Instituto Tecnológico Autónomo de México

Economía Computacional

Predicción de Abandono con Machine Learning

Prof. Isidoro García Urquieta

Alfredo Lefranc Flores

144346

Cynthia Raquel Valdivia Tirado

81358

Rafael Sandoval Fernández

143689

Marco Antonio Ramos Juárez

142244

Francisco Velazquez Guadarrama

175606

Ciudad de México 2021

Contents

1. 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?

Del análisis llevado a cabo es factible afirmar que es posible predecir la tasa de abandono a partir de los datos de la compañía telefónica y con ello poder generar un estrategia para su detección y mitigación. Esto dado que derivado de este análisis, fue posible determinar los elementos que motivan la pérdida de clientes, por lo que, habiendo detectado esto, es posible implementar mejores estrategias de retención con base en estos resultados y así anticiparse a la fuga de clientes.

2. Cuáles son las variables que explican en mayor medida el abandono?

Acorde con el modelo que mejor logró predecir el abandono en nuestro análisis, el random forest, las variables más importantes que explican en mayor medida el abandono son las siguientes:

1. Eqpdays: El número de días que el cliente permanece con el equipo telefónico
2. Chagem: El porcentaje de minutos de uso

3. Mou: La media mensual de minutos de uso

Como se menciona en el cuerpo del análisis, es factible que los clientes que tengan mayor tiempo con su equipo, se vean tentados a buscar una compañía diferente cuando al fin toman la decisión de renovar y que el abandono de aquellos clientes que utilizan más el servicio medido en minutos, puede deberse a que, precisamente por su intensidad en el uso, estén en constante búsqueda de mejores servicios en términos de calidad y precio.

3. Qué incentivos da Cell2Cell a sus usarios para prevenir el abandono?

Del anáisis de la base de datos de la compañía, fue posible identificar que la empresa envía mails con ofertas con el fin de lograr retener a sus clientes y tiene contabilizadas aquellas que fueron aceptadas (variable retaccpt) y aquellas que fueron respondidas (variable mailres, también envía mails a sus clientes con estas ofertas y hace llamadas de atención periódicamente (variable custcare); incluso, la empresa cuenta con un "equipo de retención" para estos efectos. Asimismo, es posible observar que el director de la empresa responde personalmente algunas de las llamadas que efectúa el cliente (variable directas). Es decir, todos los incentivos de la compañía para prevenir el abandono se basan en la comunicación con el cliente; sin embargo, a falta de un buen modelo predictivo, es posible que no esté focalizando de manera óptima estos esfuerzos en la atención a sus clientes.

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?

Un modelo predictivo como los que se llevaron a cabo en el análisis, permite anticipar potenciales fugas voluntarias de clientes en una empresa. El valor de una estrategia de prevención de abandono focalizada, derivada de estos modelos predictivos, es atacar el problema de pérdida de clientes desde la raíz y poder actuar de manera proactiva en la retención de clientes, toda vez que los modelos de predicción basados en datos, son ideales para descubrir los factores determinantes que están influyendo en los clientes para decidir su permanencia o cancelación del servicio, lo que facilita la personalización de estrategias como la atención a clientes, el ofrecimiento de ofertas, servicios y productos con base en el segmento específico al que pertenecen. Esto es, llevar acabo un análisis detallado de los datos y una comprensión profunda de los patrones que inciden en la tasa de abandono, es crucial para poder tener una segmentación efectiva y lograr establecer acciones de fidelización enfocadas y concretas para mitigiar esta pérdida.

El valor de una estrategia de prevención de abandono focalizada puede diferir entre los segmentos de los usuarios, por ejemplo, si se identifica que la tasa de abandono es mayor en las en las mujeres o en personas de un determinado rango de edad, o si existe el abandono voluntario (que toman la decisión de darse de baja de la compañía) y el involuntario (causas ajenas al cliente como el cambio de ubicación geográfica). Es aquí cuando el modelo predictivo tiene mayor impacto dado que se deberán implementar estrategias de prevención diferenciadas por segmento, y es el modelo el que va a permitir identificar estos segmentos de manera precisa.

Los usuarios que deberían recibir incentivos de prevención son precisamente aquellos para los cuales el modelo predijo que son los que tienden más al abandono del servicio, es decir, aquellos clientes que hayan permanecido mucho tiempo con su equipo telefónico actual y que sean intensivos en su uso. Es probable que sea necesario ofrecerles un incentivo monetario, por ejemplo, paquetes más atractivos en los que se les ofrezca un mayor número de gigabytes a un buen precio y/o que incluyan la renovación del equipo; lo anterior, sin descuidar la atención a clientes, que como ya se señaló, si bien la empresa tiene una estrategia al respecto, e incluso, un equipo de retención, es posible que, sin estar basada en datos, esta estrategia de atención no esté siendo lo suficientemente segmentada y por tanto, personalizada.

2. Datos

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

3. Análisis de Missing Values

3.1 Análisis de missing values general Primero revisamos las columnas que tienen missing values y su cantidad.

```
# funcion para NAs
check_nas <- function(df){
  df %>%
    select_if(~sum(is.na(.)) > 0) %>%
    miss_var_summary()
}

# tabla resumen de missing values
kable(check_nas(data),booktabs=T, align = 'c',
      col.names = c("Variable", "Cantidad","%"),digits = 2)%>%
  kable_styling(position = "center",
                latex_options="HOLD_position")
```

Variable	Cantidad	%
age1	1244	1.75
age2	1244	1.75
changem	502	0.71
changer	502	0.71
revenue	216	0.30
mou	216	0.30
recchrge	216	0.30
directas	216	0.30
overage	216	0.30
roam	216	0.30
phones	1	0.00
models	1	0.00
eqpdays	1	0.00

Revisamos más a detalle estas variables y notamos que todas son numéricas.

Otro elemento importante a tener en cuenta son las coincidencias de los NAs en observaciones. Por

ejemplo, los 216 valores faltantes para revenue, mou, recchrge, directas, overage y roam coinciden. Lo mismo pasa con los 502 valores faltantes de changem y changer, con el valor faltante de phones, models, eqpdays, y con los 1244 valores faltantes de age1 y age2. Esto en general nos dice qué los NAs están agrupados, lo cual es indicio de que tal vez haya un patrón detrás; y de que no hay un problema serio de NAs. No obstante, para no perder dicha información, lo mejor sería hacer imputación.

El primer paso para evaluar la estrategia de imputación es examinar la distribución de estas variables.

3.2 Analisis por variable En esta sección procederemos a hacer un análisis por cada grupo de NAs

age_1 y age_2

Se trata de variables de edad que comienzan a partir de los 18 y para las cuales existe una etiqueta "0" que curiosamente es la moda. Es decir, no conocemos la edad para una gran parte de las observaciones. Podemos aprovechar esta etiqueta y extenderla para tratar los missing values, de tal manera que ahora los NA's tienen la etiqueta "0".

```
# imputación de etiqueta
data$age1[is.na(data$age1)] <- 0
data$age2[is.na(data$age2)] <- 0
```

phones, models y eqpdays

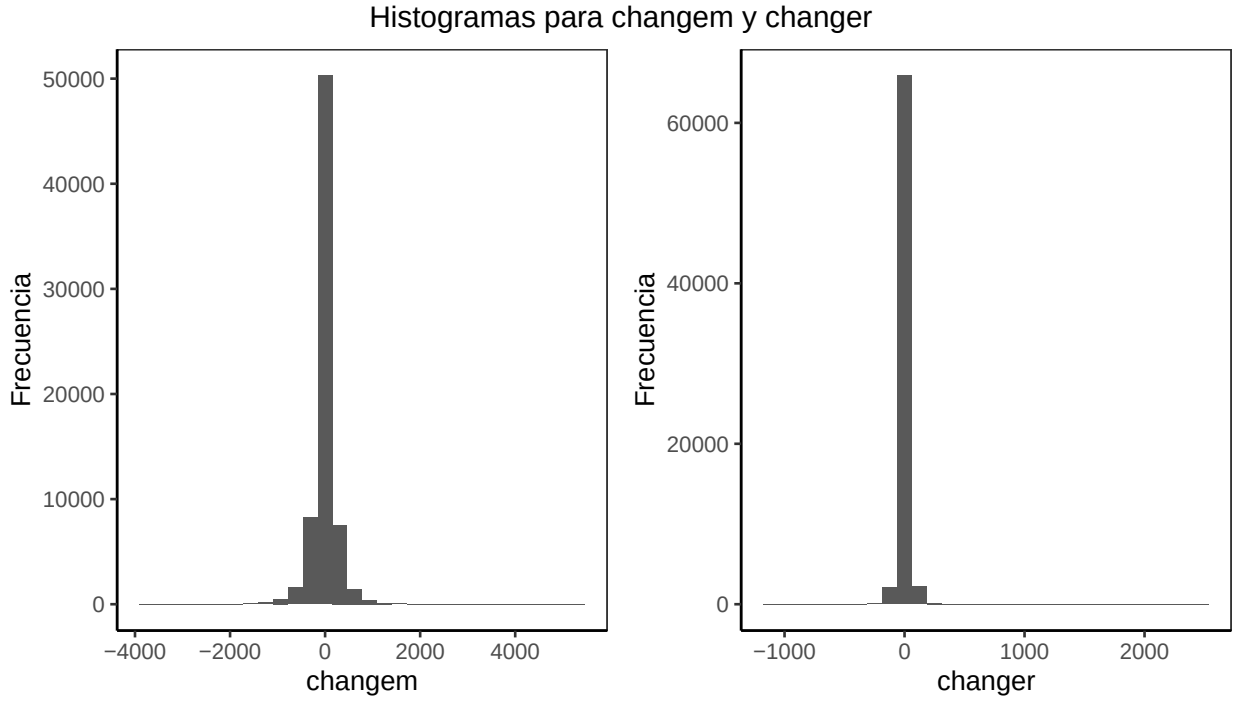
En este caso, solo estamos hablando de un missing value por lo que haremos algo sencillo, imputar la mediana.

```
# imputación de mediana
data$phones[is.na(data$phones)] <- median(data$phones[!is.na(data$phones)]) %>%
  as.numeric
data$models[is.na(data$models)] <- median(data$models[!is.na(data$models)]) %>%
  as.numeric
data$eqpdays[is.na(data$eqpdays)]<- median(data$eqpdays[!is.na(data$eqpdays)])
```

changem y changer

```
# histograma
plot1 <-  ggplot(data, aes_(x=(data$changem)))+
    geom_histogram()+
  ylab("Frecuencia")+xlab("changem")+
  theme_bw()+
  theme(axis.line = element_line(colour = "black"),
    panel.grid.major = element_blank(),
    panel.grid.minor = element_blank(),
```

```
    panel.background = element_blank())

plot2 <-  ggplot(data, aes_(x=(data$changer)))+
    geom_histogram()+
  ylab("Frecuencia")+xlab("changer")+
  theme_bw()+
  theme(axis.line = element_line(colour = "black"),
    panel.grid.major = element_blank(),
    panel.grid.minor = element_blank(),
    panel.background = element_blank())

grid.arrange(plot1, plot2, ncol=2,
             top=textGrob("Histogramas para changem y changer"))
```



