Instituto Tecnológico Autónomo de México

Economía Computacional

Tarea 2

Equipo 7

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

Contexto . . . 2
Datos . . . 3
1. Qué variables tienen missing values? Toma alguna decisión con los missing values. Justifica tu respuesta . . . 3
Análisis de missing values general . . . 3
Analisis por variable . . . 6
2. Tabula la distribución de la variable `churn`. Muestra la frecuencia absoluta y relativa. Crees que se debe hacer oversampling/undersamping? . . . 11
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) . . . 13
Model estimation . . . 16
4 (2 pts). Estima un cross validated LASSO. Muestra el la gráfica de CV Binomial Deviance vs Complejidad . . . 16
5. Grafica el Lasso de los coeficientes vs la complejidad del modelo. . . . 17
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 . . . 19
7. Genera un data frame (usando el validation set) que tenga: `customer`, `churn` y las predicciones del LASSO. . . . 21
8. Estima ahora tree. Usa `mindev = 0.05, mincut = 1000` Cuántos nodos terminales salen? Muestra el summary del árbol . . . 21
9. Grafica el árbol resultante . . . 22
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? . . . 23
11. Gráfica el árbol final. (Tip: Checa `prune.tree`) . . . 24
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 . . . 25
13 (4pts). Corre un Random Forest ahora. Cuál es la B para la que ya no ganamos mucho más en poder predictivo? . . . 25
14. Escoge un random forest para hacer las predicciones. Grafica la importancia de las variables. Interpreta . . . 27
15. Genera las predicciones OOS para el random forest. Guardalas en la misma data.frame que los otros modelos . . . 28

16 (2pts). Corre el mismo forest pero ahora con `probability = T`. Esto generará predicciones númericas en lugar de categóricas. Genera las predicciones continuas y guardalas en el mismo data frame 28
17 (4 pts). Genera graficas de las curvas ROC para los tres modelos. Cual parece ser mejor? 29
18. Genera una tabla con el AUC ROC. Cuál es el mejor modelo ? 31
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? 32
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) 36
EXTRA: XGB 40

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

Carguemos los datos

```
load("Cell2Cell.Rdata") %>% as.data.frame()
```

```
          .
1 cell2cell
```

```
# file path de archivo path_data <-
# file.path('C:/Users/rsf94/Documents/economia_computacional/tarea_2',
# 'Cell2Cell.RData')

# renombrar como data
data <- as.data.frame(cell2cell)
rm(cell2cell)
```

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

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.

```
# columnas 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)
```

