Assignment 6: Churn Management

Günter J. Hitsch

May 22, 2020

Contents

1 Overview

Cell2Cell is a wireless telecom company (the name was changed to disguise the exact identity). As part of its various CRM programs, Cell2Cell actively mitigates customer churn. For this purpose, we develop a model to predict customer churn at Cell2Cell and use the insights from the model to develop a targeted incentive plan to lower the churn rate.

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?

```
library(ggplot2)
library(tidyverse)
library(dplyr)
library(tidyr)
library(lfe)
load('Cell2Cell.RData')
```

2 Data

All data are contained in the file `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. Hence, the variable `calibrat` indicates if a customer belongs to the training (estimation) sample or to the validation sample.

Inspect the data.

Some variables have missing values, which 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.

Recall from the *Introduction to Data Science in R* guide that you can drop all rows in a data frame that contain missing values using the `na.omit()` function.

Oversampling

One important aspect of the data that is that the calibration sample was selected using *oversampling*. Oversampling is a technique to choose a calibration sample such that the values of the outcome (dependent) variable are balanced. In particular, the outcome in one half of the observations takes the value 0 (customer did not churn), and the other half takes the value 1 (the customer churned).

The purpose of oversampling is to obtain more precise estimates (lower standard errors) when estimating a logistic regression model. However, oversampling also results in biased predictions, in our example predicted churn probabilities that are larger than the true churn probabilities. How to correct for this bias is explained further below.

The validation sample, on the other hand, *was not created using oversampling* and represents the *true churn rate* in the data.

Verify that approximately 50% of observations in the training sample take the value 1, and contrast this number with the *true churn rate*, i.e. the average rate or probability at which customers churn, in the validation sample.

```
library(tidyverse)

estimation_df = na.omit(cell2cell_df[cell2cell_df$calibrat == 1, ])

validation_df = na.omit(cell2cell_df[cell2cell_df$calibrat == 0, ])

pv = mean(validation_df$churn)

pe = mean(estimation_df$churn)

print(pv)
```

```
[1] 0.01929663
```

```
print(pe)
```

```
[1] 0.4981382
```

3 Model estimation

Estimate a logit model to predict the conditional (on the customer attributes) churn probability.

- Make sure that you estimate the model using the calibration = estimation/training sample only, where `calibrat` takes the value 1. One straightforward way of achieving this is to create two new data frames, `estimation_df` and `validation_df`, from the original data.
- Obviously, `customer` and `calibrat` should not be included as independent variables in the logistic regression.

You can display the regression output using `summary`. However, especially when you have a large number of independent variables in the model, it is often convenient to store the regression estimates in a table (data frame).

A simple way to do this is to install the `broom` and `kable` packages. The `tidy` function in the `broom` package converts the regression output (here, the object `fit`) into a data frame. Furthermore, the `kable` function creates a nicely formatted table.

```
library(broom)
library(knitr)




fit_logistic = estimation_df[,c(-1, -2)] %>%
                glm(churn ~ ., data = ., family = "binomial")

estimation_results_df = as.data.frame(tidy(fit_logistic))
kable(estimation_results_df, digits = 5,
      col.names = c("variable", "estimate", "se", "z-statistic", "p-value"))
```

variable	estimate	se	z-statistic	p-value
(Intercept)	0.14970	0.09527	1.57127	0.11612
revenue	0.00196	0.00080	2.46016	0.01389
mou	-0.00028	0.00005	-5.65709	0.00000
recchrge	-0.00312	0.00089	-3.51332	0.00044
directas	-0.00120	0.00594	-0.20135	0.84042
overage	0.00076	0.00028	2.71118	0.00670
roam	0.00709	0.00206	3.43641	0.00059
changem	-0.00049	0.00005	-9.19411	0.00000
changer	0.00230	0.00037	6.24656	0.00000
dropvce	0.01134	0.00725	1.56304	0.11804
blckvce	0.00640	0.00716	0.89452	0.37104
unansvce	0.00092	0.00045	2.05780	0.03961
custcare	-0.00595	0.00255	-2.33129	0.01974
threeway	-0.03029	0.01125	-2.69107	0.00712
mourec	0.00013	0.00013	1.01771	0.30882
outcalls	0.00112	0.00059	1.89446	0.05816
incalls	-0.00311	0.00106	-2.93705	0.00331
peakvce	-0.00067	0.00022	-3.05791	0.00223
opeakvce	-0.00021	0.00027	-0.78284	0.43372
dropblk	-0.00311	0.00704	-0.44249	0.65814
callfwdv	-0.00264	0.02315	-0.11414	0.90913
callwait	0.00208	0.00314	0.66379	0.50682