En este caso, ante la sospecha notamos que las medias están muy centradas en algún valor cercano a cero pero los outliers son considerables. Por ello, decidimos imputar con la mediana muestral.

```
# imputación de mediana
data$changem[is.na(data$changem)] <- median((data$changem[!is.na(data$changem)]))
data$changer[is.na(data$changer)] <- median((data$changer[!is.na(data$changer)]))
```

revenue, mou, recchrge, directas, overage, roam

Debido a que los 216 valores que faltan son compartidos por todo este grupo de vairables, sería inadecuado hacer imputaciones con base en información entre ellas. Lo que podemos hacer es A) imputar simplemente la media muestral o B) encontrar alguna relación con otras variables que nos permitan imputar con base en un modelo lineal sencillo.

```
# dataframe de correlaciones
aux_data <- data.frame(sapply(data, function(x) as.numeric(as.character(x))))
aux_data<-aux_data[complete.cases(aux_data),]
cor_aux<-cor(aux_data)
cor_aux<-data.frame(cor_aux) %>%
  select (churn,changer,changem,revenue, mou, recchrge, directas, overage, roam)
```

En el data frame cor_aux podemos ver las correlaciones entre nuestras variables con missing values y las demás. Podemos proponer imputar en cada variable el valor predicho por una regresión, De esta manera, lograremos una mejor imputación que con solo la media muestral. Para construir los modelos simplemente elegimos para cada variable las variables más correlacionadas (tanto negativa como positivamente), sin contar revenue, mou, recchrge, directas, overage ni roam pues los missing values son compartidos y no contamos con esa información para estimar.

```
#modelos lineales
imp_revenue<-lm(revenue~peakvce+mourec,data)
imp_mou<-lm(mou~peakvce+opeakvce+mourec+outcalls+unansvce+callwait,data)
imp_recchrge<-lm(recchrge~peakvce+mourec+outcalls,data)
imp_directas<-lm(directas~peakvce+opeakvce+mourec+outcalls+callwait,data)
imp_overage<-lm(overage~peakvce+opeakvce+mourec+outcalls+callwait,data)
imp_roam<-lm(roam~peakvce+opeakvce+mourec+outcalls+callwait,data)
```

```
stargazer(imp_revenue,imp_recchrge,imp_directas,type="latex",
          header=FALSE,
          column.sep.width = "3pt",
          font.size = "small" )
```

```
stargazer(imp_mou,imp_overage,imp_roam,type="latex",
          header=FALSE ,
          column.sep.width = "3pt",
          font.size = "small" )
```

En esta tabla podemos observar que en general los modelos nos dan una predicción mejor a la media, excepto en el último modelo sobre roam. Esta información nos da luz verde para realizar una imputación con base en una regresión lineal. En el caso de roam, solo se imputará la media muestral.

Table 1

	Dependent variable:		
	revenue	recchrge	directas
	(1)	(2)	(3)
peakvce	0.185***	0.101***	0.009***
	(0.002)	(0.001)	(0.0001)
opeakvce			−0.002***
			(0.0001)
mourec	0.074***	0.023***	−0.001***
	(0.001)	(0.001)	(0.0001)
outcalls		−0.005*	0.006***
		(0.003)	(0.0004)
callwait			0.029***
			(0.002)
Constant	33.493***	35.235***	0.153***
	(0.163)	(0.099)	(0.010)
Observations	70,831	70,831	70,831
R^2	0.454	0.319	0.180
Adjusted R^2	0.454	0.319	0.180
Residual Std. Error	32.691 (df = 70828)	19.735 (df = 70827)	1.990 (df = 70825)
F Statistic	29,453.020*** (df = 2; 70828)	11,063.210*** (df = 3; 70827)	3,116.236*** (df = 5; 70825)

Note: *p<0.1; **p<0.05; ***p<0.01

Table 2

	Dependent variable:		
	mou	overage	roam
	(1)	(2)	(3)
peakvce	0.983***	0.230***	0.006***
	(0.017)	(0.005)	(0.001)
opeakvce	0.900***	−0.178***	−0.003***
	(0.021)	(0.006)	(0.001)
mourec	1.498***	0.179***	0.001***
	(0.010)	(0.003)	(0.0004)
outcalls	1.278***	0.208***	0.004**
	(0.045)	(0.014)	(0.002)
unansvce	1.346***		
	(0.038)		
callwait	−1.385***	2.620***	−0.009
	(0.252)	(0.080)	(0.009)
Constant	134.924***	0.622	0.713***
	(1.320)	(0.418)	(0.048)
Observations	70,831	70,831	70,831
R^2	0.777	0.319	0.005
Adjusted R^2	0.777	0.319	0.005
Residual Std. Error	250.401 (df = 70824)	79.533 (df = 70825)	9.060 (df = 70825)
F Statistic	41,109.470*** (df = 6; 70824)	6,623.726*** (df = 5; 70825)	67.635*** (df = 5; 70825)

Note: *p<0.1; **p<0.05; ***p<0.01

```
#imputo la predicción de la regresión
data<-data %>%
  mutate(revenue=ifelse(is.na(revenue),
                        predict(imp_revenue,newdata=data),
                        revenue))%>%
       mutate(mou=ifelse(is.na(mou),predict(imp_mou,newdata=data),mou))%>%
       mutate(recchrge=ifelse(is.na(recchrge),
                              predict(imp_recchrge,newdata=data),
                              recchrge)) %>%
       mutate(directas=ifelse(is.na(directas),predict(imp_directas,newdata=data),
                              directas)) %>%
       mutate(overage=ifelse(is.na(overage),predict(imp_overage,newdata=data),
                             overage))

#imputo la predicción la media muestral
data$roam[is.na(data$roam)] <- mean(data$roam[!is.na(data$roam)])
```

Finalmente checo mi base:

```
#imprime tabla con NAs
cols_con_nas <- names(which(colSums(is.na(data)) > 0))
df_nas <- data %>% select(all_of(cols_con_nas)) %>% as.data.frame()
summary(df_nas)
```

< table of extent 0 x 0 >

```
# como no existe nos dice que el tamaño es 0x0
```

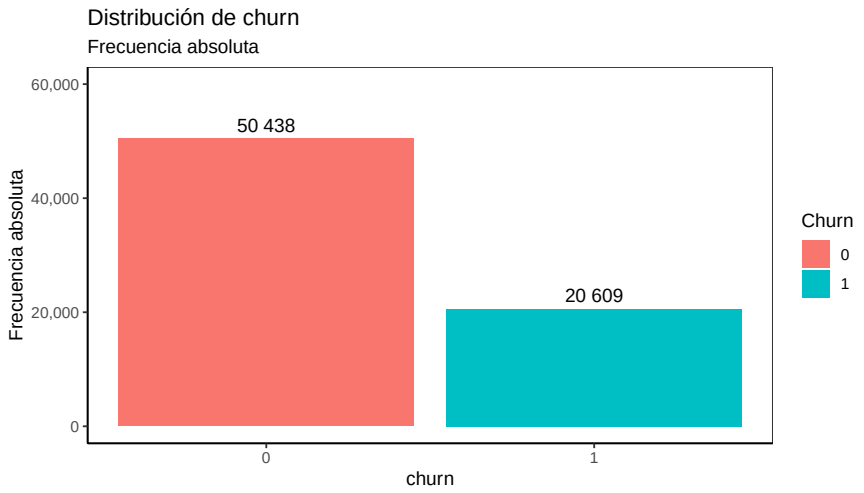
4. Remuestreo

Tabulamos la distribución de la variable `churn`, y mostramos la frecuencia absoluta y relativa. Así decidimos si conviene hacer oversampling, undersamping o una combinación de ambas.

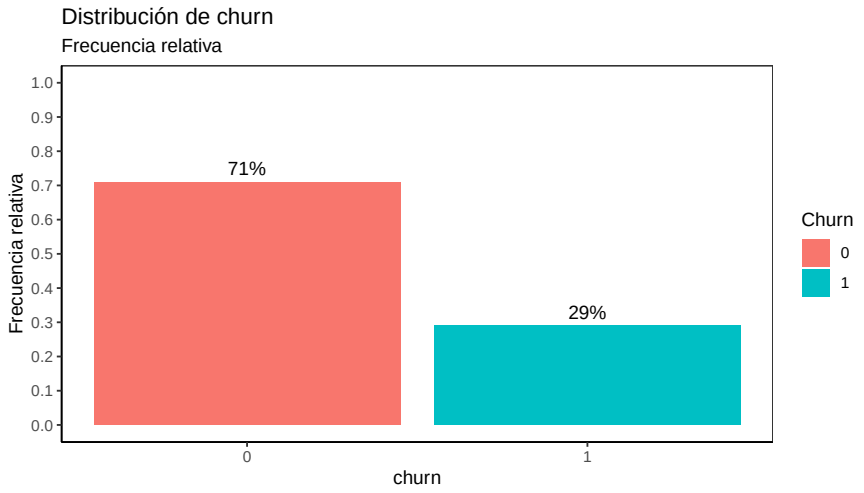
```
# tabulación
tabulado <- data %>% group_by(churn) %>% dplyr::summarise(frecuencia_absoluta=n()) %>%
  mutate(frecuencia_relativa = frecuencia_absoluta/sum(frecuencia_absoluta))

kable(tabulado,booktabs=T, align = 'c',
      col.names = c("Churn", "Cantidad","%"),digits = 2)%>%
  kable_styling(position = "center",
                latex_options="HOLD_position")
```

Churn	Cantidad	%
0	50438	0.71
1	20609	0.29

```
rm(tabulado)



data$churn <- as.numeric(data$churn)
# frecuencia absoluta
ggplot(data, aes(x=churn))+
  geom_bar(aes(y=..count..,fill=(factor(..x..)),stat="count"))+
  geom_text(aes(label= scales::number(..count..),
                y=..count..),stat="count", vjust=-0.5)+
  labs(title="Distribución de churn",
    subtitle="Frecuencia absoluta",
    fill="Churn")+
  ylab("Frecuencia absoluta")+
  scale_y_continuous(labels=scales::comma,
                     limits=c(0,60000))+
  scale_x_continuous(breaks=c(0,1))+
  theme_bw()+
  theme(axis.line = element_line(colour = "black"),
    panel.grid.major = element_blank(),
    panel.grid.minor = element_blank(),
    panel.background = element_blank())
```



```
# frecuencia relativa
ggplot(data, aes(x=churn))+
  geom_bar(aes(y=..prop.., fill=factor(..x..)),stat="count")+
    geom_text(aes(label= scales::percent(..prop..),
                  y=..prop..),stat="count", vjust=-0.5)+
  scale_y_continuous(breaks=seq(0,1, by =0.1),
                     limits=c(0,1))+
  scale_x_continuous(breaks=c(0,1))+

    labs(title="Distribución de churn",
       subtitle="Frecuencia relativa",
       fill="Churn")+
  ylab("Frecuencia relativa")+
  theme_bw()+
  theme(axis.line = element_line(colour = "black"),
    panel.grid.major = element_blank(),
    panel.grid.minor = element_blank(),
    panel.background = element_blank())
```



Sí, parece que sí se debe hacer algún tipo de remuestreo para tener una base balanceada y que los modelos a estimar tengan mayor poder predictivo.

