Predicción de Abandono

Isidoro Garcia

2021

```
library(tidyverse)
library(data.table)
library(broom)
library(knitr)
library(lubridate)
library(RCT)
library(gamlr)
library(ranger)
library(tree)
library(parallel)
library(tidymodels)
```

Contexto

Cell2Cell es una compañía de teléfonos celulares que intenta mitigar el abandono de sus usuarios. Te contratan para 1) Encontrar un modelo que prediga el abandono con acierto y para usar los insights de este modelo para proponer una estrategia de manejo de abandono.

Las preguntas que contestaremos son:

1. Se puede predecir el abandono con los datos que nos compartieron?
2. Cuáles son las variables que explican en mayor medida el abandono?
3. Qué incentivos da Cell2Cell a sus usarios para prevenir el abandono?
4. Cuál es el valor de una estrategia de prevención de abandono focalizada y cómo difiere entre los segmentos de los usuarios? Qué usuarios deberían de recibir incentivos de prevención? Qué montos de incentivos

Nota: Voy a evaluar las tareas con base en la respuesta a cada pregunta. Como hay algunas preguntas que no tienen una respuesta clara, al final ponderaré de acuerdo al poder predictivo de su modelo vs las respuestas sugeridas.

Datos

Los dotos los pueden encontrar en `Cell2Cell.Rdata`. En el archivo `Cell2Cell-Database-Documentation.xlsx` pueden encontrar documentación de la base de datos.

Cargemos los datos

```
load('Bases input/Cell2Cell.Rdata')
```

1. Qué variables tienen missing values? Toma alguna decisión con los missing values. Justifica tu respuesta

```
missings<-map_dbl(cell2cell %>% select_all(),
                  ~100*sum(is.na(.)/nrow(cell2cell)))

(missings<-missings[missings>0])
```

```
    revenue         mou    recchrge    directas     overage        roam
0.304024097 0.304024097 0.304024097 0.304024097 0.304024097 0.304024097
    changem     changer      phones      models     eqpdays        age1
0.706574521 0.706574521 0.001407519 0.001407519 0.001407519 1.750953594
       age2
1.750953594
```

```
# Revenue, mou, rcchrge, directas, overage, roam, changem changer, phones, models w/zero
cell2cell <-
   cell2cell %>%
   mutate(across(names(missings[1:10]), ~if_else(is.na(.), 0, as.double(.))))

# eqpdays, age1, age2, quitar
cell2cell<-
   cell2cell %>%
   filter(!is.na(eqpdays), !is.na(age1), !is.na(age2))
```

2. Tabula la distribución de la variable `churn`. Muestra la frecuencia absoluta y relativa. Crees que se debe hacer oversampling/undersamping?

```
kable(cell2cell %>%
         group_by(churn) %>%
         summarise(n = n()) %>%
         mutate(share = 100*n/sum(n)), digits = 2)
```

churn	n	share
0	49534	70.96
1	20268	29.04

3. (2 pts) Divide tu base en entrenamiento y validación (80/20). Además, considera hacer oversampling (SMOTE) o undersampling. (Tip: Recuerda que el objetivo final es tener muestra ~balanceada en el traning set. En el validation la distribución debe ser la original)

La distribución esta 70% vs 30%. Si queremos construir un training set = 80%. Dentro del training, hay que hacer el undersampling tal que se balanceen las clases.

```
set_validation <-
   treatment_assign(data = cell2cell,
                    share_control = 0.8,
                    n_t = 1, strata_varlist = 'customer',
                    seed = 1908, key = 'customer')
```

```
Warning: Unknown or uninitialised column: `treat`.
```

```
Warning in if (missfits == "NA") {: the condition has length > 1 and only the
first element will be used
```

```
Warning in if (missfits == "global") {: the condition has length > 1 and only
the first element will be used
```

```
Warning: Unknown or uninitialised column: `treat`.
```

```
set_validation<-set_validation$data
cell2cell<-
   left_join(cell2cell, set_validation %>%
                ungroup %>%
                select(-c(strata, missfit)))

# Divido entre training y validation
cell2cell_V<-
   cell2cell %>%
   filter(treat == 1) %>%
   select(-treat)

rm(set_validation)

cell2cell_T<-
   cell2cell %>%
   filter(treat==0) %>%
   select(-treat)

# Undersampling
cell2cell_1_T<-
   cell2cell_T %>%
   filter(churn==1)

undersampling<-
   treatment_assign(data = cell2cell_T %>% filter(churn==0),
                    share_control = 0.41,
                    n_t = 1,
                    strata_varlist = 'customer', seed = 19987, key = 'customer')
```

```
Warning: Unknown or uninitialised column: `treat`.
```

```
Warning in if (missfits == "NA") {: the condition has length > 1 and only the
first element will be used
```

```
Warning in if (missfits == "global") {: the condition has length > 1 and only
the first element will be used
```

```
Warning: Unknown or uninitialised column: `treat`.
```

```
undersampling<-undersampling$data
```

```
cell2cell_T<-left_join(cell2cell_T,
                       undersampling %>%
                          ungroup() %>%
                          select(-strata, -missfit))

cell2cell_T<-bind_rows(cell2cell_T %>% filter(treat ==0), cell2cell_1_T)
table(cell2cell_T$churn)
```

```
    0     1
16272 16153
```

```
rm(cell2cell_1_T, undersampling)
```