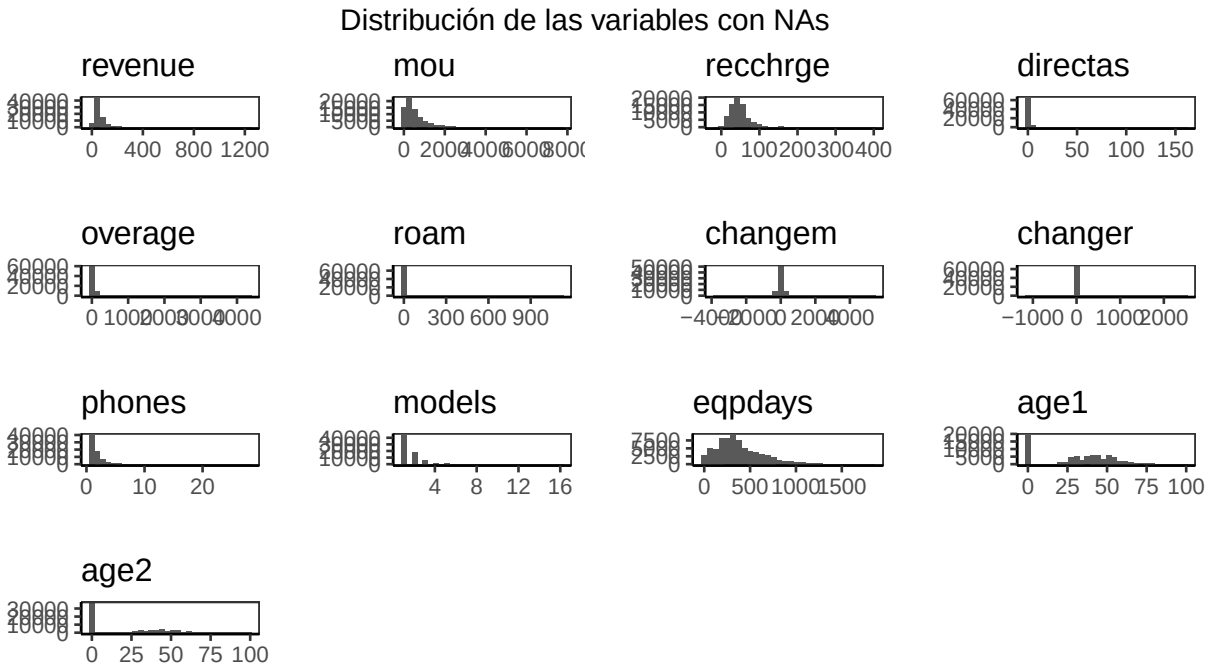
```
    revenue             mou            recchrge         directas
 Min.   :  -6.168   Min.   :   0.0   Min.   :-11.29   Min.   :  0.0000
 1st Qu.:  33.642   1st Qu.: 158.2   1st Qu.: 30.00   1st Qu.:  0.0000
 Median :  48.530   Median : 366.0   Median : 44.99   Median :  0.2475
 Mean   :  58.853   Mean   : 525.7   Mean   : 46.88   Mean   :  0.8940
 3rd Qu.:  71.030   3rd Qu.: 721.8   3rd Qu.: 59.99   3rd Qu.:  0.9900
 Max.   :1223.380   Max.   :7667.8   Max.   :399.99   Max.   :159.3900
 NA's   :216        NA's   :216      NA's   :216      NA's   :216
    overage             roam             changem            changer
 Min.   :   0.00   Min.   :   0.0000   Min.   :-3875.00   Min.   :-1107.740
 1st Qu.:   0.00   1st Qu.:   0.0000   1st Qu.:  -83.00   1st Qu.:   -7.107
 Median :   2.50   Median :   0.0000   Median :   -5.00   Median :   -0.295
 Mean   :  40.09   Mean   :   1.2211   Mean   :  -10.85   Mean   :   -1.206
 3rd Qu.:  40.75   3rd Qu.:   0.2575   3rd Qu.:   65.75   3rd Qu.:    1.605
 Max.   :4320.75   Max.   :1112.4480   Max.   : 5192.25   Max.   : 2483.482
 NA's   :216       NA's   :216         NA's   :502        NA's   :502
     phones            models          eqpdays            age1
```

```
 Min.   : 1.000   Min.   : 1.000   Min.   :  -5.0   Min.   : 0.00
 1st Qu.: 1.000   1st Qu.: 1.000   1st Qu.: 204.0   1st Qu.: 0.00
 Median : 1.000   Median : 1.000   Median : 330.0   Median :36.00
 Mean   : 1.809   Mean   : 1.562   Mean   : 380.3   Mean   :31.38
 3rd Qu.: 2.000   3rd Qu.: 2.000   3rd Qu.: 515.0   3rd Qu.:48.00
 Max.   :28.000   Max.   :16.000   Max.   :1823.0   Max.   :99.00
 NA's   :1        NA's   :1        NA's   :1        NA's   :1244
      age2
 Min.   : 0.00
 1st Qu.: 0.00
 Median : 0.00
 Mean   :21.16
 3rd Qu.:42.00
 Max.   :99.00
 NA's   :1244
```