4.1 Set-validación Ya que conocemos el desbalance de la clase objetivo, dividimos la base en entrenamiento y validación (80/20).

```
set.seed(123)

# proporción que queremos de training
```

```
training_size <- 0.8

# filas de training
training_rows <- sample(seq_len(nrow(data)),
                        size=floor(training_size*nrow(data)))

#training set
data_training <- data[training_rows,]

#validation set. guardamos la base de características y
# la variable objetivo por separado
data_validation <- data[-training_rows,-2]
churn_validation <- data[-training_rows,2]

# la muestra está balanceada? --> NO
tabulado <- data_training %>% group_by(churn) %>%
  dplyr::summarise(frecuencia_absoluta=n()) %>%
  mutate(frecuencia_relativa = frecuencia_absoluta/sum(frecuencia_absoluta))
kable(tabulado, booktabs=T, caption="Balance en los datos originales",
      col.names = c("Churn","Frecuencia absoluta","Frecuencia relativa"))%>%
  kable_styling(position = "center",
                latex_options="HOLD_position")
```

Table 3. Balance en los datos originales

Churn	Frecuencia absoluta	Frecuencia relativa
0	40303	0.7090979
1	16534	0.2909021

No está balanceada, aunque sí conserva la proporción de clases de la base original. Procedemos a rebalancear, probamos 3 estrategias: undersampling, oversampling o una mezcla de ambas. A continuación mostramos cómo realizamos las tres estrategias sobre nuestra base de entrenamiento.

```
# tamaño de churn==1 en la base de entrenamiento
freq_churn <- data_training %>% filter(churn==1) %>% nrow()


#simplemente reducimos la clase más abundante
```

```
undersampling_c0<-  sample_n((data_training %>% filter(churn==0)),
                             size=freq_churn,
                             replace = FALSE)
# Evalúo balance
undersampling<-rbind((data_training %>%
                        filter(churn==1)),undersampling_c0)
tabulado <- undersampling %>% group_by(churn) %>%
    dplyr::summarise(frecuencia_absoluta=n()) %>%
    mutate(frecuencia_relativa = frecuencia_absoluta/sum(frecuencia_absoluta))
kable(tabulado, booktabs=T, caption="Balance con undersampling",
      col.names = c("Churn","Frecuencia absoluta","Frecuencia relativa"))%>%
  kable_styling(position = "center",
                latex_options="HOLD_position")
```

Table 4. Balance con undersampling

Churn	Frecuencia absoluta	Frecuencia relativa
0	16534	0.5
1	16534	0.5

A. Undersampling

```
# librería smotefamily
# fuente: https://rikunert.com/SMOTE_explained
smote <- smotefamily::SMOTE((data_training)[,-2], # data
                            data_training$churn, # class attribute
                            K=10, # number of nearest neighbors
                            dup_size=2 # minority instances
                            # over original number of majority instances
                            )


class(data %>% na.exclude)
oversampled <- as.data.frame(smote$data)

# smote guarda la variable de clase como class.
# se renombra y convierte a numerica
colnames(oversampled)[68] <- "churn"
```

```
oversampled$churn <- oversampled$churn %>% as.numeric

# ahora está mmejor balanceado pero como el SMOTE da vueltas completas,
# en este caso de tamaño 2, el balance no es perfecto
tabulado <- oversampled %>% group_by(churn) %>%
    dplyr::summarise(frecuencia_absoluta=n()) %>%
    mutate(frecuencia_relativa = frecuencia_absoluta/sum(frecuencia_absoluta))
kable(tabulado, booktabs=T, caption="Balance con oversampling",
      col.names = c("Churn","Frecuencia absoluta","Frecuencia relativa"))%>%
  kable_styling(position = "center",
                latex_options="HOLD_position")
```

B. oversampling mediante Synthetic Minority Oversampling Technique (SMOTE)

```
# para rebalancear de manera más precisa el oversampling, simplemente
# "podamos"(con un remuestreo) las observaciones sinteticas de tal manera
# que nos quede una base de datos perfectamente balanceada.

freq_churn_os <- oversampled %>% filter(churn==0) %>% nrow()

#podamos
undersampling_c1<-  sample_n((oversampled %>% filter(churn==1)),
                             size=freq_churn_os,replace = FALSE)

#armamos la base
under_over_sampling<-rbind((oversampled %>% filter(churn==0)),undersampling_c1)

#balance perfecto
tabulado <- under_over_sampling %>% group_by(churn) %>%
  dplyr::summarise(frecuencia_absoluta=n()) %>%
    mutate(frecuencia_relativa = frecuencia_absoluta/sum(frecuencia_absoluta))

kable(tabulado,booktabs=T, caption="Balance con over y undersampling",
      col.names = c("Churn","Frecuencia absoluta","Frecuencia relativa"))%>%
  kable_styling(position = "center",
                latex_options="HOLD_position")
```

C. Under y over sampling De esta manera quedan 3 bases para comparar: undersampling, oversampling y una combinación de ambos. Comparamos las tres bases con los tres modelos y

obtuvimos los mejores resultados de la base con undersampling. En adelante, mostramos nuestros resultados con esa base. En el script `remuestreo_alt.R` incluimos algunos resultados con las otras bases.

```
data_training_a<-undersampling %>% na.exclude
```

5. Estimación de modelos

Pondremos a competir 3 modelos:

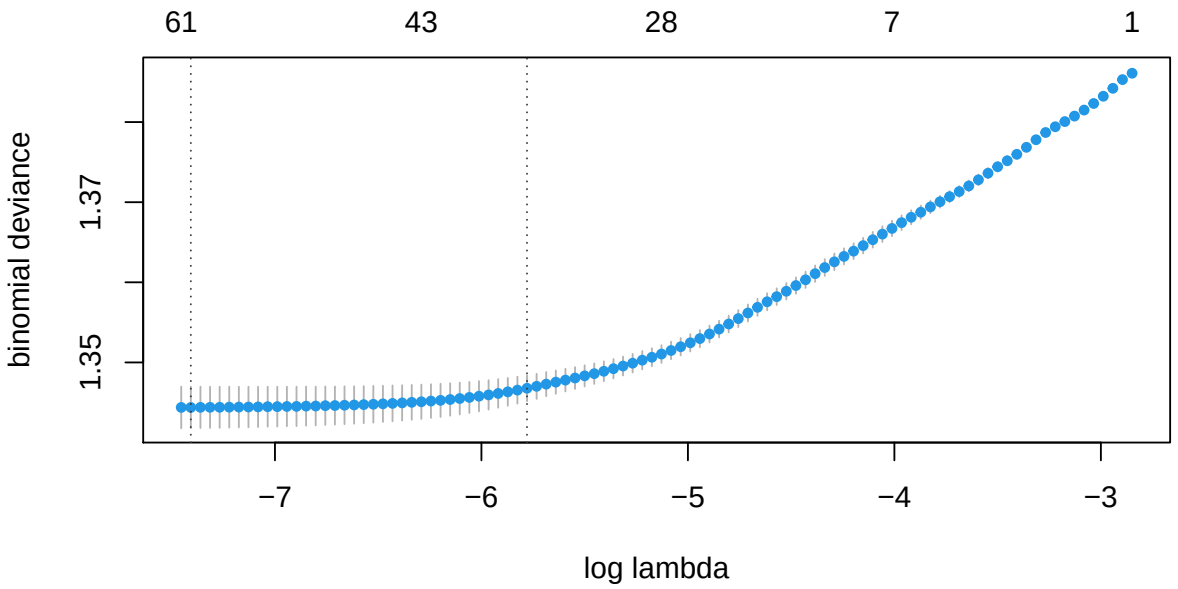
1. Cross-Validated LASSO-logit
2. Prune Trees
3. Random Forest

5.1 Cross Validated LASSO. Estimamos un CV Lasso y mostramos la gráfica de CV Binomial Deviance vs Complejidad.

```
#Matriz de covariates
Xa <-data_training_a %>% select(-customer,-churn)

#se quita intercepto
Xa <- sparse.model.matrix(~.+0, data = Xa)

#vector de Y´s
Ya<-data_training_a$churn

#CV LASSO
cvlasso_a<-cv.gamlr(x = Xa, y = Ya, verb = T, family = 'binomial', nfold = 5)
```

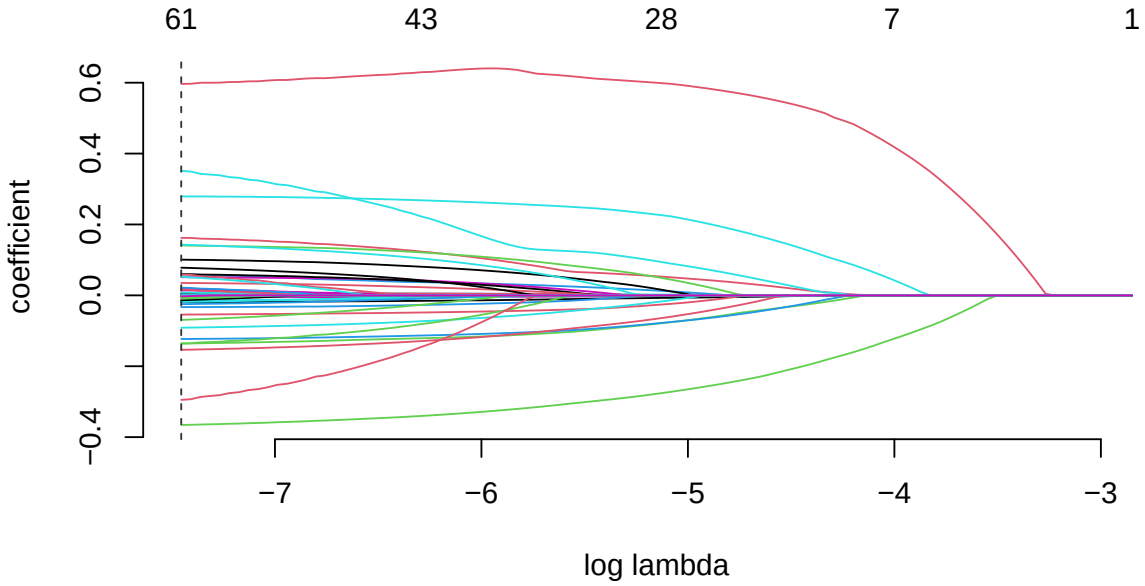
```
fold 1,2,3,4,5,done.
```

```
#Grafica
plot(cvlasso_a)
```