Model estimation

Pondremos a competir 3 modelos:

1. Cross-Validated LASSO-logit
2. Prune Trees
3. Random Forest

4 (2 pts). Estima un cross validated LASSO. Muestra el la gráfica de CV Binomial Deviance vs Complejidad

```
# Matriz de covariates
cell2cell_T$treat<-NULL

X<-
   cell2cell_T %>%
   ungroup() %>%
   select(-customer, -churn)

X<-sparse.model.matrix(~.+0, data = X)
dim(X)
```

```
[1] 32425    66
```

```
churn<-cell2cell_T$churn

# CV LASSO
detectCores()
```

```
[1] 12
```

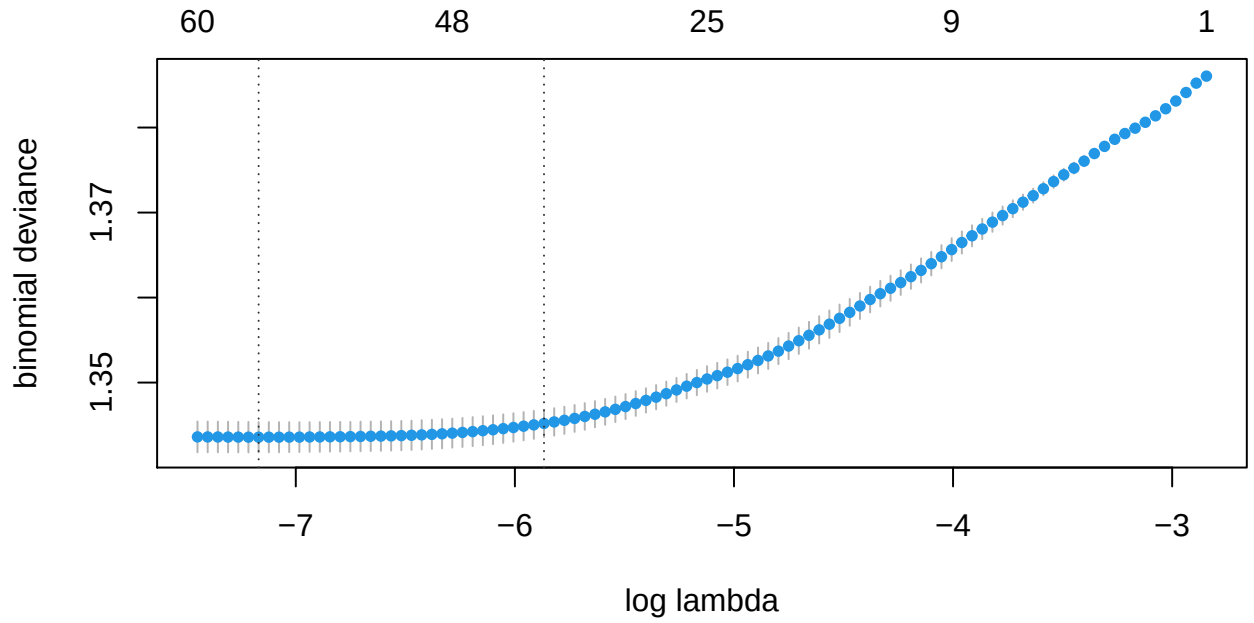
```
cl<-makeCluster(12)

inicio<-Sys.time()
lasso<-cv.gamlr(x = X, y = churn, verb = T, cl = cl, family = 'binomial')
```

```
fold done.
```

```
(tiempo<-Sys.time() - inicio)
```

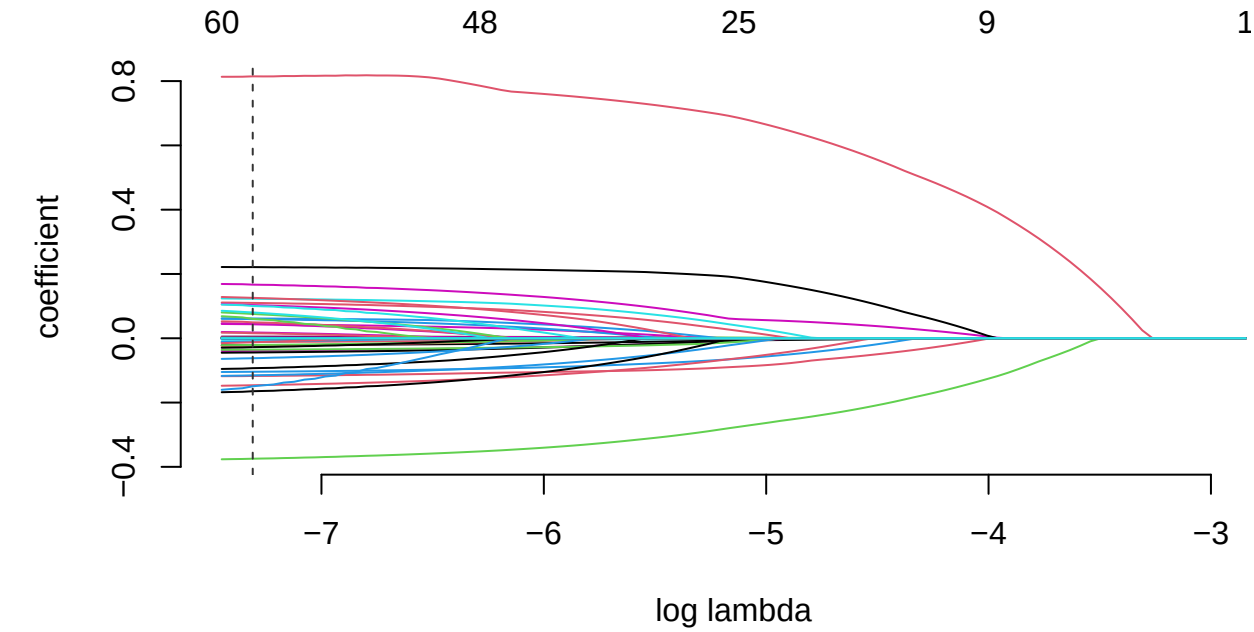
```
Time difference of 14.47572 secs
```

```r
stopCluster(cl)

# Plot del CV binomial dev
plot(lasso)
```