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.

```
# creo una función para hacer los histogramas pertinentes

myhist <- function(yvar) {
    ggplot(df_nas, aes_(x = as.name(yvar))) + geom_histogram() +
        ggtitle(paste0(as.name(yvar))) + xlab("") + ylab("") +
        geom_rangeframe() + theme_bw() + theme(axis.line = element_line(colour = "black"),
        panel.grid.major = element_blank(), panel.grid.minor = element_blank(),
        panel.background = element_blank())
}
hists <- df_nas %>% names() %>% lapply(myhist)

# grafico las variables
grid.arrange(grobs = hists, ncol = 4, top = textGrob("Distribución de las variables con NAs"))
```



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
```

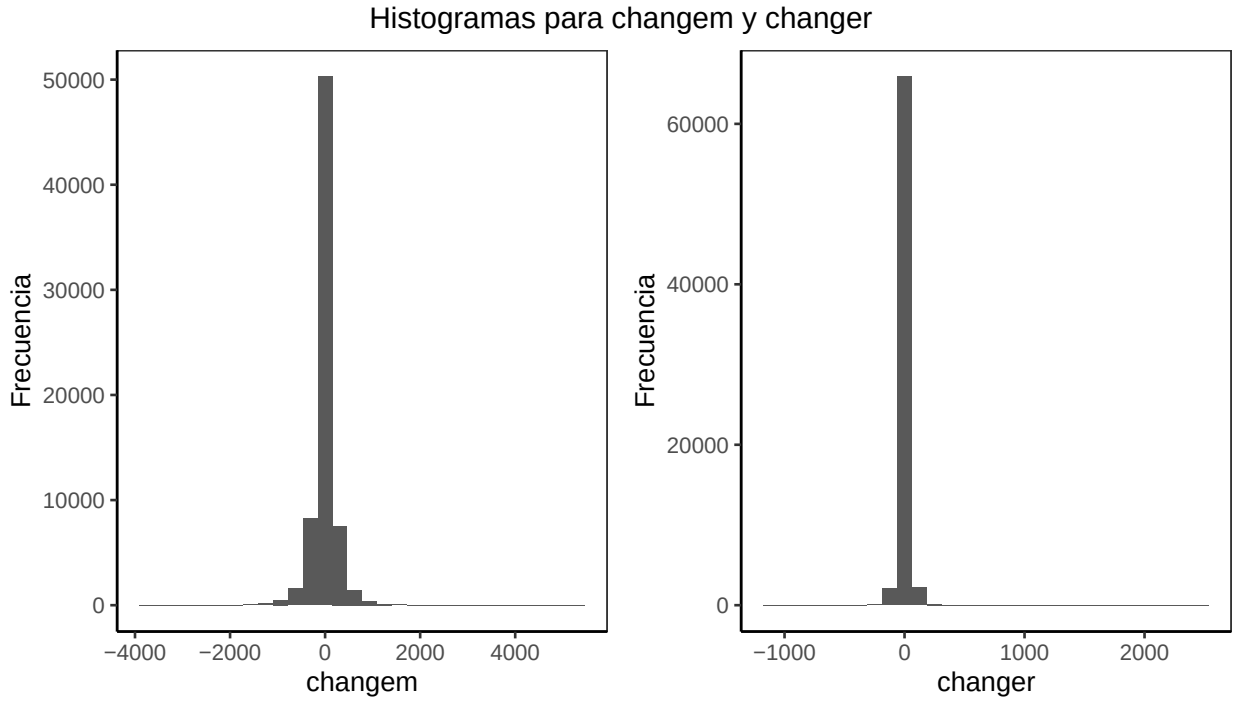
phones, models y eqpdays

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 =
    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 =
    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(direct
    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
```