Posteriormente graficamos los coeficientes contra la complejidad del modelo.

```
plot(cvlasso_a$gamlr)
```



Para encontrar la λ óptima, enlistamos las variables que quedan en cero, y revisamos las variables más importantes para predecir el abandono.

```
# Identificador para el lambda deseado
# Valor del lambda deseado
#lambda resultante

a_lambda<- colnames(coef(cvlasso_a, select="min"))
cvlasso_a$gamlr$lambda[a_lambda]
```

```
      seg99
0.0006067582
```

Son 6 variables a las que deja iguales a cero:

1. blckvce: la media de las llamdas bloqueadas
2. models: número de modelos expedidos
3. occcrft: la dummy de ocupación de artesanos
4. occstud: la dummy de ocupación de estudiantes
5. mailord: la dummy de los que compran vía mail
6. incmiss: la dummy de faltantes de ingreso

Se determinó que, dado el tamaño de su coeficiente, las tres variables más importantes para predecir el abandono son las siguientes:

1. retcall: la dummy de que el cliente ha hecho llamadas al equipo de retención
2. refurb: la dummy de que el celular está renovado
3. uniqsubs: el número de personas incluidas en la cuenta del teléfono

Las variables arrojadas por el modelo como las más importantes para la predicción son razonables ya que si el cliente ha hecho llamdas al equipo de retención, es posible que sea para efectuar una queja o reportar alguna deficiencia en el servicio; los clientes generalmente llaman para comunicar quejas o inconformidades, lo que implicaría que, ante tal descontento, puedan cancelar el servicio con la compañía telefónica. Asimismo, el que se haya hecho una renovación del teléfono celular, puede implicar que el cliente haya sido captado por otra compañía que le haya ofrecido el teléfono y por ello efectuó la renovación, o que la renovación haya estado aparejada de un paquete telefónico de manera que el cliente ya no necesite el servicio de telefonía de Cell2Cell. Por último, la media del número de personas incluidas en la cuenta telefónica es 1.5, es decir, menos de dos personas por cuenta, por tanto, es posible que sea más sencillo tomar la decisión de abandonar la compañía telefónica o cambiar de compañía cuando en tu servicio no están incluidas más personas.

Guardamos las predicciones del LASSO en una tabla independiente.

```
#Predicciones
lasso_score <- predict(cvlasso_a,
```

```
                newdata = data_validation[,-1],
                type='response',
                select = "min")

#dataframe
A <- data.frame(data_validation$customer, churn_validation, lasso_score)
colnames(A)[3] <- c('lasso_score')
```

5.2 Árbol Ahora estimamos un árbol. Usamos `mindev = 0.05, mincut = 1000`.

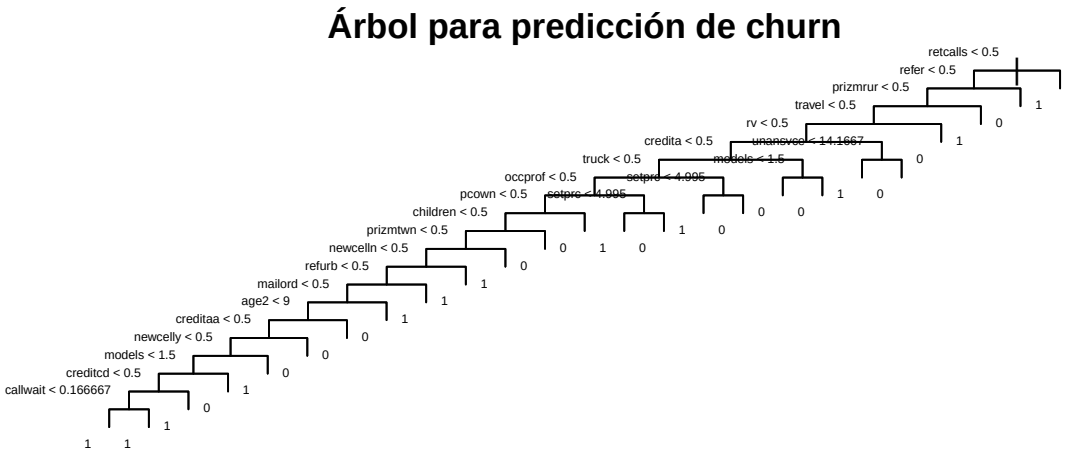
```
tree_control <- tree.control(nobs=nrow(data_training_a[,-1]),
             mincut = 1000,
             mindev = 0.05)


tree_estimation <- tree(as.factor(churn) ~ . ,
                        data = data_training_a[,-1],
                        control = tree_control,
                        split = c("gini"))

summary(tree_estimation)
```

```
Classification tree:
tree(formula = as.factor(churn) ~ ., data = data_training_a[,
    -1], control = tree_control, split = c("gini"))
Variables actually used in tree construction:
 [1] "retcalls" "refer"    "prizmrur" "travel"   "rv"       "credita"
 [7] "truck"    "occprof"  "pcown"    "children" "prizmtwn" "newcelln"
[13] "refurb"   "mailord"  "age2"     "creditaa" "newcelly" "models"
[19] "creditcd" "callwait" "setprc"   "unansvce"
Number of terminal nodes:  25
Residual mean deviance:  1.376 = 45480 / 33040
Misclassification error rate: 0.4655 = 15394 / 33068
```

Obtenemos 25 nodos finales

El árbol resultante es el siguiente:

```
plot(tree_estimation,type = "uniform")
text(tree_estimation,pretty=0, pos='2.8',cex=0.4)
title("Árbol para predicción de churn",line=0.5)
```



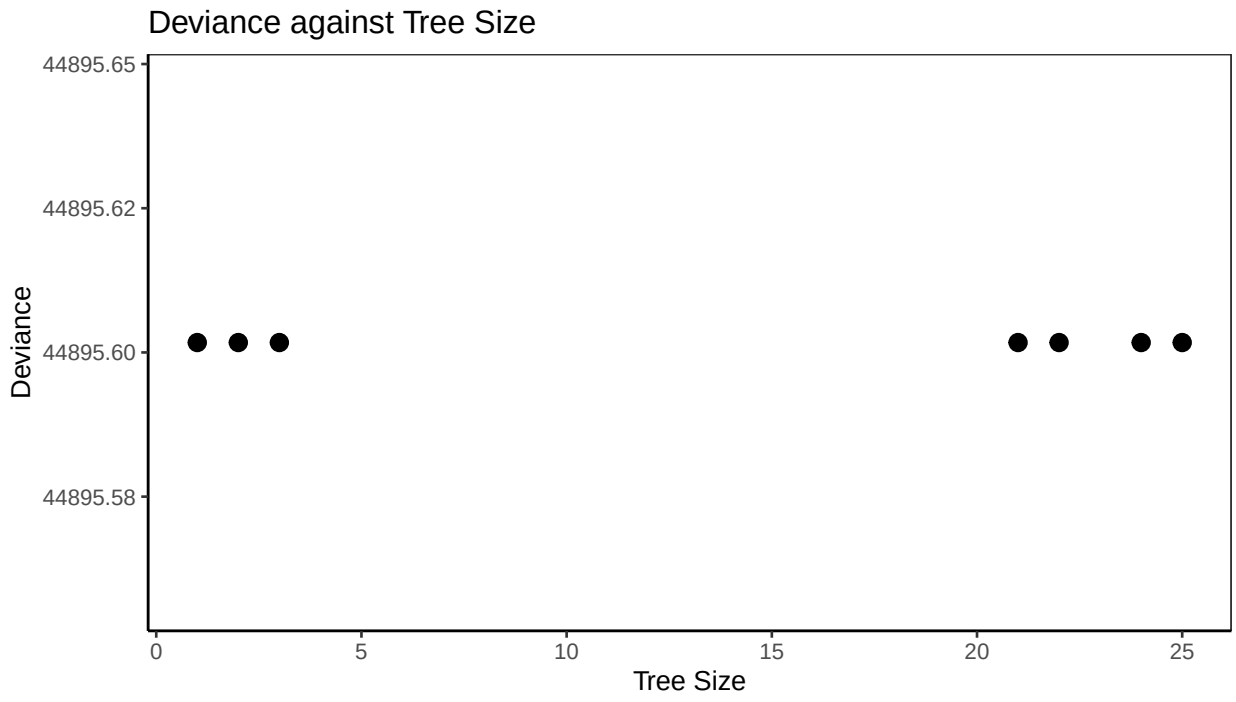
Ahora podamos el árbol usando cross-validation.

```
cv_tree <- cv.tree(tree_estimation, K=10)
cv_tree
```

```
$size
[1] 25 24 22 21  3  2  1

$dev
[1] 44895.6 44895.6 44895.6 44895.6 44895.6 44895.6 44895.6

$k
[1]        -Inf   0.1267368   1.3400605   4.1129528   8.1676465  14.0495505
[7] 193.9543018

$method
[1] "deviance"

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

```
# Size con menor deviance
min_dev_size <- cv_tree$size[match(min(cv_tree$dev),cv_tree$dev)]
```

```
# Gráfica Tree Size vs. Binomial Deviance
plot_size_dev <- function(cv_tree){

  ggplot(data=as.data.frame(cbind(size=cv_tree$size,dev=cv_tree$dev)),
       aes(x=size,y=dev))+
  geom_point(size=3)+
  labs(title="Deviance against Tree Size")+
  xlab("Tree Size")+
  ylab("Deviance")+
  theme_bw()+
    theme(axis.line = element_line(colour = "black"),
    panel.grid.major = element_blank(),
    panel.grid.minor = element_blank(),
    panel.background = element_blank())