variable	estimate	se	z-statistic	p-value
months	-0.02128	0.00200	-10.65184	0.00000
uniqsubs	0.18438	0.01999	9.22522	0.00000
actvsubs	-0.20572	0.02791	-7.37167	0.00000
phones	0.04866	0.01817	2.67837	0.00740
models	0.01380	0.02787	0.49501	0.62060
eqpdays	0.00144	0.00007	19.30918	0.00000
age1	-0.00330	0.00087	-3.78708	0.00015
age2	-0.00117	0.00068	-1.71810	0.08578
children	0.09455	0.02815	3.35918	0.00078
credita	-0.17807	0.03550	-5.01584	0.00000
creditaa	-0.36265	0.03458	-10.48750	0.00000
prizmrur	0.06649	0.04956	1.34154	0.17975
prizmub	-0.03963	0.02441	-1.62389	0.10440
prizmtwn	0.04622	0.03145	1.46985	0.14160
refurb	0.23404	0.03196	7.32343	0.00000
webcap	-0.15613	0.03756	-4.15660	0.00003
truck	0.02689	0.03600	0.74698	0.45508
rv	0.01186	0.04801	0.24700	0.80491
occprof	-0.01987	0.03250	-0.61131	0.54100
occcler	0.03949	0.07491	0.52720	0.59805
occcrft	-0.02013	0.06290	-0.32009	0.74890
occstud	0.11997	0.12187	0.98441	0.32492
occhmkr	0.25587	0.19008	1.34611	0.17827
occret	-0.03993	0.09055	-0.44096	0.65924
occself	-0.07057	0.08059	-0.87566	0.38122
ownrent	0.00255	0.04272	0.05978	0.95233
marryun	0.10881	0.03403	3.19771	0.00139
marryyes	0.05570	0.03249	1.71446	0.08644
mailord	0.00077	0.08565	0.00897	0.99284
mailres	-0.12971	0.08604	-1.50754	0.13167
mailflag	-0.04818	0.08445	-0.57055	0.56830
travel	-0.00053	0.04732	-0.01124	0.99103
pcown	0.03418	0.03096	1.10391	0.26963
creditcd	0.04202	0.04371	0.96123	0.33644
retcalls	0.01203	0.18367	0.06552	0.94776
retaccpt	-0.12786	0.10764	-1.18781	0.23491
newcelly	-0.07053	0.02727	-2.58607	0.00971
newcelln	-0.00508	0.03153	-0.16127	0.87188
refer	-0.05003	0.04214	-1.18703	0.23521
incmiss	-0.09151	0.06006	-1.52357	0.12761
income	-0.01324	0.00603	-2.19481	0.02818
mcycle	0.12231	0.08898	1.37445	0.16930
setprcm	-0.09632	0.04051	-2.37748	0.01743
setprc	0.00062	0.00028	2.19418	0.02822
retcall	0.79370	0.19458	4.07899	0.00005

```
validation_predict_df = predict(fit_logistic, validation_df[,c(-1, -2)], type = "response")

mean(validation_predict_df)
```

```
[1] 0.4772273
```

```
pv = mean(validation_df$churn)

pe = mean(estimation_df$churn)
```

Use the help function, `?kable`, to learn more about the options.

Skimming the estimation results, is there evidence that the independent variables may be able to predict differences in churn rates across customers?

Next, predict (`predict`) the churn rates (probabilities) in the validation sample.

Calculate the mean of the predicted churn rates and verify that this mean reflects the average observed churn rate in the estimation sample, not the mean of the observed churn in the validation sample.

Recall that the validation sample was chosen without oversampling and indicates the *true churn rate* in the data. Hence, we need to account for the upward bias in the predicted churn probabilities due to oversampling in the estimation sample. How to do this is explained in the next section.

4 Model estimation and prediction: Accounting for oversampling

4.1 Explanation

Note: Fully understanding this explanation is not very important, and you can simply skim the section.

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 a biased estimate of the intercept (constant) term, although all other estimates will be unbiased.[1] 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:

$$\text{offset} = (\log(\bar{p}_e) - \log(1 - \bar{p}_e)) - (\log(\bar{p}_v) - \log(1 - \bar{p}_v))$$

Here, $\bar{p}_e$ is the average response rate in the estimation sample and $\bar{p}_v$ is the average response rate in the validation sample.

Then, 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 is equivalent to subtracting the offset from the intercept. Subtracting the offset amounts to:

(i) Subtracting $\log(\bar{p}_e) - \log(1 - \bar{p}_e)$, the log-odds of the artificial response rate in the estimation 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.

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

```
offset = (log(pe) - log(1-pe)) - (log(pv) - log(1-pv))
```

[1] To be technically correct, the other estimates will be *inconsistent*.

4.2 Implementation

(1) To create the offset variable, `offset_var`, compute the mean observed churn rates in the estimation and training samples, $\bar{p}_e$ and $\bar{p}_v$.

Then calculate the offset using the formula

$$\texttt{offset\_var} = (\log(\bar{p}_e) - \log(1 - \bar{p}_e)) - (\log(\bar{p}_v) - \log(1 - \bar{p}_v))$$

```
offset_var = (log(pe) - log(1-pe)) - (log(pv) - log(1-pv))
```

(2) Add the offset as a new column to the estimation sample. Then estimate the logistic regression model.

To tell `glm` that you want to use `offset_var` you need to use a formula of the form:

```
estimation_df$offset_var = offset_var

fit_logistic = estimation_df[,c(-1, -2)] %>%
            glm(churn ~ . + offset(offset_var), data = ., family = "binomial")
```

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

For example, if you use all variables in your data (except `y` and `offset_var`) as independent variables:

```
glm(y ~ . + offset(offset_var), ...)
```

(3) Set the offset variable `offset_var` to 0 and add it to the validation sample. This is simple: In the validation sample, create a column that has the same name as the offset variable column in the estimation sample, and make sure this column takes values `= 0`.