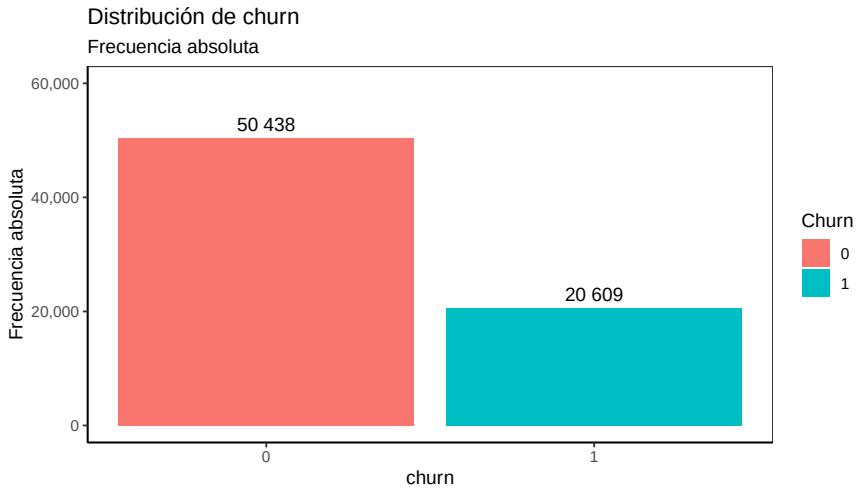
2. Tabula la distribución de la variable `churn`. Muestra la frecuencia absoluta y relativa. Crees que se debe hacer oversampling/undersamping?

```
# 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")
```

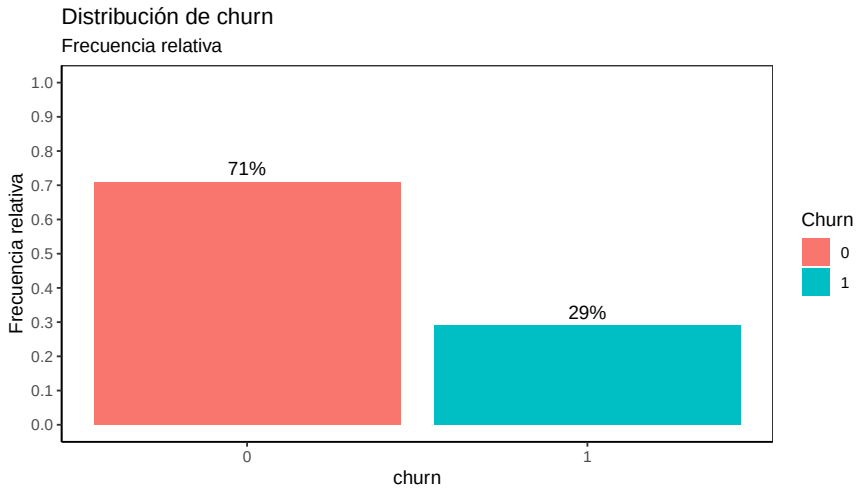
Churn	Cantidad	%
0	50438	0.71
1	20609	0.29

```
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.

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)

Primero dividimos la base.

```
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, col.names = c("Churn", "Frecuencia absoluta",
    "Frecuencia relativa")) %>% kable_styling(position = "center",
    latex_options = "HOLD_position")
```

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)

undersampling <- rbind((data_training %>% filter(churn == 1)),
    undersampling_c0)
```

A. Undersampling

```
# Segundo intento, sugiere hacer Synthetic Minority Oversampling Technique (SMOTE) librería sm
# 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 # desired times of synthetic 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
tabulado <- oversampled %>% group_by(churn) %>%
    dplyr::summarise(frecuencia_absoluta=n()) %>%
    mutate(frecuencia_relativa = frecuencia_absoluta/sum(frecuencia_absoluta))
kable(tabulado, col.names = c("Churn","Frecuencia absoluta","Frecuencia relativa"))%>%
  kable_styling(position = "center",
                latex_options="HOLD_position")
```

B. oversampling

```
# 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 =
    mutate(frecuencia_relativa = frecuencia_absoluta/sum(frecuencia_absoluta))

kable(tabulado, 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
```

Model estimation

Pondremos a competir 3 modelos:

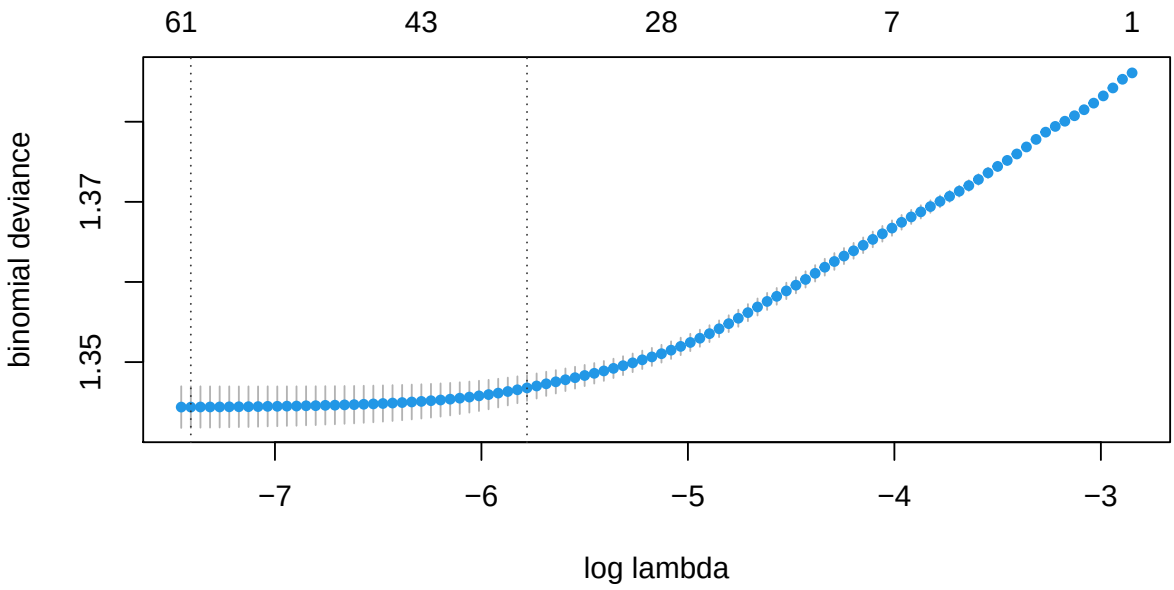
1. Cross-Validated LASSO-logit
2. Prune Trees
3. Random Forest

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

```
# X <- sparse.model.matrix(~.+0, data =
# data_training[,-c(1,2)]) # Transformar a sparse matrix la
# información relevante

# cv_lasso <- cv.gamlr(x = X, y = data_training$churn, family
# = 'binomial', nfold = 5, verb = TRUE) # Estimar el CV LASSO

# par(mar=c(5,4,4,2) + 0.1) plot(cv_lasso) # Gráfico


## A

# 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)
```