                                    }

plot_size_dev(cv_tree)
```

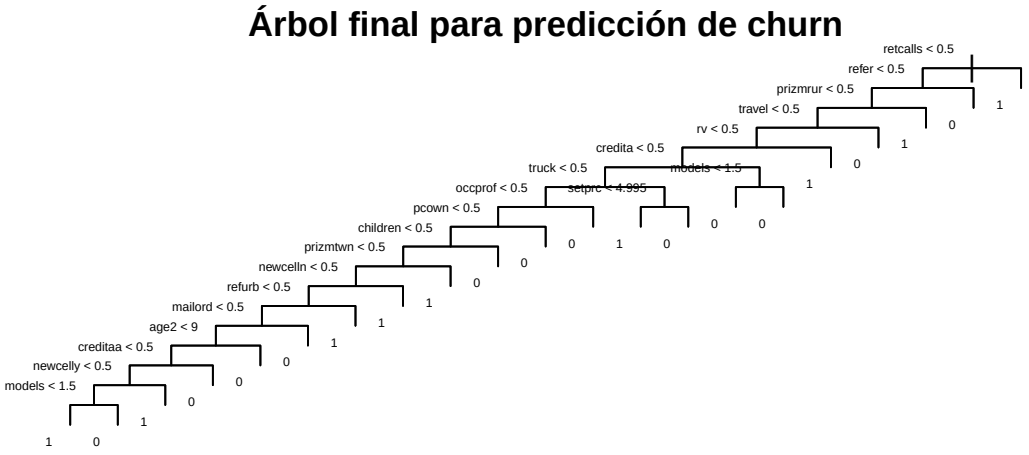


Por lo general buscaríamos el árbol que nos de menor *deviance* , en este caso tenemos una *deviance*

constante. Esto nos podría indicar que los split son espurios y probablemente este modelo no sea el adecuado para llevar a cabo la predicción de *churn*. Por ahora, elegimos el modelo de 4 nodos.

En seguida se muestra el árbol final:

```
tree_cut <- prune.tree(tree_estimation, best = 4)

plot(tree_cut,type = "uniform")
text(tree_cut,pretty=0, pos='2.8',cex=0.4)
title("Árbol final para predicción de churn",line=0.5)
```



Anexamos las predicciones del árbol en la base independiente, tanto el score como la predicción categórica.

```
tree_score_predict <- predict(tree_cut,newdata=data_validation)

tree_class_predict <- predict(tree_cut,newdata=data_validation,type="class")

# unir con base de predicciones
A <- data.frame(A, tree_score_predict[,2], tree_class_predict)
names(A)[4:5] <- c("tree_score","tree_predict")
```

5.3 Random Forest Ahora corremos un Random Forest. Primero exploramos cuál es la B para la que ya no ganamos mucho más en poder predictivo. Hacemos pruebas con diferentes cantidades

de árboles, 100,200,300, 500, 700 y 800, y comparamos los errores de predicción `out of bag`.

```
# eficientar el Random Forest corriendolo en los nodos disponibles del equipo
cl<- detectCores() %>% makeCluster()
cl
```

```
socket cluster with 8 nodes on host 'localhost'
```

```
# vector con el número de los arboles
trees <- c()
# vector con el prediction error de cada forest
error <- c()

# Estimation. Loop que estima un random forest para distintos numeros de árboles

k <- 1 # iterador
for (i in c(100,200,300,500,700,800)){

  rf<-ranger(churn~.,
             data = data_training_a[-1],
             num.trees = i,
             mtry = (ncol(data_training_a)-2) %>% sqrt() %>% floor(),
             min.node.size = 1,
             splitrule = "gini",
             classification = T,
             verbose = F,
             status.variable.name = 0
             )

  trees[k] <- rf$num.trees
  error[k] <- rf$prediction.error

  rf <- NULL
  k <- k + 1
}

stopCluster(cl)

rf <- data.frame(trees,error)

kable(rf, col.names = c("Trees","Error"), booktabs=T,
```

```
      caption="Error en random forest")%>%
  kable_styling(position = "center",
                latex_options="HOLD_position")
```

Table 5. Error en random forest

Trees	Error
100	0.4028668
200	0.3913753
300	0.3880186
500	0.3846619
700	0.3786138
800	0.3783718

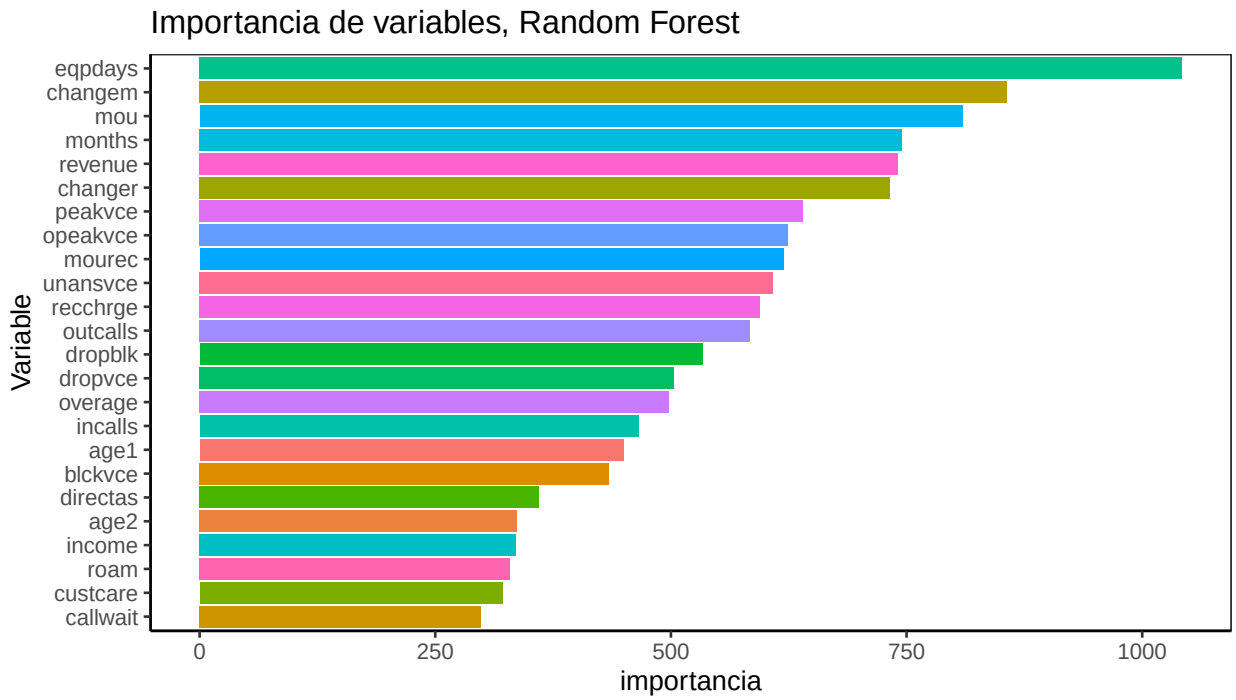
Parece ser que con una $B = 200$, el error de predicción ya no se reduce mucho. Cabe mencionar que el error in sample era mucho menor con las bases resultantes de oversampling, pero lo opuesto pasaba en OOS. Con este parámetro definido, graficamos la importancia de las variables correspondiente.

```
# vuelvo a correr el mejor random forest
cl<- detectCores() %>% makeCluster()
cl
```

```
socket cluster with 8 nodes on host 'localhost'
```

```
best_rf <- ranger(churn~.,
              data = data_training_a[,-1],
              num.trees = 200,
              mtry = (ncol(data_training_a)-2) %>% sqrt() %>% floor(),
              importance = "impurity",
              classification = T)

stopCluster(cl)

# Grafica
df_imp <- data.frame(names=(importance(best_rf) %>%
               names()),importance=(importance(best_rf)))


ggplot(df_imp[df_imp$importance>250,],
       aes(x=reorder(names,importance),y=importance, fill= names)) +
  geom_bar(stat="identity") +
```

```
  xlab("Variable")+
  ylab("importancia")+
  ggtitle('Importancia de variables, Random Forest')+
  coord_flip()+
  theme(axis.text.y = element_text(size = 9))+
  theme_bw()+
  theme(axis.line = element_line(colour = "black"),
    panel.grid.major = element_blank(),
    panel.grid.minor = element_blank(),
    panel.background = element_blank(),
    legend.position = 'none')
```



Ahora generamos las predicciones OOS para el random forest y las guardamos en la base con las demás predicciones, tanto categóricas como continuas. Para estas últimas es necesario correr de nuevo el random forest, pero con `probability = T`.

```
pred_rf<-predict(best_rf, data = (data_validation[,-1]), type = "response")

# añadir prediccion al data frame con las otras predicciones

A <- data.frame(A,pred_rf$predictions)
names(A)[6] <- "rf_predict"
```

```
# vuelvo a correr el mejor random forest, con probability=T
cl<- detectCores() %>% makeCluster()
cl
```

```
socket cluster with 8 nodes on host 'localhost'
```

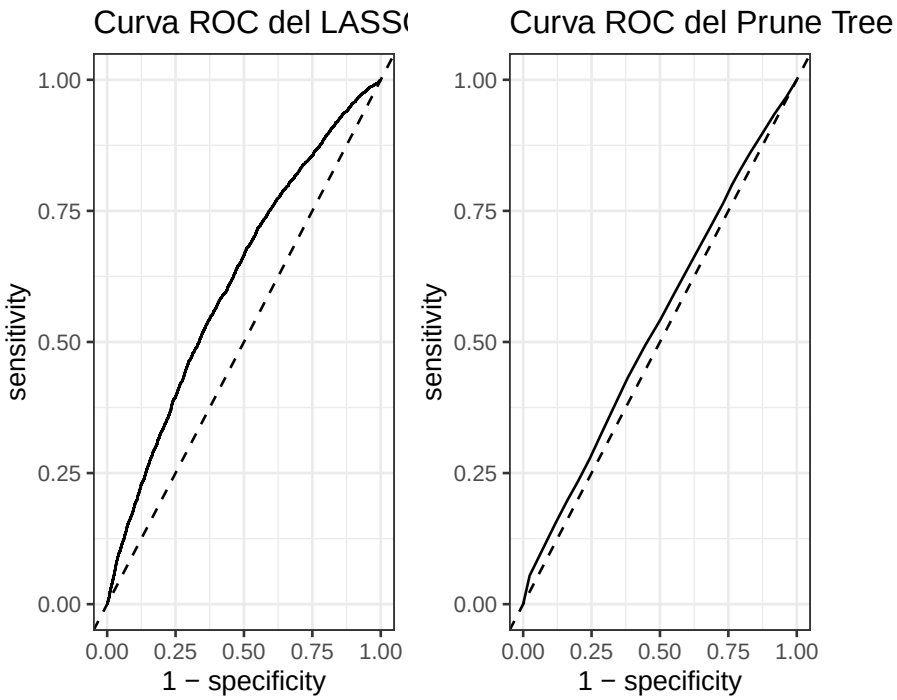
```
best_rf2 <- ranger(churn~.,
             data = data_training_a[,-1],
             num.trees = 200,
             mtry = (ncol(data_training_a)-2) %>% sqrt() %>% floor(),
             importance = "impurity",
             classification = T,
             probability = T
             )
stopCluster(cl)

# prediccion continua de churn==1
# eval$pred_rf_cont <- predict(best_rf2, data=data_validation)$predictions[,1]
rf_score <- predict(best_rf2, data=data_validation[,-1])

# guardamos las predicciones
A <- data.frame(A, rf_score$predictions[,1])
names(A)[7] <- "rf_score"
A <- A[c(1,2,3,4,5,7,6)]
```

6. Evaluación

6.1 Curvas ROC Para comparar el desempeño de los modelos, generamos gráficas de las curvas ROC para los tres modelos.