5. Grafica el Lasso de los coeficientes vs la complejidad del modelo.

```r
plot(lasso$gamlr)
```



6 (2 pts). Cuál es la λ resultante? Genera una tabla con los coeficientes que selecciona el CV LASSO. Cuántas variables deja iguales a cero? Cuales son las 3 variables más importantes para predecir el abandono? Da una explicación intuitiva a la última pregunta

Las variables más relevantes son:

- `retcall` [El usuario ya recibió una llamada del equipo de retención]: Esto puede indicar que el usuario está próximo a terminar contrato o que las llamadas del equipo de retención en verdad tienen un efecto contraproducente en términos de la supervivencia del usuario.
- `creditaa` [El usuario tiene la calificación crediticia más alta]: Esta disminuye la probabilidad de abandono. Muy útil en programas de pos-pago.
- `refurb` [El auricular del teléfono es reparado]: Esto podría indicar una experiencia negativa de uso de estos auriculares.

```
lasso$lambda.min
```

```
[1] 0.0007696674
```

```
coeficientes<-as.matrix(coef(lasso, select = 'min'))

coeficientes<-tibble(variable = rownames(coeficientes),
                     coeficiente = coeficientes[,1])

coeficientes<-coeficientes %>% filter(coeficiente !=0)

kable(
   coeficientes<-
   coeficientes %>%
   arrange(desc(abs(coeficiente))))
```

variable	coeficiente
retcall	0.8156834
creditaa	-0.3726389
refurb	0.2208763
uniqsubs	0.1647732
actvsubs	-0.1615133
refer	-0.1443873
retaccpt	-0.1378846
occcler	0.1226888
children	0.1220908
mailres	-0.1165030
credita	-0.1133504
prizmrur	0.1086174
webcap	-0.1034437
mcycle	0.1001941
creditcd	0.0959783
setprcm	-0.0905156
occhmkr	0.0740675
occstud	0.0702888
phones	0.0610163
occret	-0.0597822
rv	0.0581167
intercept	0.0554109
retcalls	0.0516047
marryyes	0.0463030

variable	coeficiente
mailord	-0.0433610
prizmtwn	0.0427891
marryun	0.0412705
threeway	-0.0382060
prizmub	-0.0350963
travel	-0.0258979
callfwdv	-0.0221297
months	-0.0211867
newcelly	-0.0199876
pcown	0.0162476
models	0.0138686
income	-0.0111488
mailflag	-0.0094103
newcelln	-0.0089338
roam	0.0049909
dropvce	0.0049693
age1	-0.0041687
custcare	-0.0036080
changer	0.0023073
recchrge	-0.0021778
dropblk	0.0018898
eqpdays	0.0014490
directas	-0.0012966
overage	0.0011625
truck	0.0009712
age2	-0.0009244
incalls	-0.0008989
revenue	0.0004786
changem	-0.0004677
peakvce	-0.0003127
mou	-0.0002350
setprc	0.0002069
unansvce	0.0001318
outcalls	-0.0000402
blckvce	0.0000014

7. Genera un data frame (usando el validation set) que tenga: `customer`, `churn` y las predicciones del LASSO.

```
X<-
   cell2cell_V %>%
   select(-customer, -churn)

X<-sparse.model.matrix(~.+0, data = X)

table(cell2cell_V$churn)
```

```
   0    1
9846 4115
```

```
predicciones<-
   cell2cell_V %>%
   select(customer, churn) %>%
   mutate(lasso = drop(predict(lasso, newdata = X, select = 'min', type = 'response')))
```

