## [15] train-merror:0.006639
## [16] train-merror:0.005975
## [17] train-merror:0.005809
## [18] train-merror:0.005643
## [19] train-merror:0.002988
## [20] train-merror:0.002158
## [21] train-merror:0.001162
## [22] train-merror:0.000996
## [23] train-merror:0.000830
## [24] train-merror:0.000996
## [25] train-merror:0.000830
## [26] train-merror:0.000498
## [27] train-merror:0.000498
## [28] train-merror:0.000498
## [29] train-merror:0.000332
## [30] train-merror:0.000332
```

```r
importance <- xgb.importance(feature_names = sparse_matrix@Dimnames[[2]], model = bst)
head(importance)
```

```
##            Feature       Gain      Cover  Frequency
## 1:   first_booking 0.52421874 0.56545646 0.56644918
## 2:    transactions 0.26065088 0.18212594 0.25068681
## 3:       referrals 0.03825412 0.04670622 0.03016255
## 4: address_status3 0.03424895 0.01218573 0.03548535
## 5:       language2 0.03205712 0.03150286 0.01654075
## 6:    is_referred1 0.02166324 0.01487639 0.02037546
```

It seems that two variables are much more important than others - seconds from account creation until **first_booking** and the **transacions** count.

Testing the model

```r
# predict
rez <- predict(bst, test)

# ROC value
auc(test_labels, rez)
```

```
## [1] 0.5194049
```

```r
# calculate prediction 'gap'
error = test_labels - rez

# percent of correct predictions (precision)
sum(error == 0 )/length(error)
```

```
## [1] 0.7847534
```

```r
# since the precission is not good, see if prediction are even close to the actual score:
#   treat prediction with error range from -2 to +2 as correct
sum( abs(error) < 3 )/length(error)
```

```
## [1] 0.9237668
```

Now compare precision of our xgboost model to a model which always predicts 10:

```
# all tens
all_tens <- rep(10, length(test_labels))

# ROC value
auc(test_labels, all_tens)
```

```
## [1] 0.5
```

```
# calculate 'gap' to all 10s
error2 = test_labels - all_tens

# percent of correct predictions (precision)
sum(error2 == 0 )/length(error2)
```

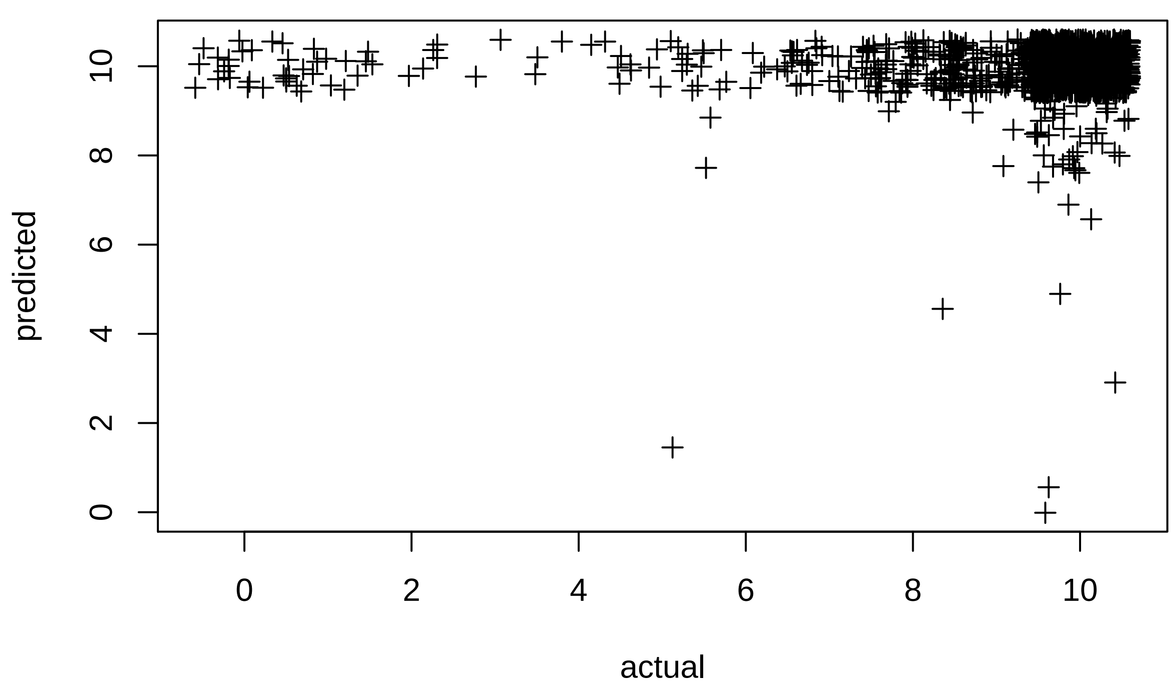
```
## [1] 0.8116592
```

```
# treat prediction with error range from -2 to +2 as correct
sum( abs(error2) < 3 )/length(error2)
```

```
## [1] 0.9282511
```

Visualize the xgboost prediction errors

```
# lets compare predicted
plot(jitter(rez,3) ~ jitter(test_labels, 3), pch = 3, xlab='actual', ylab='predicted')
```



As one can see the model mostly predicts 10s and fails to predict score for unseen data.