```
# Lasso
roc1 <- roc_curve(data = A,
          truth = as.factor(churn_validation),
          lasso_score,
          event_level = "second") %>%
  ggplot(aes(x = 1-specificity, y = sensitivity)) +
  geom_abline(slope = 1, intercept = 0, linetype = 'dashed') +
  ggtitle("Curva ROC del LASSO")+
  geom_path() +
  theme_bw()
```

```r
# Trees
roc2 <- roc_curve(data = A,
           truth = as.factor(churn_validation),
           tree_score,
           event_level = "second") %>%
  ggplot(aes(x = 1-specificity, y = sensitivity))+
  geom_abline(slope = 1, intercept = 0, linetype = 'dashed')+
  ggtitle("Curva ROC del Prune Tree")+
  geom_path()+
  theme_bw()

# Random Forest
roc3 <- roc_curve(data = A,
          truth = factor(churn_validation),
          rf_score,
          event_level = "second") %>%
  ggplot(aes(x = 1-specificity, y = sensitivity)) +
  geom_abline(slope = 1, intercept = 0, linetype = 'dashed') +
  ggtitle("Curva ROC del Random Forest")+
  geom_path() +
  theme_bw()

grid.arrange(roc1,roc2, ncol=3)
```



Dado que la línea punteada es un modelo nulo (aleatorio), entre más rápido gane sensibilidad sin perder especificidad, el modelo será mejor. Gráficamente, buscamos una curva cóncava y pegada al eje y del lado derecho y al eje x por arriba. En este sentido, el Random Forest parece ser el mejor modelo.

También generamos una tabla con el AUC ROC.

```
# roc_auc(eval, factor(validation), pred_rf, event_level = "second")
lasso_auc <- roc_auc(data = A,
                     truth = factor(churn_validation),
                     lasso_score,
                     event_level = "second")

tree_auc <- roc_auc(data = A,
                    truth = factor(churn_validation),
                    tree_score,
                    event_level = "second")

rf_auc <- roc_auc(data = A,
                  truth = factor(churn_validation),
                  rf_score,
                  event_level = "second")

aucs <- (full_join(lasso_auc, tree_auc) %>%
```

```
  full_join(rf_auc))[,3] %>%
  rename(AUC = .estimate) %>% as.data.frame()
model_names <- c('CV-LASSO','Pruned Tree','Random Forest')
aucs<-data.frame(model_names, aucs$AUC)


kable(aucs, col.names = c("Modelos","AUC"), booktabs=T,
      caption="Comparación de modelos")%>%
  kable_styling(position = "center",
                latex_options="HOLD_position")
```

Table 6. Comparación de modelos

Modelos	AUC
CV-LASSO	0.6157044
Pruned Tree	0.5340553
Random Forest	0.6629787

Confirmamos que el mejor modelo es el Random Forest, seguido del CV-LASSO.

6.2 Matrices de confusión Escogemos un punto de corte para generar predicciones categóricas para el LASSO basado en la Curva ROC.

```
summary(A$lasso_score)
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
0.06522 0.42516 0.48946 0.49042 0.55234 0.99391
```

```
# Loop que tome diferentes puntos de corte y mida especificidad y sensibilidad

sens <- NULL
spec <- NULL
corte <- NULL
it <- 1 # iterador

for (x in seq(0.45,0.55,0.01)){

  A$lasso_1 <- as.numeric(A$lasso_score > x)

  confm <- caret::confusionMatrix(as.factor(A$lasso_1),
                          as.factor(A$churn_validation))
```

Punto de Corte	Sensibilidad	Especificidad
0.45	0.375	0.774
0.46	0.417	0.741
0.47	0.461	0.702
0.48	0.503	0.662
0.49	0.549	0.613
0.50	0.594	0.574
0.51	0.637	0.533
0.52	0.679	0.486
0.53	0.716	0.441
0.54	0.750	0.395
0.55	0.780	0.353

```
  sens[it] <- confm$byClass['Sensitivity'] %>% as.numeric() %>% round(3)

  spec[it] <- confm$byClass['Specificity'] %>% as.numeric() %>% round(3)

  corte[it] <- x
  confm <- NULL
  it <- it + 1
}

x <- data.frame(corte,sens,spec)

  kable(x, booktabs=T, align = 'c',
        col.names = c("Punto de Corte","Sensibilidad", "Especificidad"))%>%
  kable_styling(position = "center")
```

```
# punto de corte elegido
A$lasso_predict <- as.numeric(A$lasso_score > 0.52)
```

Siguiendo este criterio, y premiando sensibilidad frente a especificidad (por razones que se explican más abajo), elegimos 0.52 como punto de corte.

La curva ROC representa la sensibilidad y especificidad para ciertos puntos de corte. Para elegir un punto de corte óptimo, calculamos la sensibilidad y especificidad de distintos puntos de corte, buscando el óptimo en la gráfica, el cual está alrededor del 0.6 de sensibilidad y 0.5 de especificidad.

Ahora generamos las matrices de confusión para cada modelo.

```
# LASSO
matconf_lasso <- caret::confusionMatrix(as.factor(A$lasso_predict),
                                        as.factor(A$churn_validation))

# TREE
matconf_tree <- caret::confusionMatrix(as.factor(A$tree_predict),
                                       as.factor(A$churn_validation))

# RANDOM FOREST
matconf_rf<-caret::confusionMatrix(as.factor(A$rf_predict),
                                   as.factor(A$churn_validation))
```

A partir de las matrices de confusión es posible calcular 5 métricas de éxito:

- Sensibilidad
- Especificidad
- Positivos Acertados
- Negativos Acertados
- Accuracy

A continuación se comparan dichas métricas en una tabla:

```
exito<- as.data.frame(rbind(matconf_lasso$byClass[1:5],
      matconf_tree$byClass[1:5],
      matconf_rf$byClass[1:5]))

Modelo <- c("Lasso","Tree","Random Forest")
predictive_power <- cbind(Modelo,exito)
rm(exito, Modelo)