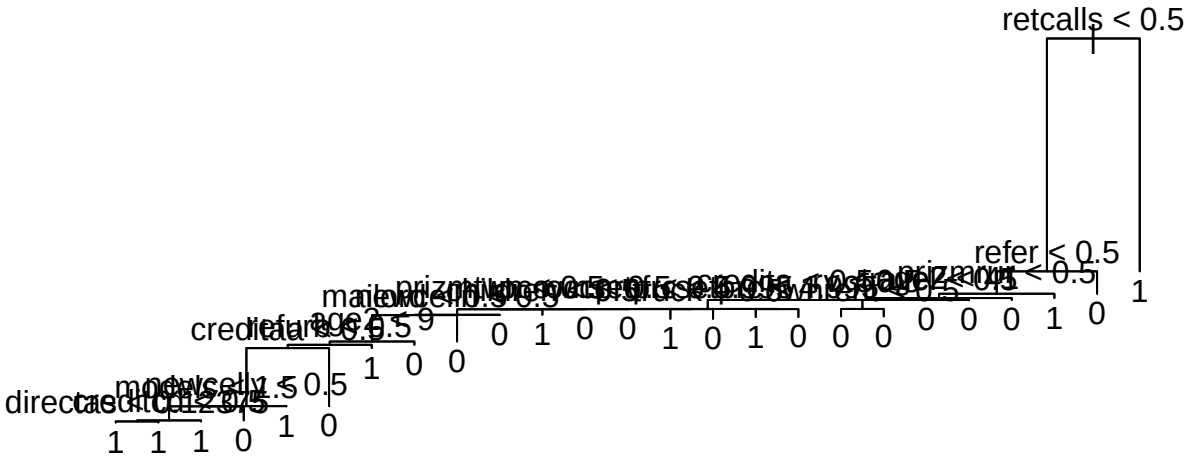
8. Estima ahora tree. Usa `mindev = 0.05, mincut = 1000` Cuántos nodos terminales salen? Muestra el summary del árbol

```
# Primero el árbol
cell2cell_T<-
   cell2cell_T %>%
   mutate(churn = factor(churn, levels = c("0","1")))


arbol<-tree(churn~., split = 'gini',
            cell2cell_T %>% ungroup() %>% select(-customer),
            mindev = 0.05, mincut = 1000)

summary(arbol)
```

```
Classification tree:
tree(formula = churn ~ ., data = cell2cell_T %>% ungroup() %>%
    select(-customer), split = "gini", mindev = 0.05, mincut = 1000)
Variables actually used in tree construction:
 [1] "retcalls" "refer"    "prizmrur" "travel"   "rv"       "credita"
 [7] "truck"    "occprof"  "pcown"    "children" "prizmtwn" "newcelln"
[13] "mailord"  "age2"     "refurb"   "creditaa" "newcelly" "models"
[19] "creditcd" "directas" "setprc"   "ownrent"
Number of terminal nodes:  25
Residual mean deviance:  1.377 = 44630 / 32400
Misclassification error rate: 0.4661 = 15114 / 32425
```

9. Grafica el árbol resultante

```
plot(arbol); text(arbol, pretty = 0)
```

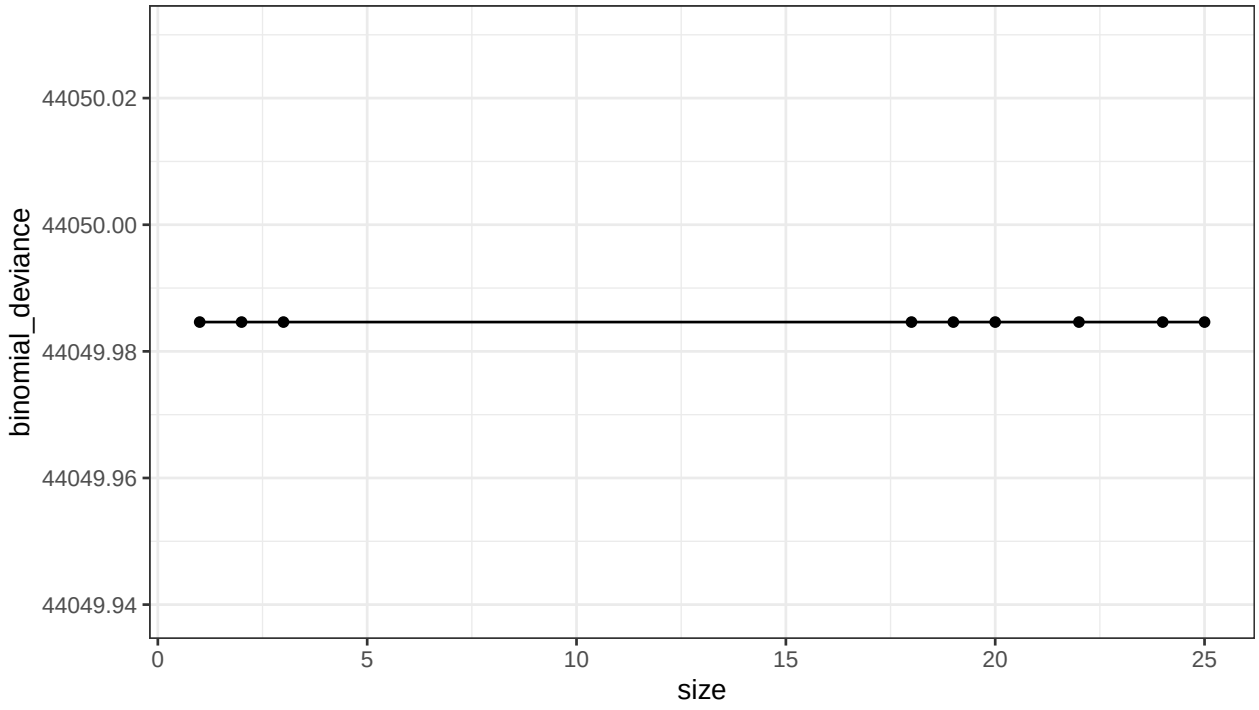


10. Poda el árbol usando CV. Muestra el resultado. Grafica Tree Size vs Binomial Deviance. Cuál es el mejor tamaño del árbol? Mejora el Error?

```
(cv_arbol<-cv.tree(arbol, K = 5))
```