Then predict the churn rates in the validation sample.

Note: When predicting the churn rates you will likely see the warning message, *prediction from a rank-deficient fit may be misleading*. Simply ignore. To suppress the warning message, add `warning = FALSE` to the code chunk header, i.e. `{r, warning = FALSE}`.

Calculate the mean of the predicted churn rates and verify that this mean reflects the observed average churn rate in the validation sample.

```
validation_df$offset_var = 0

validation_predict_df = predict(fit_logistic, validation_df[,c(-1, -2)], type = "response")

c(churn_rate = pv, churn_rate_pred = mean(validation_predict_df), error_rate = (mean(validation_predict_
```

```
 churn_rate churn_rate_pred      error_rate
0.019296628     0.019432437     0.007037954
```

5 Model validation

Provide evidence for the validity of the model using a lift table and a lift chart.

Note: All model validation steps are need to be performed in the validation sample.

(1) Create a table that contains both the observed `churn` and the predicted churn rate (probability) from the logistic regression model that you estimated in the previous section. You can simply add the predicted churn rate as a new column to the validation sample data frame.

```
compare = validation_df[, "churn"]

compare = cbind(compare, validation_predict_df)

compare =data.frame(compare)

names(compare)[1] <- "churn"
names(compare)[2] <- "predict"
```

(2) Add a column, `score`, that assigns a number from 1 to 10 to each customer based on the predicted churn rate. Use the `cut_number` function that is part of the `tidyverse` (to be precise, the function is in the `ggplot2` package):

```
score = cut_number(Pr_churn, n = 10, labels = FALSE)
```

`cut_number` creates `n` equally sized groups, where group 1 represents customers with the lowest predicted churn rates and `n` (here: `n = 10`) represents customers with the highest predicted churn rates.

```
compare$score = cut_number(compare$predict, n = 10, labels = FALSE)

head(compare)
```

```
  churn     predict score
1     0 0.006351394     1
2     0 0.006508693     1
3     0 0.009064107     1
4     0 0.007762943     1
5     0 0.011014744     2
6     0 0.016396780     5
```

(3) Create a summary table that, for each of the scores, contains:

(i) Mean predicted churn rate
(ii) Mean observed churn rate
(iii) Lift factor

Recall the definition of the lift:

$$\text{lift in segment } k = 100 \cdot \frac{\text{churn rate in segment } k}{\text{average churn rate across all customers}}$$

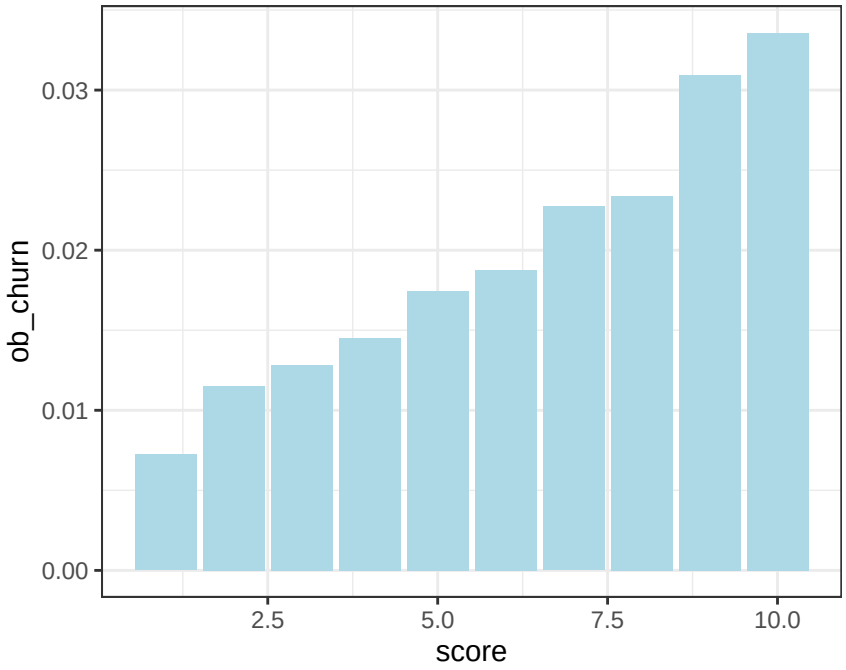
Hint: Create the summary table using `group_by` and `summarize`.

```
compare_score = compare %>%
  group_by(score) %>%
  summarize(ob_churn = mean(churn),
            pr_churn = mean(predict),
            lift = 100*ob_churn/pv)
```

(4) Provide a graph that displays score-level mean observed churn on the y-axis and the customer `score`

