Churn Management

Giovanni Compiani

Contents

```
library(bit64)
library(data.table)
library(ggplot2)
library(broom)
library(knitr)
```

1 Overview

Cell2Cell, a wireless telecommunications company (with its name altered for confidentiality), is working to reduce customer churn. The objective of this project is to create a model that predicts customer churn at Cell2Cell and to leverage insights from the model to design a targeted incentive strategy aimed at decreasing churn rates.

In the assignment, you will address these key issues:

1. Is customer churn at Cell2Cell predictable from the customer information that Cell2Cell maintains?
2. What factors drive customer churn? Which factors are particularly important?
3. What incentives should Cell2Cell offer to prevent customer churn?
4. What is the economic value of a proposed targeted plan to prevent churn, and how does this value differ across customer segments? Compare the economic value to an incentive with a cost of $100 and another incentive with a cost of $175. Which customers segments should receive the incentive? Does the answer depend on the success probability?

Note that, in what follows, the key steps you need to take are highlighted in *italic*.

2 Data

All data are contained in the file `Cell2Cell.RData`, which is posted on Canvas.

```
load("C:/Users/gcompiani/Dropbox/Teaching/37105/Assignments/04 Churn-Management/Data/Cell2Cell.RData")
```

Please consult the file `Cell2Cell-Database-Documentation.xlsx` for a description of the data and some summary statistics. Note that *calibration sample* is an alternative term for *training* or *estimation* sample.

Report the churn rate in the calibration sample and in the validation sample and compare the two.

You can see that the calibration sample was selected using *oversampling*. The purpose of oversampling was to obtain more precise estimates (lower standard errors) when estimating a logistic regression model. The validation sample, on the other hand, was not created using oversampling and represents the *true churn rate* in the data.

As you can see, some variables have missing values, which—as you know by now—is common and of no concern (unless the missing values indicate some *systematic* flaws or bias in how the data were constructed). Most estimation methods in R will automatically delete rows with missing values before estimating the model. However, the `predict` methods will yield `NA` values if a row in the data used for prediction contains missing values. Hence, in a situation where you don't need to keep the full data I recommend to remove any observations with missing values before you conduct the main analysis.

Perform this data-cleaning step.

3 Model estimation

Estimate a logit model to predict the conditional churn probability.

You can inspect the regression output using methods you already used, such as `summary`. Having said this, especially when you have a large number of inputs, it can be convenient to store the regression estimates in a table. A simple way to do this is to install the broom package that has the purpose of cleaning up messy R output.

Using the `tidy` function in the `broom` package it is trivial to capture the regression output in the form of a data.table:

```
results_DT = as.data.table(tidy(fit))
kable(results_DT, digits = 5)
```

For `kable` to work, you need to load the `knitr` library.

4 Prediction: Accounting for oversampling

The idea of oversampling is as follows. If the response rate in the data is small, there is a strong imbalance between observations with a response of $Y = 1$ and a response of $Y = 0$. As a consequence, estimating the model is difficult and the estimates will be imprecise, i.e. they will have large standard errors.

The solution: Create a training sample with one half of observations randomly chosen from the original data with response $Y = 1$, and the other half randomly chosen from the original data with response $Y = 0$. Now estimation is easier and the standard errors will be smaller.

However, when applied to logistic regression, oversampling will result in an inconsistent estimate of the intercept (constant) term, although all other estimates will be consistent. Hence, if we do not de-bias (adjust) the intercept, the predicted probabilities will be too large, reflecting the artificial response rate of $\frac{1}{2}$ in the over-sampled training data.

In order to de-bias the scale of the predicted response (in this example: churn) in the validation sample we need to supply an *offset variable* to the logistic regression model. An offset is a known number that is added to the right-hand side of the regression when estimating the model, and adding the offset will correspondingly change the estimate of the intercept. The offset takes the form:

$$\text{offset} = [\log(\bar{p}_t) - \log(1 - \bar{p}_t)] - [\log(\bar{p}_v) - \log(1 - \bar{p}_v)]$$

Here, $\bar{p}_t$ is the average response rate in the training sample and $\bar{p}_v$ is the average response rate in the validation sample. Note that the offset is positive (given that $\bar{p}_t > \bar{p}_v$), so that including the offset term when estimating the model accounts for the fact that the training sample has a higher share of $Y = 1$ relative to the validation sample.