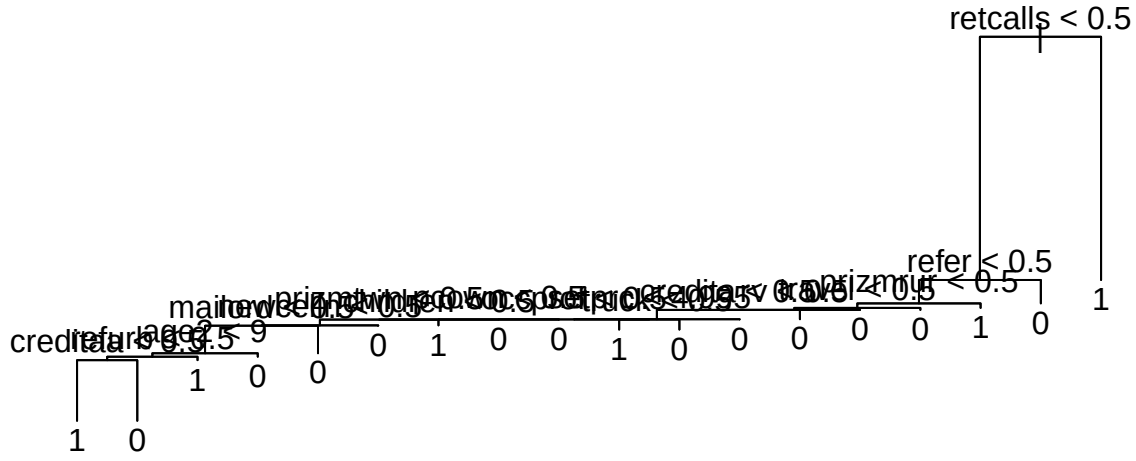
```
$size
[1] 25 24 22 20 19 18  3  2  1

$dev
[1] 44049.98 44049.98 44049.98 44049.98 44049.98 44049.98 44049.98 44049.98
[9] 44049.98

$k
[1]        -Inf   0.2889196   0.8183143   5.6150824   5.8942294   6.2601607
[7]   6.4101217  17.6499275 183.3326909

$method
[1] "deviance"

attr(,"class")
[1] "prune"         "tree.sequence"
```

```
arbol_graf<-
   tibble(size = cv_arbol$size, binomial_deviance = cv_arbol$dev)


ggplot(arbol_graf, aes(size, binomial_deviance))+geom_point()+geom_path()+
   theme_bw()
```



11. Gráfica el árbol final. (Tip: Checa `prune.tree`)

```
arbol_prune<-prune.tree(arbol, best = 4)

plot(arbol_prune); text(arbol_prune, pretty = 0)
```



12. Genera las predicciones del árbol pruned. Guardalas en la base de predicciones. Guarda el score y la prediccion categorica en la misma data frame donde guardaste las predicciones del LASSO

```
X<-
   cell2cell_V %>%
   select(-customer, -churn)

predicciones<-
   predicciones %>%
   mutate(arbol_class = drop(predict(arbol_prune, newdata = X, type = 'class')),
          arbol_prob = drop(predict(arbol_prune, newdata = X, type = 'vector'))[,2])
```

13 (4pts). Corre un Random Forest ahora. Cuál es la B para la que ya no ganamos mucho más en poder predictivo?

- Corre para `num.trees=100,200,300, 500, 700, 800`
- En cada caso, guarda únicamente el `prediction.error`

```
X<-
   cell2cell_T %>%
   ungroup() %>%
   select(-customer, -churn)

X<-sparse.model.matrix(~.+0, data = X)
dim(X)
```

```
[1] 32425    66
```