kable(predictive_power,booktabs=T, align = 'c',
      col.names = c("Modelo",
                    "Sensibilidad",
                    "Especificidad",
                    "Positivos acertados",
                    "Negativos acertados",
                    "Accuracy"),digits = 3)%>%
  kable_styling(position = "center",
                latex_options="HOLD_position")
```

Modelo	Sensibilidad	Especificidad	Positivos acertados	Negativos acertados	Accuracy
Lasso	0.679	0.486	0.767	0.378	0.767
Tree	0.554	0.494	0.731	0.308	0.731
Random Forest	0.592	0.650	0.808	0.390	0.808

El modelo que arroja las mejores métricas de éxito es el *Random Forest*. En este caso nos interesa una sensibilidad relativamente alta, ya que nos indica cuál es la probabilidad de que encontremos a clientes que efectivamente se van ($churn = 1$).

Adicionalmente, nos interesa ver el porcentaje de positivos acertados, lo que nos dice cuántos clientes que predecimos que se vayan efectivamente se irán.

En este caso el error más pernicioso sería el error tipo II (falso negativo), o sea que el modelo nos diga que un cliente no se va cuando en realidad si se irá. El modelo con menos falsos negativos es el *Random Forest* .

6.3 Lift Table Ahora construimos una lift table. Esto es, para 20 grupos del score predecido, genera 1) El promedio de las predicciones, 2) el promedio del churn observado. Con ella, nos interesa averiguar si existe monotonía.

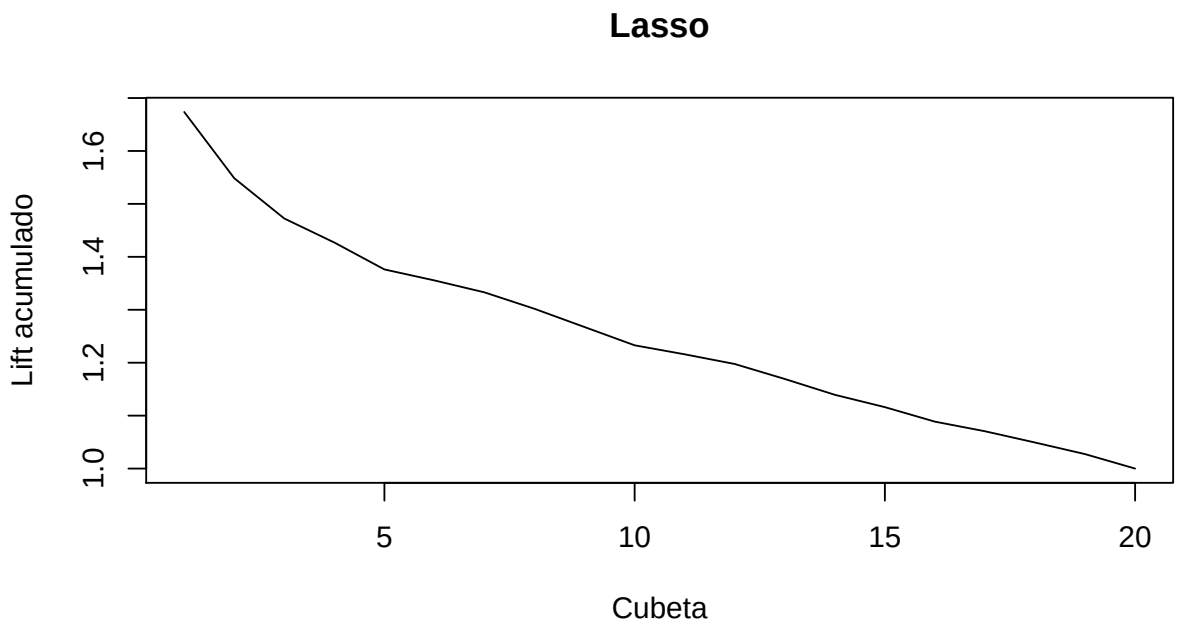
```
# tablas
tabla_lasso<-liftTable( A$lasso_score, A$churn_validation, resolution = 1/20)
tabla_tree<-liftTable( A$tree_score, A$churn_validation, resolution = 1/20)
tabla_rf<-liftTable( A$rf_score, A$churn_validation, resolution = 1/20)
tabla_lift<-data.frame(tabla_lasso$Percentile,
                       tabla_lasso$expectedIncidence,
                    tabla_lasso$trueIncidence,
                    tabla_tree$trueIncidence,
                    tabla_rf$trueIncidence)
kable(tabla_lift, booktabs=T, col.names = c("Percentil",
                                "Valor esperado",
                                "Promedio Lasso",
                                "Promedio Tree",
                                "Promedio RF"), caption="Lift")%>%
  kable_styling(position = "center",
                latex_options="HOLD_position")
```

Table 7. Lift

Percentil	Valor esperado	Promedio Lasso	Promedio Tree	Promedio RF
5	0.2867699	0.4802817	0.4366197	0.5563380
10	0.2867699	0.4433498	0.3511612	0.4940183
15	0.2867699	0.4218677	0.3383388	0.4763022
20	0.2867699	0.4088670	0.3223082	0.4595355
25	0.2867699	0.3944257	0.3150338	0.4481982
30	0.2867699	0.3882243	0.3140981	0.4288060
35	0.2867699	0.3818621	0.3130907	0.4144380
40	0.2867699	0.3731527	0.3115764	0.4044687
45	0.2867699	0.3633094	0.3079449	0.3922427
50	0.2867699	0.3531316	0.3030260	0.3812808
55	0.2867699	0.3485605	0.3014715	0.3705694
60	0.2867699	0.3433028	0.2993197	0.3612479
65	0.2867699	0.3353183	0.2986141	0.3530749
70	0.2867699	0.3267317	0.2977782	0.3437217
75	0.2867699	0.3200713	0.2967064	0.3352726
80	0.2867699	0.3121042	0.2957424	0.3255630
85	0.2867699	0.3069217	0.2943368	0.3172711
90	0.2867699	0.3008054	0.2908750	0.3081554
95	0.2867699	0.2946144	0.2883177	0.2981702
100	0.2867699	0.2867699	0.2867699	0.2867699

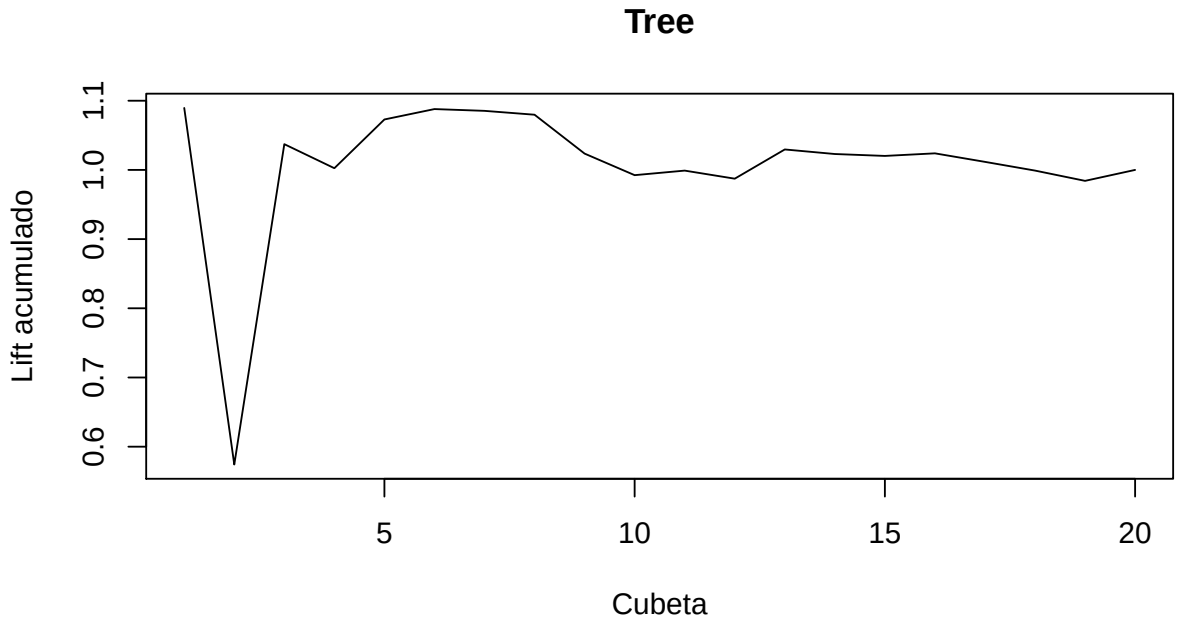
```
# Esto es para generar las graficas
tabla_lift <- function(variable_objetivo, score_prediccion, groups) {
  if(is.factor(variable_objetivo)) variable_objetivo <- variable_objetivo %>%
    as.character() %>% as.integer()
  if(is.factor(score_prediccion)) score_prediccion <- score_prediccion %>%
    as.character() %>% as.integer()

  helper = data.frame(cbind(variable_objetivo, score_prediccion))

  helper[,"bucket"] = ntile(-helper[,"score_prediccion"], groups)

  gaintable = helper %>% group_by(bucket)  %>%
    summarise_at(vars(variable_objetivo), funs(total = n(),
    totalresp=sum(., na.rm = TRUE))) %>%
```

```
    mutate(Cumresp = cumsum(totalresp),
    Gain=Cumresp/sum(totalresp)*100,
    Cumlift=Gain/(bucket*(100/groups)))


  return(gaintable)
}

# Grafico el lift
lift_lasso<-tabla_lift(A$churn_validation , A$lasso_score, groups = 20)
graphics::plot(lift_lasso$bucket,
               lift_lasso$Cumlift,
               type="l",
               ylab="Lift acumulado",
               xlab="Cubeta",
               main = "Lasso")
```

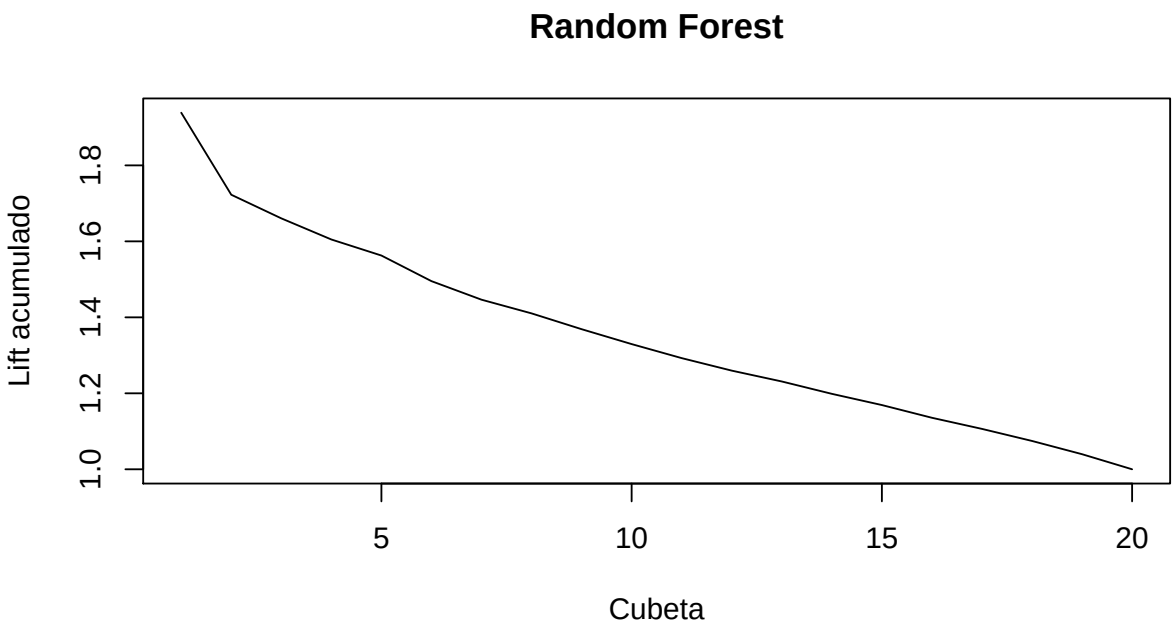



```
lift_tree<-tabla_lift(A$churn_validation ,
                      A$tree_score,
                      groups = 20)
graphics::plot(lift_tree$bucket,
               lift_tree$Cumlift,
```

```
                type="l",
                ylab="Lift acumulado",
                xlab="Cubeta",
                main = "Tree")
```




```
lift_rf<-tabla_lift(A$churn_validation , A$rf_score, groups = 20)

graphics::plot(lift_rf$bucket, lift_rf$Cumlift, type="l",
               ylab="Lift acumulado", xlab="Cubeta",main = "Random Forest")
```



De este análisis es posible concluir que existe monotonía en los modelos LASSO y Random Forest, no así en el del árbol. La monotocidad implica que al ordenar el score de cada observación de mayor a menor, tendrémos una mayor cantidad de observaciones con etiqueta de churn igual a 1 al principio de la tabla. Si dividimos las observaciones ordenadas en grupos, la monotonicidad implicaría que el primer grupo es el que más contiene 1s en churn, el segundo grupo sería el segundo que más contiene 1s y así sucesivamente hasta llegar al último grupo. Esta propiedad se refleja visualmente en las gráficas, las del Random Forest y LASSO, efectivamente cumplen con la función de monotonicidad, a diferentcia de la gráfica del árbol. El mejor algoritmo de nuestro análisis, random forest, es monotónico.

7. Conclusión

La estrategia general que se sugeriría implementar a partir del modelo es precisamente que la empresa focalice sus esfuerzos en la permanencia, retención y cuidado de sus clientes, toda vez que es muy común que las empresas dirijan toda su atención y centren su éxito en planes de crecimiento, es decir, en conseguir nuevos clientes, dejando de lado el monitoreo de los que ya tiene. Esta visión nublada de algunas compañías, puede incluso traducirse, en el largo plazo, en más gastos, ya que destina recursos de manera ineficiente. Las decisiones basadas en datos siempre tendrán un valor agregado, ya que serán las más acertadas, toda vez que que responden de manera precisa a la satisfacción de los clientes y al beneficio de cada compañía.

En este caso, el modelo detectó como principal determinante del abandono la variable eqpdays, que es el número de días que el cliente permanece con el equipo de teléfono actual, seguida del porcentaje de minutos de uso -changem- y la media mensual de minutos de uso -mou-.

En consecuencia, a partir de los resultados arrojados por este modelo, la compañía debe plantarse una estrategía para lograr que el cliente permanezca el menor tiempo posible con su equipo telefónico, esto es, incentivar las renovaciones de teléfono, toda vez que es factible que los clientes que tengan mayor tiempo con su equipo, se vean tentados a buscar una compañía diferente cuando al fin toman la decisión de renovar. Por tanto, la empresa debe crear estrategias tal como un paquete que implique la renovación automática del equipo, anticiparse a aquellos al abandono identificando a los clientes que llevan más tiempo con su equipo para ofrecerles ofertas al renovar o al continuar con el servicio, la cración de un sisitema de recomedación personalizado para los clientes, entre otras.

De igual manera el abandono de aquellos clientes que utilizan más el servicio medido en minutos, puede deberse a que, precisamente por su intensidad en el uso, estén en constante búsqueda de mejores servicios en términos de calidad y precio, por lo que al haber identificado este motivante en el abandono a partir del modelo, se propone que la empresa fortalezca sus acciones de fidelización, para lo cual es clave valorar la experiencia del cliente, tomar en cuenta sus inquietudes, para finalmente tomar acciones con base en sus sugerencias; ya sea para mejorar la calidad de un producto, el precio, la atención al cliente o el mejoramiento del servicio.

El potencial de este tipo de modelos radica en que pueden analizar datos a un nivel que ningún ser humano podría lograr, con lo cual puede ampliar abismalmente el campo de acción de cualquier estrategia y con insumos de calidad; a través del aprendizaje automatizado, es posible perfeccionar el modelo para que cada vez propocione resultados más precisos, y con ello poder tener cada vez mejores estrategias para reducir la tasa de abandono y afianzar la cartera de clientes de la empresa.

EXTRA: XGBoosting