Conversely, when we predict the response rate in the validation sample, we set the offset variable to 0.

Why does this work? — Conceptually, logistic regression is a regression model for the log-odds of the response (outcome) probability,

$$\log\left(\frac{p}{1-p}\right) = \log(p) - \log(1-p) = \beta_0 + \beta_1 X_1 + \beta_2 X_1 + \dots$$

When we add the offset variable to the right hand side of the regression model the estimation algorithm will "incorporate" the offset in the intercept, β_0. The effect of setting the offset to 0 (when applying the model to the validation sample) is equivalent to subtracting the offset from the intercept. Subtracting the offset amounts to:

(i) Subtracting $\log(\bar{p}_t) - \log(1 - \bar{p}_t)$, the log-odds of the artificial response rate in the training sample, and

(ii) Adding $\log(\bar{p}_v) - \log(1 - \bar{p}_v)$, the log-odds in the validation sample that reflects the true log-odds in the data.

This process de-biases the predicted response, i.e. restores the correct response level in the validation sample.

Note: Never use over-sampling to create the validation sample, otherwise the offset variable approach will not work.

Create an `offset_var` *variable and add it to the data set. Then re-estimate the logistic regression. To tell* `glm` *that you want to use* `offset_var`*, you need to use a formula of the form:*

```
y ~ offset(offset_var) + <all other variables>
```

Where you place `offset()` on the right-hand side of the formula is irrelevant.

Before predicting the response rate in the validation sample set the offset to 0. Then, when you invoke the `predict` *function, supply the data with the offset set to 0 using the* `newdata` *option.*

Compare the average predicted response with the average observed response rate in the validation sample.

5 Predictive power: Lift

We evaluate the predictive fit of the logistic regression model using a lift table and lift chart. To develop reusable code, we develop a function that returns a lift table. The function (call it `liftTable`) will need to take the following inputs:

- Predicted outcome or score
- Observed outcome
- Number of segments to be created based on the score

`liftTable` will return a data.table that contains:

- An index (`score_group`) for each segment that was created based on the score
- The average score value (predicted outcome) in the `score_group`
- The average observed outcome in the `score_group`
- The lift factor

To code the `liftTable` command, I recommend to use the `cut_number` function in the ggplot2 package. `cut_number` takes a variable `x` and creates `n` groups with an approximately equal number of observations in each group. Observations are assigned to the groups based on their ranking along the variable `x`. The format is:

```
cut_number(x, n = <no. of groups>)
```

To illustrate, we draw 10,000 random numbers from a uniform distribution on $[0, 5]$. `cut_number` assigns each number to one of five (because we set `n = 5`) groups.

```
set.seed(123)
DT = data.table(x = runif(10000, min = 0, max = 5))
DT[, group    := cut_number(x, n = 5)]
DT[, group_no := as.integer(group)]
```

```
head(DT)
```

```
          x        group group_no
      <num>       <fctr>    <int>
1: 1.4378876     (1,2.01]        2
2: 3.9415257  (2.98,3.98]        4
3: 2.0448846  (2.01,2.98]        3
4: 4.4150870     (3.98,5]        5
5: 4.7023364     (3.98,5]        5
6: 0.2277825 [0.000327,1]        1
```

```
table(DT$group)
```

```
[0.000327,1]     (1,2.01]  (2.01,2.98]  (2.98,3.98]     (3.98,5]
        2000         2000         2000         2000         2000
```

As expected, because `x` is uniformly distributed on $[0, 5]$, the five groups created by `cut_number` correspond almost exactly to a $[k, k+1]$ interval ($k = 0, 1, \ldots, 4$), and each of these intervals contains exactly 20 percent of all observations based on the rank of the `x` values. The `group` variable that we created is a factor that we converted to an integer score.

Calculate a lift table for 20 segments. Inspect the lift table. Then provide two charts. First, plot the ***`score_group`*** *segments on the x-axis versus the observed churn rate on the y-axis. Second, plot the segments versus the lift factor, and add a horizontal line at $y = 100$. How to do this in ggplot2 is explained in the ggplot2 guide (look for the* ***`yintercept`*** *option).*

6 Why do customers churn? — Effect sizes

We would like to understand *why* customers churn, which can help us to propose incentives to prevent customer churn

To this end, construct a table that contains comparable effect sizes (changes in the churn probability) for all independent variables, as we discussed in class.