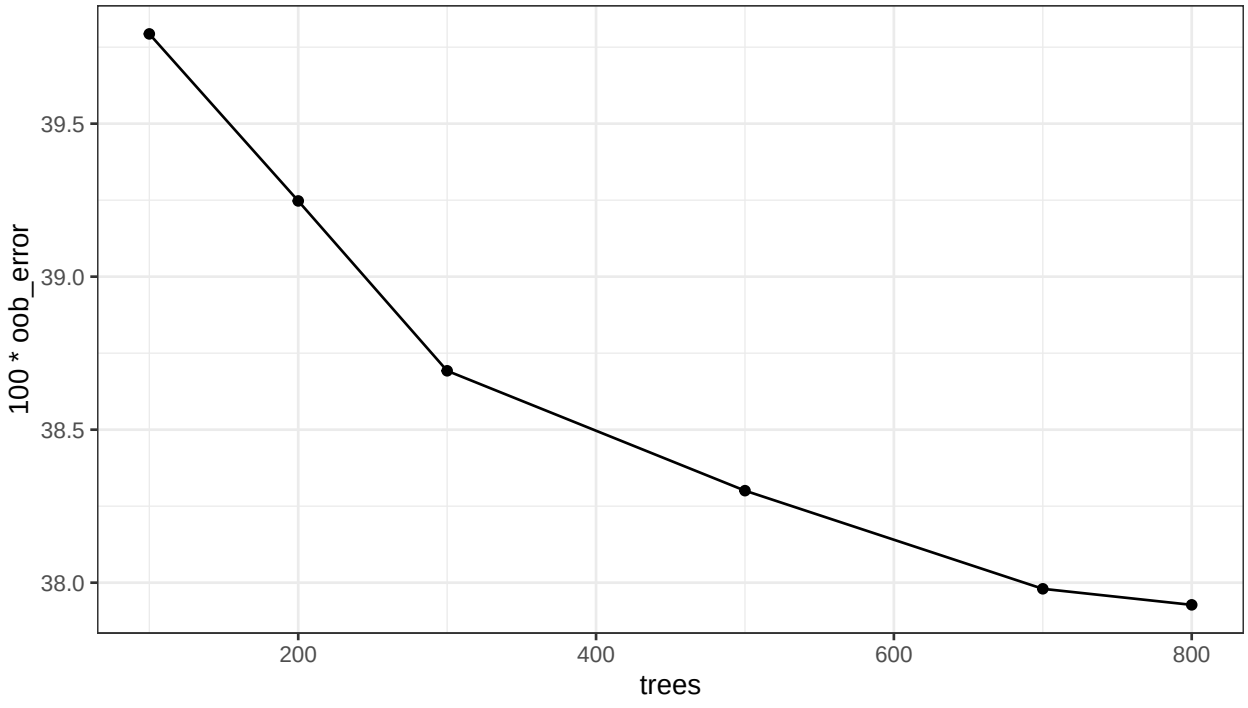
```
churn<-cell2cell_T$churn

# Grid de Random Forests
detectCores()
```

```
[1] 12
```

```
cl<-makeCluster(12)

inicio<-Sys.time()
random_forest<-map(c(100,200,300,500, 700, 800),
                   function(z)
                      ranger(x = X,
                             y = churn,
                             importance = 'impurity',
                             num.trees = z))

(tiempo<-Sys.time() - inicio)
```

```
Time difference of 1.917933 mins
```

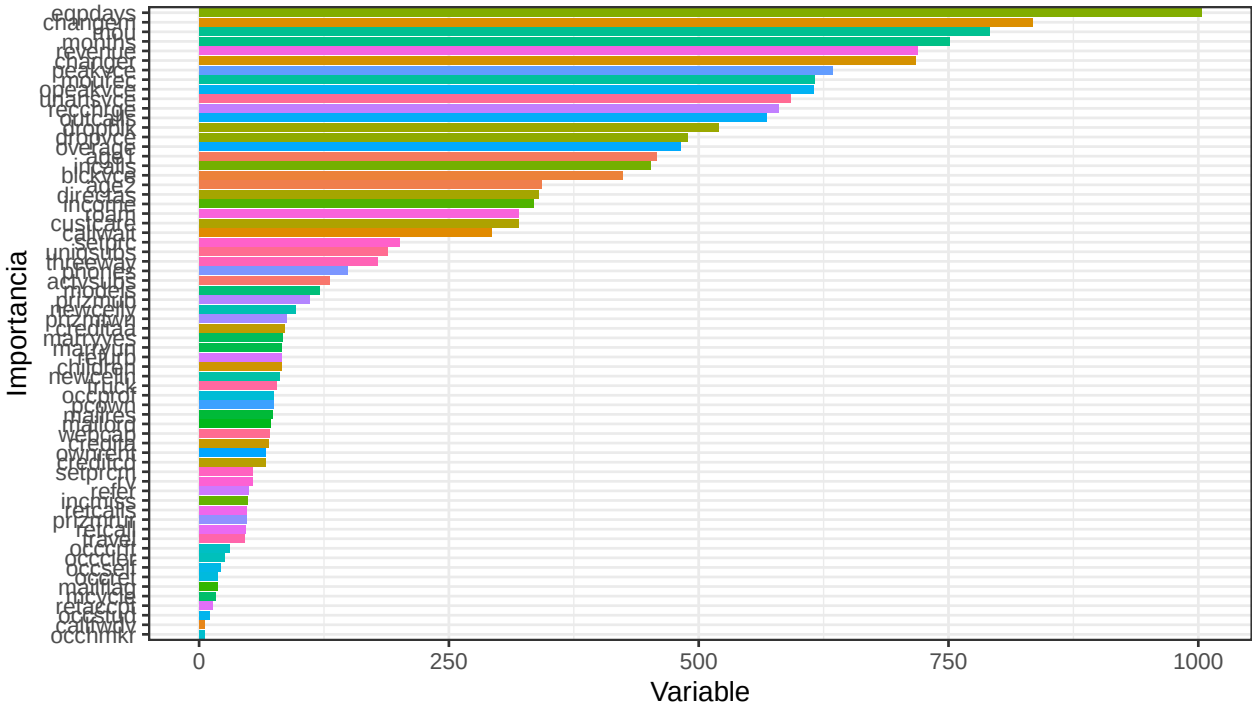
```
stopCluster(cl)

error_prediccion<-tibble(trees = c(100,200,300,500, 700, 800),
                         oob_error = map_dbl(random_forest, ~.$prediction.error))


ggplot(error_prediccion, aes(trees, 100*oob_error))+
   geom_point()+geom_path()+theme_bw()
```



14. Escoge un random forest para hacer las predicciones. Grafica la importancia de las variables. Interpreta

```
random_forest<-random_forest[[6]]

variable_importance<-random_forest$variable.importance
```

```
variable_importance<-tibble(variable = names(variable_importance),
                            importance = variable_importance)

ggplot(variable_importance, aes(fct_reorder(variable, importance), importance, fill = variable))+
   geom_col()+coord_flip()+theme_bw()+theme(legend.position = 'none')+
   labs(y = 'Variable', x = 'Importancia')
```