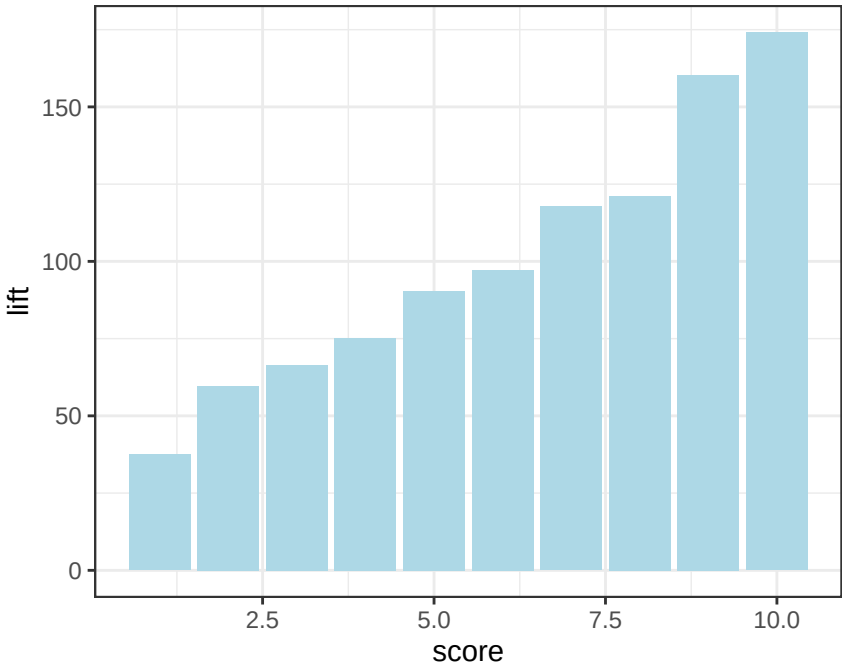
on the x-axis.

```
library("ggplot2")

compare_score %>%
ggplot(aes(x = score, y = ob_churn), data = .) + geom_col(fill = "lightblue") + theme_bw()
```



(5) Similarly, provide a lift chart (graph).

```
compare_score %>%
ggplot(aes(x = score, y = lift), data = .) + geom_col(fill = "lightblue") + theme_bw()
```



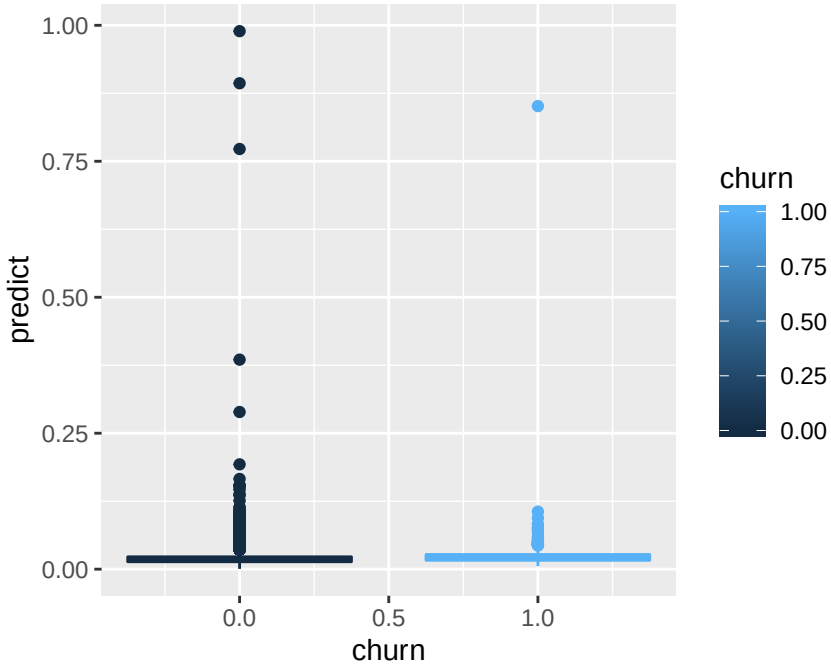
(6) Display the data in (4) and (5) in the form of a table.

```
compare_score[-3]
```

```
# A tibble: 10 x 3
   score ob_churn  lift
   <int>    <dbl> <dbl>
 1     1  0.00724  37.5
 2     2  0.0115   59.7
 3     3  0.0128   66.5
 4     4  0.0145   75.1
 5     5  0.0175   90.4
 6     6  0.0188   97.3
 7     7  0.0227  118.
 8     8  0.0234  121.
 9     9  0.0310  160.
10    10  0.0336  174.
```

Discuss the results. Do the results provide evidence for the validity of the model?

```
compare %>%
ggplot(aes(x = churn, y = predict, group = churn, color = churn), data = .) + geom_boxplot( )
```



6 Effect sizes: Why do customers churn?

We would like to understand *why* customers churn, which can help us to propose incentives to prevent customer churn. Hence, we now construct a table that contains comparable effect sizes (changes in the churn probability) for all independent variables, as we discussed in class.

Because logistic regression coefficients are not directly interpretable, we first need to estimate a linear probability model that predicts customer churn.

Note: Do *not* use an offset variable when estimating the linear probability model!

Once you have estimated the model, you can construct the effect size table in Excel, or you can use the code below to construct the table in an automated fashion

6.1 Construct effect size table in Excel

Assuming your linear probability model output is called `fit_linear_prob`, you can convert the estimation results into a data frame and then save the estimation results as a csv file:

```
fit_linear_prob = estimation_df[,c(-1, -2, -70)] %>%
                  lm(churn ~ ., data = .)

linear_prob_results_df = as.data.frame(tidy(fit_linear_prob))
write_csv(linear_prob_results_df, path = "Assign6_export.csv")
```

Now proceed in Excel, using both the `Cell2Cell-Database-Documentation.xlsx` file and the estimation results files.