```
# Preparar la base de entrenamiento
sparse_train <- sparse.model.matrix(~.+0, data = data_training_a[,-c(1,2)])
label_train <- data_training_a[,2]
dtrain <- xgb.DMatrix(sparse_train, label = label_train)
# Label es el target

# Preparar la base de validación
data_validationn <- data[-training_rows,]
sparse_test <- sparse.model.matrix(~.+0, data = data_validation[,-c(1)])
label_test <- data_validationn[,2]
dtest <- xgb.DMatrix(sparse_test, label = label_test)
watchlist <- list(train = dtrain, eval = dtest)
# Para evaluar el performance del modelo

# Entrenamiento del modelo
param <- list(max_depth = 6, learning_rate = 0.06,
```

```
              objective = "binary:logistic",
              eval_metric = "auc", subsample = 0.85, colsample_bytree = 0.7)
xgb_model <- xgb.train(params = param, dtrain,
                       early_stopping_rounds =  10,
                       nrounds = 100,
                  watchlist)

# Predicción
xgb_pred <- predict(xgb_model, sparse_test)
A$XGB_score <- xgb_pred
```

Se muestran las evaluaciones del modelo, tanto in sample como out of sample, para las primeras y últimas iteraciones.

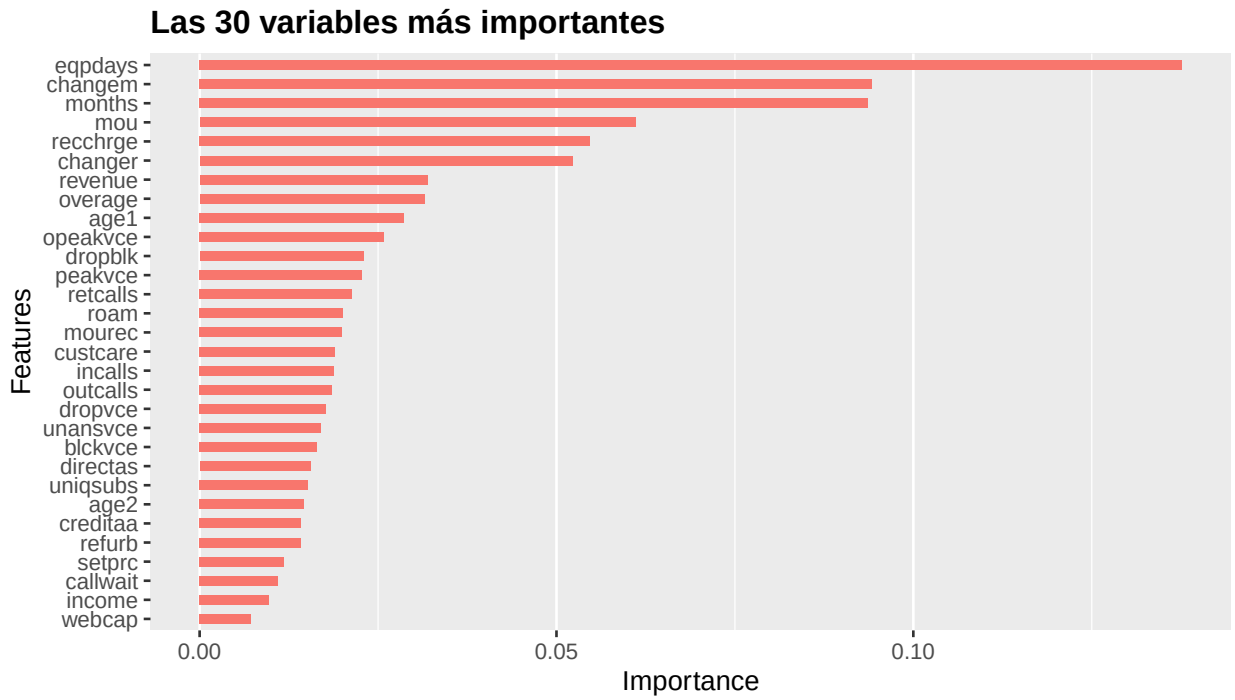
```
kable(xgb_model$evaluation_log[c(1:3,98:100)],
      col.names = c('Iteración', 'In sample AUC','Out of sample AUC'))
```

Iteración	In sample AUC	Out of sample AUC
1	0.644690	0.620477
2	0.670057	0.641358
3	0.674264	0.647677
98	0.780996	0.675217
99	0.781871	0.675722
100	0.782704	0.675847

Cabe destacar que la mejor iteración del modelo fue la 100, y esta tiene un mejor AUC-ROC que cualquiera de los modelos presentados anteriormente.

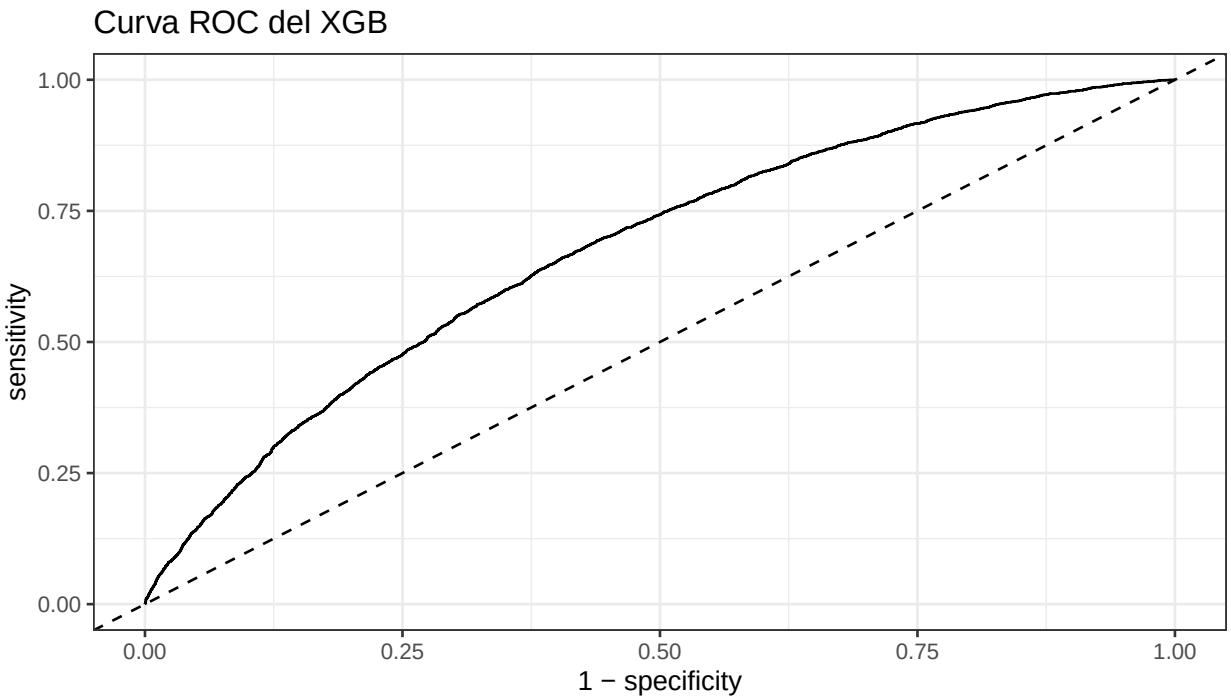
También se presenta la matriz de importancia.

```
# Matriz de importancia
names <- dimnames(data.matrix(data_training_a[,-c(1,2)]))[[2]]
importance_matrix <- xgb.importance(names, model = xgb_model)

# Gráfico
gg <- xgb.ggplot.importance(importance_matrix[1:30], n_clusters = 1)
gg + ggtitle("Las 30 variables más importantes") + theme(legend.position = "none")
```



Se presenta también la curva ROC

```
roc_curve(data = A,
          truth = as.factor(churn_validation),
          XGB_score,
          event_level = "second") %>%
  ggplot(aes(x = 1-specificity, y = sensitivity)) +
  geom_abline(slope = 1, intercept = 0, linetype = 'dashed') +
  ggtitle("Curva ROC del XGB")+
  geom_path() +
  theme_bw()
```



Finalmente, se presenta una tabla con los valores de la sensibilidad y especificidad para distintos threshold de p.

```
sens <- NULL
spec <- NULL
corte <- NULL
it <- 1 # iterador

for (x in seq(0.45,0.55,0.01)){

  A$XGB_pred <- as.numeric(A$XGB_score > x)

  confm <- caret::confusionMatrix(as.factor(A$XGB_pred),
                          as.factor(A$churn_validation))

  sens[it] <- confm$byClass['Sensitivity'] %>% as.numeric() %>% round(3)

  spec[it] <- confm$byClass['Specificity'] %>% as.numeric() %>% round(3)

  corte[it] <- x
  confm <- NULL
  it <- it + 1
}
```

Punto de Corte	Sensibilidad	Especificidad
0.45	0.454	0.781
0.46	0.481	0.759
0.47	0.510	0.734
0.48	0.539	0.710
0.49	0.568	0.686
0.50	0.598	0.656
0.51	0.628	0.621
0.52	0.658	0.589
0.53	0.687	0.558
0.54	0.713	0.527
0.55	0.738	0.492

```
x <- data.frame(corte,sens,spec)

  kable(x, booktabs=T, align = 'c',
        col.names = c("Punto de Corte","Sensibilidad", "Especificidad"))%>%
  kable_styling(position = "center")
```

Nótese cómo para un valor de sensibilidad con un corte en 0.54, con este modelo se puede mantener una especificidad decente. Recordemos que lo que más nos preocupa es prevenir el abandono de los clientes, por lo que es válido elegir una sensibilidad alta al costo de la especificidad.