15. Genera las predicciones OOS para el random forest. Guardalas en la misma data.frame que los otros modelos

```
X<-
   cell2cell_V %>%
   select(-customer, -churn)

X<-sparse.model.matrix(~.+0, data = X)

predicciones<-
   predicciones %>%
   mutate(random_forest = predict(random_forest, data = X)$predictions)
```

16 (2pts). Corre el mismo forest pero ahora con `probability = T`. Esto generará predicciones números en lugar de categóricas. Genera las predicciones continuas y guardalas en el mismo data frame

```
X<-
   cell2cell_T %>%
   ungroup() %>%
   select(-customer, -churn)
```

```
X<-sparse.model.matrix(~.+0, data = X)

detectCores()
```

```
[1] 12
```

```
cl<-makeCluster(12)

inicio<-Sys.time()

random_forest_p<-
   ranger(x = X,
       y = churn,
      importance = 'impurity',
      num.trees = 800,
      probability = T)

(tiempo<-Sys.time() - inicio)
```

```
Time difference of 31.75389 secs
```

```
stopCluster(cl)

# predicciones continuas
X<-
   cell2cell_V %>%
   select(-customer, -churn)

X<-sparse.model.matrix(~.+0, data = X)



predicciones<-
   predicciones %>%
   mutate(random_forest_prob = predict(random_forest_p, data = X)$predictions[,2])


rm(a, arbol, arbol_graf, arbol2, cl)
```

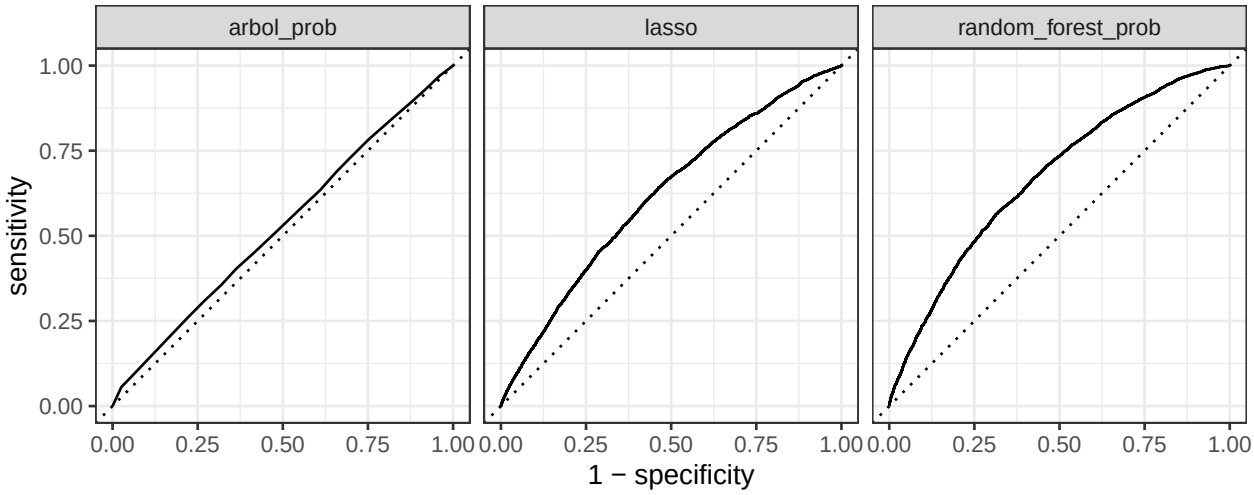
```
Warning in rm(a, arbol, arbol_graf, arbol2, cl): object 'a' not found
```

```
Warning in rm(a, arbol, arbol_graf, arbol2, cl): object 'arbol2' not found
```

17 (4 pts). Genera graficas de las curvas ROC para los tres modelos. Cual parece ser mejor?

```
# Factor truth
predicciones<-
   predicciones %>%
   mutate(churn = factor(churn, levels = c('1', '0')))

# ROC para cada una
predicciones1<-
   predicciones %>%
   pivot_longer(cols = c("lasso", "arbol_prob", "random_forest_prob"),
                names_to = 'modelo',
                values_to = 'pred')
```

```
bases<-predicciones1 %>% split(.$modelo)

rocs<-map2_dfr(.x =  bases,
               .y = names(bases),
               function(x,y) roc_curve(data = x, churn, pred) %>%
                  mutate(modelo = y))


ggplot(rocs, aes(1-specificity, y = sensitivity))+
   geom_path()+
   geom_abline(lty =3)+coord_equal()+theme_bw()+
   facet_wrap(~modelo)
```



18. Genera una tabla con el AUC ROC. Cuál es el mejor modelo ?

```
auc_rocs<-map2_dfr(.x =  bases,
               .y = names(bases),
               function(x,y) roc_auc(data = x, churn, pred) %>%
                  mutate(modelo = y))

kable(auc_rocs)
```

.metric	.estimator	.estimate	modelo
roc_auc	binary	0.5286201	arbol_prob
roc_auc	binary	0.6174667	lasso
roc_auc	binary	0.6709421	random_forest_prob