6.2 Automated approach to construct effect size table

Run all four code chunks below.

For the code to work ensure that:

(i) Your main data frame, `cell2cell_df`, is in memory, and you have removed all rows with missing values.

```
cell2cell_df = na.omit(cell2cell_df)
```

(ii) Either the estimation results object from the linear probability model is called `fit_linear_prob`, or you modify the object name in the third code chunk below.

(iii) The objects `p_e` and `p_v` that contain the mean observed churn rates in the estimation and validation samples are in memory.

```
p_e = pe
p_v = pv
```

If these conditions are met, the code below will work.

First, we create the function `isDummy` that returns the value 1 if `x` is a dummy variable, and 0 otherwise.

```
isDummy <- function(x) {
   elements = unique(x)
   if (length(elements) == 2L & all(elements %in% c(0L,1L))) is_dummy = TRUE
   else is_dummy = FALSE
   return(is_dummy)
}
```

Second, we create `summary_df`, which contains the standard deviation of all variables and an indicator, `is_dummy = 1`, if the variable is a dummy variable.

```
summary_df = cell2cell_df %>%
   summarize_all(list(sd = sd, is.dummy = isDummy)) %>%
   pivot_longer(cols = everything(),
                names_to = "statistic", values_to = "value") %>%
   separate(statistic, into = c("variable", "statistic"), sep = "_") %>%
   pivot_wider(id_cols = variable, names_from = "statistic", values_from = "value") %>%
   rename(is_dummy = is.dummy)
```

Third, we create the `effect_sizes_df` table that combines the estimation results (in the object `fit_linear_prob`) and the summary statistics.

```
effect_sizes_df = as.data.frame(tidy(fit_linear_prob)) %>%
   transmute(variable = term, estimate = estimate, t_stat = statistic) %>%
   inner_join(summary_df) %>%
   mutate(change_prob = ifelse(is_dummy, estimate, estimate*sd)*(p_v/p_e)*100)
```

Fourth, we sort the variables according to the magnitude of the effect sizes, and print the results table using `kable`.

```
effect_sizes_df = effect_sizes_df %>%
   arrange(desc(abs(change_prob)))

kable(effect_sizes_df, digits = 4)
```

variable	estimate	t_stat	sd	is_dummy	change_prob
retcall	0.1843	4.1857	0.1773	1	0.7138
creditaa	-0.0905	-11.0304	0.3344	1	-0.3504
eqpdays	0.0003	19.5489	252.6937	0	0.3352
occhmkr	0.0637	1.4134	0.0566	1	0.2469
refurb	0.0561	7.3507	0.3464	1	0.2175
months	-0.0051	-10.8988	9.7285	0	-0.1924
credita	-0.0465	-5.4732	0.3060	1	-0.1802
webcap	-0.0390	-4.3565	0.2955	1	-0.1510
mou	-0.0001	-5.5561	528.6856	0	-0.1333
mailres	-0.0322	-1.5647	0.4864	1	-0.1247
occstud	0.0294	1.0034	0.0875	1	0.1140
mcycle	0.0293	1.3761	0.1163	1	0.1137
changem	-0.0001	-9.1453	253.8868	0	-0.1110
marryun	0.0267	3.2786	0.4838	1	0.1036
setprcm	-0.0238	-2.4737	0.4955	1	-0.0923
children	0.0229	3.3872	0.4312	1	0.0886
uniqsubs	0.0200	6.7008	1.1325	0	0.0877
dropvce	0.0025	1.4844	8.9989	0	0.0874
incmiss	-0.0206	-1.4307	0.4248	1	-0.0798