Here are a few more details on the steps needed to create this table:

1. Because logistic regression coefficients are not directly interpretable, we estimate a linear probability model of customer churn. In a linear probability model we regress the $Y = 0, 1$ output on all the customer features. The estimated coefficients can be interpreted as differences in $\Pr\{Y = 1|X_1, X_2, \dots\}$ for a one-unit difference in one of the features, X_k. Note: **The *offset variable* should not be included in the linear probability model as it is specific to logistic regression.**
2. Note that our analysis is based on a *comparison* of the effect sizes of the different variables. However, because the variables have different scales, the effect sizes are not directly comparable. For example, `revenue` (mean monthly revenue) and `mou` (mean monthly minutes use) have different means and standard deviations, and hence the effects of increasing `revenue` and `mou` by one unit on the churn probabilities are not comparable without taking the scale differences into account.
3. To solve this problem we **standardize** the independent variables in the data. To standardize, we divide the values of each independent variable by its standard deviation, except if the variable is a 0/1 dummy. Once standardized, all variables except the dummies will have a standard deviation of 1, and a one unit difference corresponds to a one standard deviation difference in the original, non-standardized variable. Here's a function, `standardize_columns`, that takes a column `x` as input and returns the standardized values of the column:

```
standardize_columns <- function(x) {

   # Check if the column is a dummy variable
   elements = unique(x)
   if (length(elements) == 2L & all(elements %in% c(0L,1L))) {
      is_dummy = TRUE
   } else {
      is_dummy = FALSE
   }

   # If not a dummy, divide values in x by its standard deviation
   if (is_dummy == FALSE) x = x/sd(x, na.rm = TRUE)

   return(x)
}
```

The first part of the function checks that the input `x` has exactly two elements and that these elements are the integers 0 and 1. Note that in R, numbers are represented as floating point numbers by default. However, adding `L` after the numbers tells R to represent the number as an integer.

```
class(1)
```

```
[1] "numeric"
```

```
class(1L)
```

```
[1] "integer"
```

In order to standardize all independent variables in the training data, you can use:

```
DT_lin_prob = cell2cell_DT[calibrat == 1]
```

```r
# Create a vector that contains the names of all inputs (covariates)
all_columns   = names(DT_lin_prob)
input_columns = all_columns[-c(1:3, length(all_columns))]

# Standardize all input columns
DT_lin_prob[, (input_columns) := lapply(.SD, standardize_columns), .SDcols = input_columns]
```

4. In order to create a table that captures the linear probability model estimates, use the `tidy` function. Add a column, e.g. `effect_size`, that scales the estimates by the factor

$$100 \cdot \frac{\bar{p}_v}{\bar{p}_t}$$

   This scales the effect sizes to the correct magnitude of the churn probabilities in the validation sample and puts the effects on a 0-100% scale. Sort the variables according to the magnitude of the effect sizes, and print the results table using `kable`.
5. Inspect the results. Identify some factors that are strongly associated with churn. If actionable, propose an incentive that can be targeted to the customers to prevent churn.

7 Economics of churn management

Next, we would like to predict the value of a proposed churn management program in order to assess the maximum amount that we would spend to prevent a customer from churning for one year.

Perform this prediction, under the following assumptions:

1. We consider a planning horizon of 4 years (the current year and three additional years), and an annual discount rate of 8 percent.
2. Predict the churn management value for 20 groups, but keep in mind that it is good practice to make sure the code works for an arbitrary number of groups in case we wish to modify that in the future. Predict the program value for these 20 customer segments based on the predicted churn rate. Note that we create these segments based on the validation sample data. We predict current and future customer profits at the year-level. Hence, we also need to convert both the monthly churn rate and the revenue data to the year-level.
3. Assume that the churn management program has a success probability `gamma` (γ) and compare the results for $\gamma = 0.25$ and $\gamma = 0.5$.

Hint: It is easy to make little mistakes in the lifetime value predictions. Hence, be very clear about what your code is supposed to achieve, and check that every step is correct.

8 Summarize your main results

Please organize your main results along the four questions posed in the overview.