19 (2pts). Escoge un punto de corte para generar predicciones categoricas para el LASSO basado en la Curva ROC. Genera las matrices de confusión para cada modelo. Compáralas. Qué tipo de error es mas pernicioso?

```
# Fijemos un sensitivity de 80%
corte_lasso<-
   rocs %>%
```

```
   filter(modelo == 'lasso') %>%
   filter(sensitivity >= 0.80) %>%
   arrange(sensitivity) %>%
   slice(1) %>%
   select(.threshold) %>%
   pull()

predicciones<-
   predicciones %>%
   mutate(lasso_cat = if_else(lasso>=corte_lasso, 1, 0),
          lasso_cat = factor(lasso_cat, levels  = c('0', '1')),
          churn = factor(as.character(churn), levels = c('0', '1')))


# Matrices
kable(prop.table(table(predicciones$lasso_cat, predicciones$churn), margin = 2), digits = 2, caption =
```

Table 4: Matriz de Confusión LASSO

	0	1
0	0.34	0.2
1	0.66	0.8

```
kable(prop.table(table(predicciones$arbol_class, predicciones$churn), margin = 2), digits = 2, caption =
```

Table 5: Matriz de Confusión Prune Tree

	0	1
0	0.64	0.6
1	0.36	0.4

```
kable(prop.table(table(predicciones$random_forest, predicciones$churn), margin = 2), digits = 2, captio
```

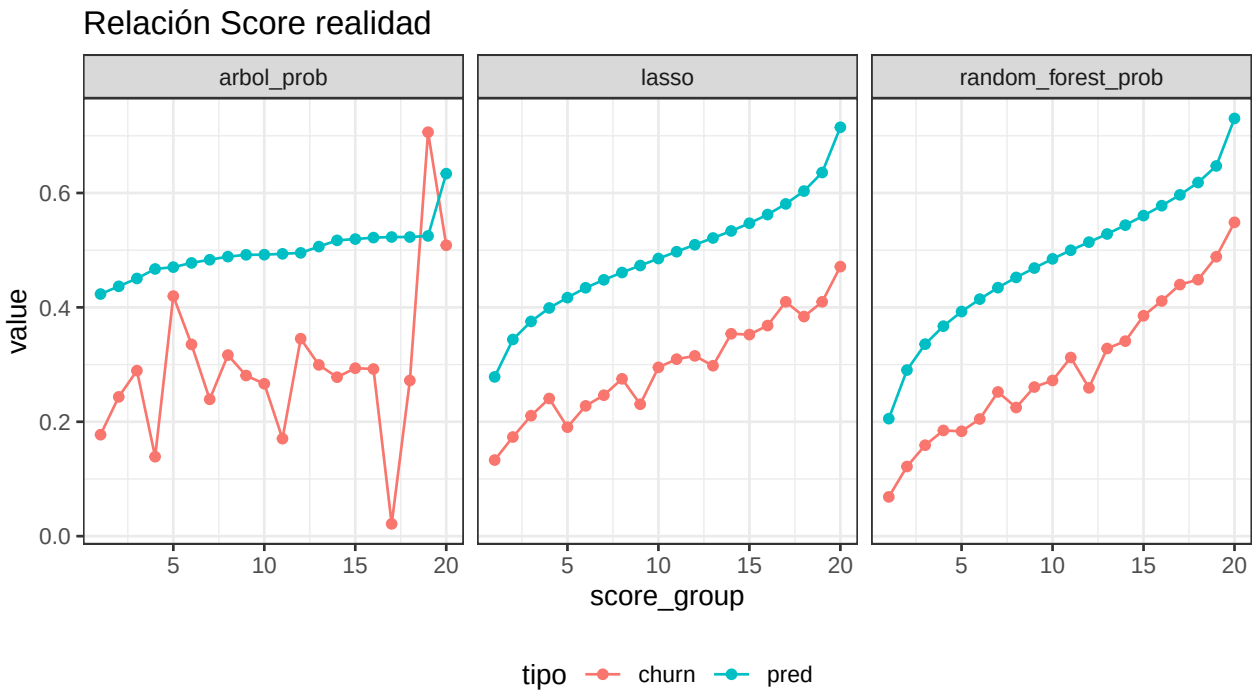
Table 6: Matriz de Confusión RF

	0	1
0	0.6	0.36
1	0.4	0.64

20 (2pts). Finalmente, construye una lift table. Esto es, para 20 grupos del score predecido, genera 1) El promedio de las predicciones, 2) el promedio del churn observado. Existe monotonía? El mejor algoritmo es monotónico? (Tip: usa `ntile` para generar los grupos a partir de las predicciones)

```
predicciones1<-
   predicciones1 %>%
   group_by(modelo) %>%
   mutate(score_group = ntile(x = pred, 20)) %>%
   group_by(modelo, score_group) %>%
```

```r
   summarise(pred  = mean(pred),
             churn = mean(as.numeric(as.character(churn))))


base_grafica<-
   predicciones1 %>%
   pivot_longer(cols = c(pred, churn), names_to = 'tipo', values_to = 'value')

ggplot(base_grafica, aes(score_group, value, fill = tipo, color = tipo))+
   geom_point()+geom_line()+
   facet_wrap(~modelo)+
   theme_bw()+
   labs(title = 'Relación Score realidad')+
   theme(legend.position = 'bottom')
```



21. Concluye. Que estrategia harías con este modelo? Cómo generarías valor a partir de el?