variable	estimate	t_stat	sd	is_dummy	change_prob
changer	0.0005	6.3966	38.6240	0	0.0770
revenue	0.0004	2.4965	44.0563	0	0.0764
actvsubs	-0.0299	-5.0207	0.6532	0	-0.0757
age1	-0.0008	-3.8069	22.0768	0	-0.0681
recchrge	-0.0007	-3.5772	23.8365	0	-0.0679
newcelly	-0.0174	-2.6616	0.3952	1	-0.0673
prizmrur	0.0171	1.4373	0.2144	1	0.0663
overage	0.0002	2.8024	95.8450	0	0.0653
occself	-0.0165	-0.8557	0.1335	1	-0.0638
peakvce	-0.0002	-2.9350	104.5710	0	-0.0611
phones	0.0115	2.6823	1.3365	0	0.0597
blckvce	0.0013	0.7972	10.5992	0	0.0546
marryyes	0.0138	1.7781	0.4833	1	0.0536
mailflag	-0.0124	-0.6098	0.1201	1	-0.0479
prizmtwn	0.0118	1.5662	0.3573	1	0.0458
occret	-0.0112	-0.5159	0.1206	1	-0.0434
incalls	-0.0007	-2.6953	16.4609	0	-0.0418
roam	0.0012	3.2463	9.0720	0	0.0409
creditcd	0.0101	0.9662	0.4630	1	0.0392
outcalls	0.0003	1.9429	35.0190	0	0.0365
income	-0.0030	-2.0787	3.1096	0	-0.0362
prizmub	-0.0093	-1.5846	0.4676	1	-0.0359
occcler	0.0091	0.5042	0.1415	1	0.0351
dropblk	-0.0006	-0.3429	15.2906	0	-0.0333
unansvce	0.0002	2.0252	38.8999	0	0.0320
threeway	-0.0070	-2.7640	1.1647	0	-0.0316
setprc	0.0001	2.0716	56.9807	0	0.0308
pcown	0.0079	1.0617	0.3913	1	0.0306
truck	0.0067	0.7802	0.3932	1	0.0261
custcare	-0.0013	-2.3523	5.1840	0	-0.0256
age2	-0.0003	-1.6921	23.9138	0	-0.0255
opeakvce	-0.0001	-1.0304	93.3470	0	-0.0231
occcrft	-0.0056	-0.3706	0.1711	1	-0.0217
mourec	0.0000	0.9979	165.9213	0	0.0199
occprof	-0.0047	-0.5970	0.3817	1	-0.0180
models	0.0044	0.6602	0.9085	0	0.0154
refer	-0.0142	-1.4086	0.2594	0	-0.0143
retaccpt	-0.0258	-1.0597	0.1419	0	-0.0142
rv	0.0035	0.3036	0.2759	1	0.0135
callwait	0.0005	0.7261	5.5653	0	0.0113
mailord	0.0012	0.0595	0.4824	1	0.0047
newcelln	0.0010	0.1280	0.3463	1	0.0037
ownrent	0.0008	0.0800	0.4664	1	0.0032
directas	-0.0002	-0.1308	2.1891	0	-0.0016
callfwdv	-0.0005	-0.0841	0.5536	0	-0.0010
travel	-0.0001	-0.0092	0.2347	1	-0.0004
retcalls	-0.0001	-0.0025	0.2014	0	-0.0001

Alternatively, you can write the effect size table to a csv file:

```
write_csv(effect_sizes_df, path = "Assign6_export2.csv")
```

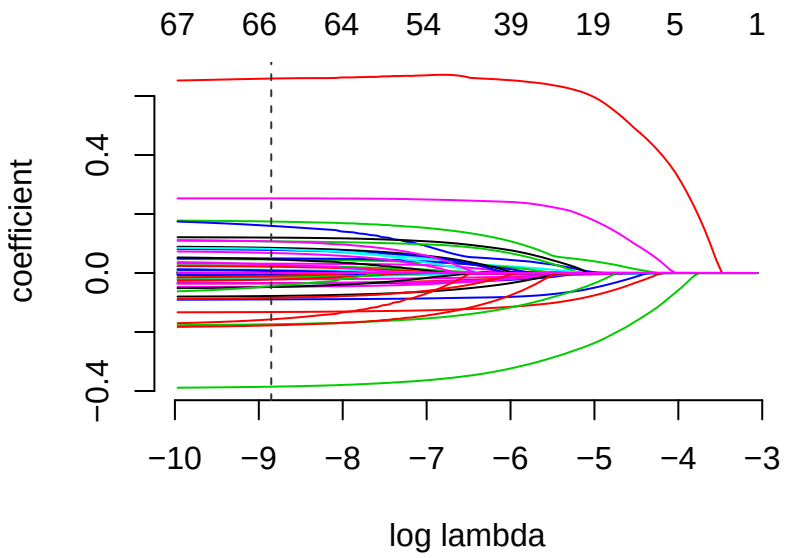
Interpretation of the `change_prob` variable: For a dummy variable, `change_prob` is the change in the churn probability associated with an increase in the dummy value from 0 to 1. For all other variables, `change_prob` is the change in the churn probability associated with an increse in the value of the variable by one standard deviation.

`change_prob` is defined on the scale from 0 to 100. I.e., if `change_prob` takes the value 0.6, the effect size is a change in the churn probability by 0.6%, not by 60%!

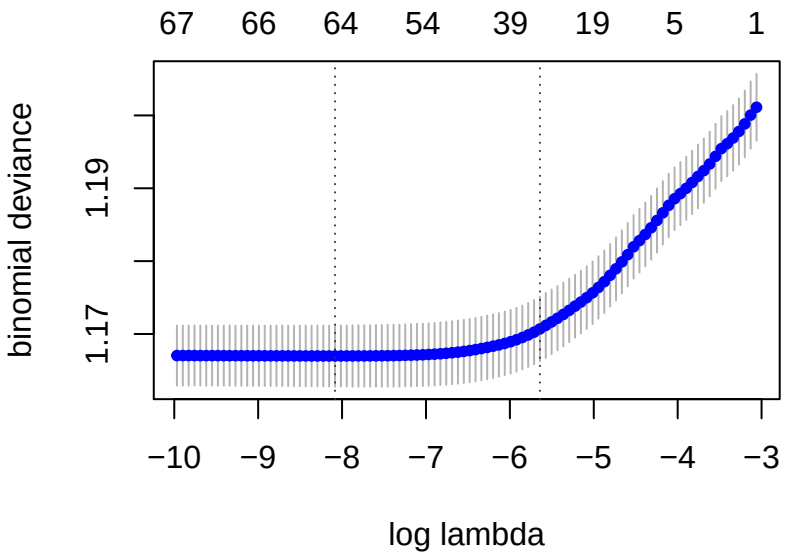
```
library(gamlr)

y = cell2cell_df$churn

x = cell2cell_df[, c(-1, -2, -3)]

cv.fit <- cv.gamlr(x, y,
                   lambda.min.ratio=1e-3,
                   family="binomial",
                   verb=TRUE)
```

```
fold 1,2,3,4,5,done.
```

```
plot(cv.fit$gamlr)
```




```
plot(cv.fit)
```



