Today we will continue discussing Models Evaluation (Continuation from the last class). In particular, we concentrate on the retention problem. Business data analytics can help you identify who is about to churn by training classification model.

We will explore today a Telecommunication churn, and how we can detect it. Let’s proceed in our usual way:

```r
library(tidyverse)
library(data.table)
setwd("path of your file")
dt <- fread("edw_cdr.csv")
dt <- as.data.frame(dt)
dt <- dt[,c(6,1:5,7:ncol(dt))]  # it is more convinient to have customer id in the first column

Now, we assign proper types to features. This is how we can change multiple features to factors:

cols_to_factor <- c("customersuspended", "education", "gender", "homeowner", "churn", "maritalstatus", "occupation", "state", "usesinternetservice", "usesvoiceservice")
dt_cleaned <- dt %>%
  mutate_at(cols_to_factor, funs(factor(.)))
dt_cleaned <- dt_cleaned[!duplicated(dt_cleaned),]
#summary(dt_cleaned)

Next, let’s split the dataset on a training and a test set (50/50).

```r
set.seed(4698)
train_idx <- <YOUR CODE HERE>
train <- <YOUR CODE HERE>
test <- <YOUR CODE HERE>
str(dt_cleaned)
```

Solution:

```r
set.seed(123)
train_idx <- sample(c(TRUE, FALSE), nrow(dt_cleaned), replace = T, prob = c(0.6,0.4))
train <- dt_cleaned[train_idx , ]
test <- dt_cleaned[!train_idx , ]
```
Let's first fit a very simple logistic regression:

```r
logistic_simple <- glm(data = train[, -1], formula = as.factor(churn) ~ calldroprate + gender,
                         family = 'binomial')
summary(logistic_simple)
```

```
##
## Call:  
glm(formula = as.factor(churn) ~ calldroprate + gender, family = "binomial",  
data = train[, -1])
##
## Deviance Residuals:  
##    Min     1Q Median     3Q    Max  
## -0.4682 -0.4547 -0.4463 -0.4365  2.2093  
##
## Coefficients:  
##             Estimate Std. Error z value Pr(>|z|)  
## (Intercept) -2.26361  0.06749   -33.54   <2e-16 ***  
## calldroprate 1.54704  1.46907    1.053   0.292  
## genderMale  -0.08579  0.06782   -1.265   0.206  
## ---  
## Signif. codes:  
##     0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1  
##
## (Dispersion parameter for binomial family taken to be 1)
##
## Null deviance: 6442.8 on 10232 degrees of freedom  
## Residual deviance: 6440.4 on 10230 degrees of freedom  
## AIC: 6446.4  
##
## Number of Fisher Scoring iterations: 5
```

```
exp(logistic_simple$coefficients)
```

```
## (Intercept) calldroprate genderMale  
## 0.1039742  4.6975240  0.9177900
```

Let's fit it with all available data:

```r
logistic <- glm(data = train[, -1], formula = as.factor(churn)~., family = 'binomial')

# we predict the churn porbability for a new test set and assing them to the test  
test$logistic_predictions <- predict(newdata=test[, -1], logistic, type='response')
```
```r
code
head(test[,c(1, ncol(test)-1, ncol(test))])

## customerid churn logistic_predictions
## 1     1     0    0.11023530
## 2     1     0    0.11876528
## 3     2     0    0.07686952
## 4     2     0    0.07719955
## 5     3     0    0.03465510
## 6     3     0    0.03433982
```

We have a scores on a continuous scale from 0 to 1.

```r
code
library(ggplot2)
ggplot(data = test, aes(x=logistic_predictions, fill=churn)) + geom_density(alpha=0.3) + theme_bw()
```
Now, we need to decide on a threshold. If the dataset is **balanced** (number of positive and negative cases are approximately equal), 0.5 is a good shot.