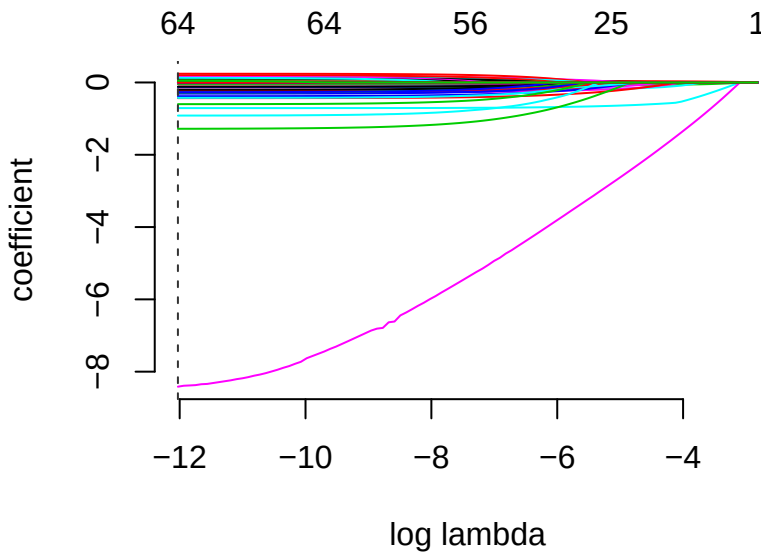

```
c = drop(coef(cv.fit, select="min"))

c[abs(c) > 0.2]
```

```
 intercept   creditaa     refurb    retcall
-0.8338031 -0.3805984  0.2523771  0.6611026
```

```
#creditaa, refurb, retcall are most strong and stable
#check causal about refurb, retcall and creditaa with double lasso

#creditaa
d1 = x[, 32]
#refurb
d2 = x[, 36]
#retcall
d3 = x[, 66]
x = x[, c(-32, -36, -66)]

#double lasso

treat1 <- gamlr(x,d1,lambda.min.ratio=1e-4, family = "binomial")
```

```
Warning in gamlr(x, d1, lambda.min.ratio = 1e-04, family = "binomial"):
numerically perfect fit for some observations.
```

```
plot(treat1)
```




```
d1hat <- predict(treat1, x, type="response")

d1hat = drop(d1hat)

cor(d1hat, d1)^2
```

```
[1] 0.1632827
```

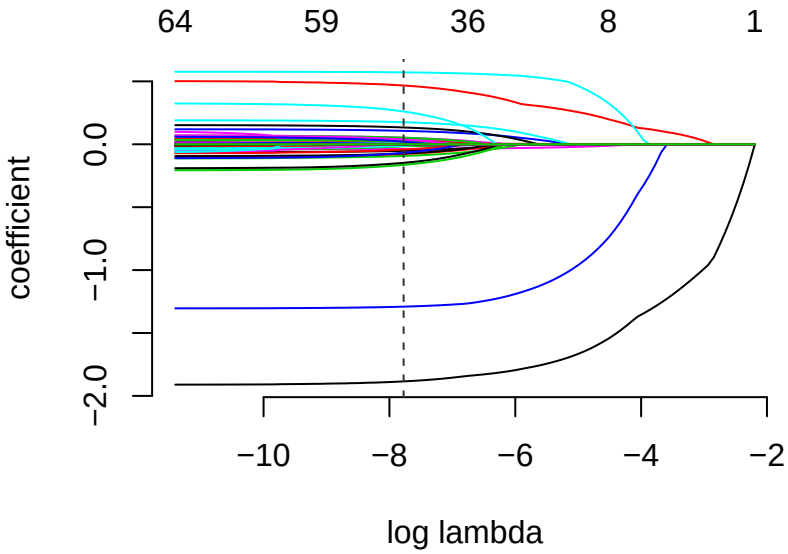
```
causal1 <- gamlr(cBind(d1,d1hat,x),y,free=2,lmr=1e-4, , family = "binomial")
```

```
Warning: 'cBind' is deprecated.
 Since R version 3.2.0, base's cbind() should work fine with S4 objects
```

```
coef(causal1)["d1",]
```

```
[1] -0.3569162
```

```
#which is not 0, so creditaa is causal

treat2 <- gamlr(x,d2,lambda.min.ratio=1e-4, family = "binomial")

plot(treat2)
```




```
d2hat <- predict(treat2, x, type="response")

d2hat = drop(d2hat)

cor(d2hat, d2)^2
```

```
[1] 0.1398452
```

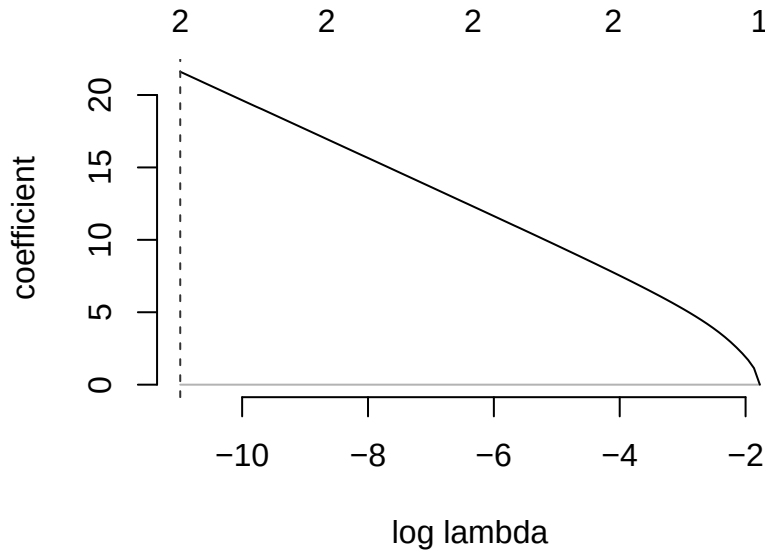
```
causal2 <- gamlr(cBind(d2,d2hat,x),y,free=2,lmr=1e-4, , family = "binomial")

coef(causal2)["d2",]
```

```
[1] 0.2441916
```

```
#which is not 0, so refurb is causal

treat3 <- gamlr(x,d3,lambda.min.ratio=1e-4, family = "binomial")
```

```
Warning in gamlr(x, d3, lambda.min.ratio = 1e-04, family = "binomial"):
numerically perfect fit for some observations.
```

```
plot(treat3)
```




```
d3hat <- predict(treat3, x, type="response")

d3hat = drop(d3hat)

cor(d3hat, d3)^2
```

```
[1] 1
```

```
#prefect cover

causal3 <- gamlr(cBind(d3,d3hat,x),y,free=2,lmr=1e-4, , family = "binomial")

coef(causal3)["d3",]
```

```
[1] 0
```

```
#it is 0, so retcall is not causal

coef(treat3)
```

```
64 x 1 sparse Matrix of class "dgCMatrix"
              seg100
intercept -12.54056
revenue     .
mou         .
recchrge    .
directas    .
overage     .
roam        .
changem     .
changer     .
dropvce     .
blckvce     .
unansvce    .
custcare    .
threeway    .
mourec      .
outcalls    .
```

```
incalls     .
peakvce     .
opeakvce    .
dropblk     .
callfwdv    .
callwait    .
months      .
uniqsubs    .
actvsubs    .
phones      .
models      .
eqpdays     .
age1        .
age2        .
children    .
credita     .
prizmrur    .
prizmub     .
prizmtwn    .
webcap      .
truck       .
rv          .
occprof     .
occcler     .
occcrft     .
occstud     .
occhmkr     .
occret      .
occself     .
ownrent     .
marryun     .
marryyes    .
mailord     .
mailres     .
mailflag    .
travel      .
pcown       .
creditcd    .
retcalls   21.60589
retaccpt    .
newcelly    .
newcelln    .
refer       .
incmiss     .
income      .
mcycle      .
setprcm     .
setprc      .
#the retcall is covered by retcalls.
```

7 Develop incentives to prevent churn

Using the effect size table that you constructed in the previous section, identify some factors that are strongly associated with churn. If actionable, propose an incentive that can be targeted to the customers to prevent churn.

8 Summarize your main results

1. Is customer churn at Cell2Cell predictable from the customer information that Cell2Cell maintains?

Customer churn at Cell2Cell is predictable but not accurate from the customer information that Cell2Cell maintains. The churn rate is predicted with an error rate of 0.7%, relatively accurate. However, the false negative rate is high, where the model predicts low churn probability for positive churns (churn = 1). The false positive rate is low numerically, but is high comparing to the churn rate. The model would be of low sensitivity (true positive rate) and high specificity, while the high specificity is from the high base non-churn rate not the model accuracy.

The model can be used to roughly test the factor importance but not to predict exact customers' churn probability. Better model techniques are needed with variable selections to predict the churns.

2. What factors drive customer churn? Which factors are particularly important?

From the change probability ranking, the top five factors are 'retcall', 'creditaa', 'eqpdays', 'refurbs' and 'months', excluding the ones with low t values (greater than -2 and smaller than 2). The variable importance are also tested and the top three are 'retcall', 'creditaa' and 'refurbs' (per results and plots from section 6.2).

The causality is tested for each of them by double lasso. The factors 'creditaa' and 'refurbs' are causal. The factor 'retcall' is not causal, but highly correlated to 'retcalls'. Variable selection is needed to further improve the model.

3. What incentives should Cell2Cell offer to prevent customer churn?

Cell2Cell need to offer customers more promotions proactively to prevent them from thinking to churn. Once they start to think of churning to call to the retention team, they are much more likely to churn. When they call, Cell2Cell need to offer more promotions, discounts or extra benefits to prevent them from churning.

Customer segmentation is needed to specifity more targeted strategies to different customers. For high credit rating customers, they are much less likely to churn than normal credit rating customers (change_prob(creditaa) ranking second). Cell2Cell needs to offer more promotions or benefits to the lower credit rating customers than higher. Same for customers buying refurbished devices, longer warranties and discounts would be beneficial to prevent them from churning.