```r
test <- mutate(test, logistic_predictions_binary = ifelse(logistic_predictions >= 0.5, 1, 0))
table(real = test$churn, predicted = test$logistic_predictions_binary)
```

**Task 5 (2 p).** Is our dataset balanced? How do you check it?

Let’s calculate precision \(\frac{tp}{tp+fp}\) and recall \(\frac{tp}{tp+fn}\)

```r
precision_recall = function(real, predicted){
  tbl = as.data.frame(table(real=real, predicted=predicted))
  tp = filter(tbl, real==1 & predicted==1)$Freq
  fp = filter(tbl, real==0 & predicted==1)$Freq
  fn = filter(tbl, real==1 & predicted==0)$Freq
  precision = tp/(tp+fp)
  recall = tp/(tp+fn)
  return(c(precision, recall))
}
precision_recall(test$churn, test$logistic_predictions_binary)
## [1] 0.1334207 0.5738444
```

**Task 6 (1 p).** Interpret these numbers in a context of our particular case of telecom churn.

Now, if we want to find an optimal threshold (and plot ROC with AUC), we could use the following commands:

```r
# Get threshold
library(pROC)
roc_curve <- roc(test$churn, test$logistic_predictions_binary, plot = T)
```
bestthr <- coords(roc_curve, "best", ret = "threshold")
bestthr
## [1] 0.0851389
Let's compare it to more powerful:

**Random Forest**

```r
library(randomForest)
rf_m <- randomForest(data = train, as.factor(churn)~., importance = T, do.trace = F)
test$rf_predictions <- predict(newdata=test, rf_m, type = 'prob')[,2]
# the prediction output is different from Logistic regression.
# It gives a matrix, where we take the second column
```

Task 7 (2 p). Fill the blanks and compare it to the logistic model. Which one is better?

```r
test$rf_predictions_binary <- ifelse(rf_predictions >= YOUR CODE HERE, 1, 0)
table(real = YOUR CODE HERE, predicted = YOUR CODE HERE)
#confusionMatrix(data=test$rf_predictions_binary, reference=test$churn)
precision_recall(YOUR CODE HERE, YOUR CODE HERE)
ggplot(YOUR CODE HERE)
```

**Balancing data**

If the data is very imbalanced, the algorithm might predict everything according to the majority class. If this happens, one should balance data. Basic methods are **undersampling** and **oversampling**. By using undersampling we compose the training set with all available minor class and add the same amount of major class. The ratio of the test set should remain the same as in the initial data (otherwise your results will not reflect the real situation). In oversampling, you do the opposite. You pick in the training set the majority class available and duplicate minor class instances until it reaches approximately 50/50 ratio. Here we will try undersampling.
# balancing - undersampling

```r
set.seed(258)
churn <- filter(dt_cleaned, churn==1) # separate set for churn
regular <- filter(dt_cleaned, churn==0) # separate for regular

train_idx_churn <- sample(nrow(churn), nrow(churn)/2, replace = F)
# half of churn goes to train
train_churn <- churn[train_idx_churn,] # half churn for train
test_churn <- churn[-train_idx_churn,] # and half for test

train_idx_regular <- sample(nrow(regular), nrow(train_churn), replace = F)
# we take as much regular as there will be churn in the train
train_regular <- regular[train_idx_regular,]

train_balanced <- rbind.data.frame(train_churn,train_regular) # combine

# check the correctness of the results

table(dt_cleaned$churn)
```

```r
##
##    0   1
## 18604 1863
```

```
1863/18604
```

```
## [1] 0.1001398
```

```r

table(train_balanced$churn)
```

```r
##
##    0   1
##   931  931
```
table(test_original_ratio$churn)
##
## 0 1
## 9320 932

Decision trees

Let’s briefly investigate, how to fit the single decision tree on a subset of data:

```r
library(rpart)
tree <- rpart(data=train_balanced[, -1], churn ~ ., method='class')
#summary(tree)
library(rpart.plot)
rpart.plot(tree)
```

References:

1) [http://uc-r.github.io/logistic_regression](http://uc-r.github.io/logistic_regression)
3) [https://stats.idre.ucla.edu/r/dae/logit-regression/](https://stats.idre.ucla.edu/r/dae/logit-regression/)
7) [https://rstudio-pubs-static.s3.amazonaws.com/123908_3df6d25c8ad04030a2a429661125a59b.html#2](https://rstudio-pubs-static.s3.amazonaws.com/123908_3df6d25c8ad04030a2a429661125a59b.html#2) decision tree visualization