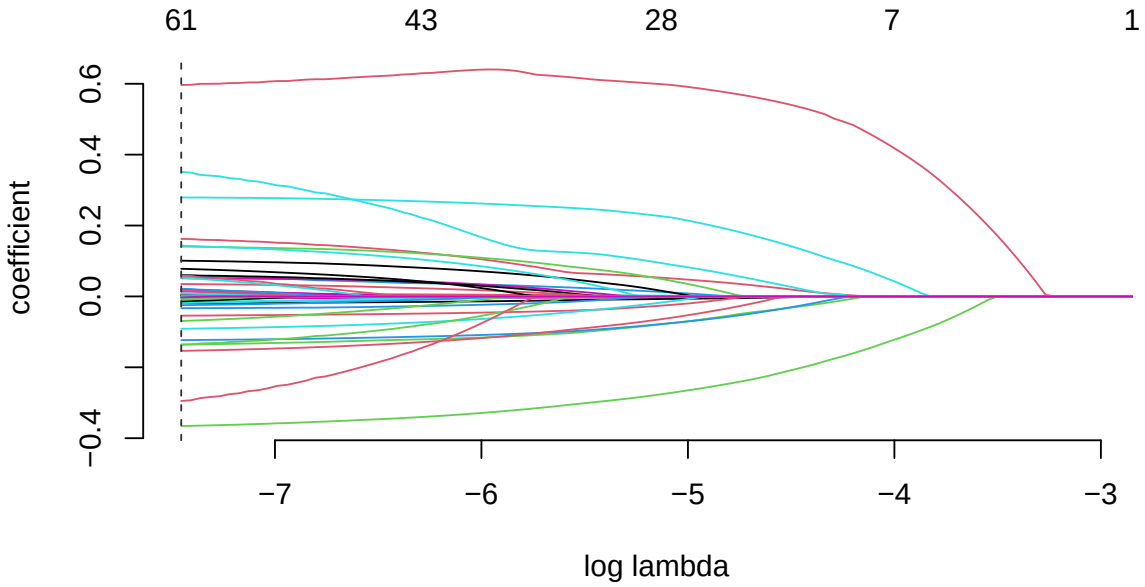
```
fold 1,2,3,4,5,done.
```

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



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

```
# plot(cv_lasso$gamlr, select=FALSE)
plot(cvlasso_a$gamlr)
```


61
43
28
7
1
coefficient
0.6
0.4
0.2
0.0
−0.4
−7
−6
−5
−4
−3
log lambda

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

```
# coef(cv_lasso, select='min') # Coeficientes de CV LASSO
coef(cvlasso_a, select = "min", k = 2, corrected = TRUE)  #a
```

```
67 x 1 sparse Matrix of class "dgCMatrix"
                   seg99
intercept -1.137975e-01
revenue    6.302478e-04
mou       -1.986502e-04
recchrge  -1.899325e-03
directas   2.986691e-03
overage    9.849538e-04
roam       5.470179e-03
changem   -5.759147e-04
changer    2.804547e-03
dropvce    6.503516e-03
blckvce    .
unansvce   3.104458e-04
custcare  -5.321727e-03
threeway  -1.451446e-02
mourec     1.798823e-04
outcalls   9.242483e-05
incalls   -1.369377e-03
peakvce   -6.702399e-04
opeakvce  -1.576663e-04
dropblk    1.483887e-03
callfwdv  -5.425391e-03
callwait   8.500811e-04
months    -1.908024e-02
uniqsubs   1.619105e-01
actvsubs  -1.351256e-01
phones     5.957587e-02
models     .
eqpdays    1.382974e-03
age1      -2.772636e-03
age2      -8.534496e-04
```

```
children   1.403129e-01
credita   -1.531635e-01
creditaa  -3.652155e-01
prizmrur   1.003076e-01
prizmub   -5.409206e-02
prizmtwn   3.472156e-02
refurb     2.789570e-01
webcap    -1.230094e-01
truck      5.103158e-02
rv         1.365985e-02
occprof   -1.823898e-02
occcler    7.726085e-02
occcrft    .
occstud    .
occhmkr    5.071779e-02
occret    -1.285097e-02
occself   -6.820692e-02
ownrent    1.527827e-02
marryun    5.873367e-02
marryyes   2.092830e-02
mailord    .
mailres   -1.356902e-01
mailflag  -1.909547e-03
travel    -2.452862e-02
pcown      1.789798e-02
creditcd   5.803585e-02
retcalls   3.492776e-01
retaccpt  -2.930552e-01
newcelly  -3.298212e-02
newcelln   7.610507e-03
refer     -9.091549e-02
incmiss    .
income    -2.855282e-03
mcycle     1.422297e-01
setprcm   -8.667456e-03
setprc     7.111671e-04
retcall    5.969897e-01
```

```
# lambda_id <- colnames(coef(cv_lasso, select='min')) #
# Identificador para el lambda deseado
```

```
# cv_lasso$gamlr$lambda[lambda_id] # Valor del lambda deseado



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

```
      seg99
0.0006067582
```

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

```
# Predicciones y_pred<-predict(cv_lasso$gamlr, newdata =
# data_validation[,-1], type = 'response', select =
# cv_lasso$seg.min) y_pred<-as.numeric(y_pred)

# A
lasso_score <- predict(cvlasso_a, newdata = data_validation[,
    -1], type = "response", select = "min")

# FVG lasso_predict <- as.numeric(lasso_score > 0.5)

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

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

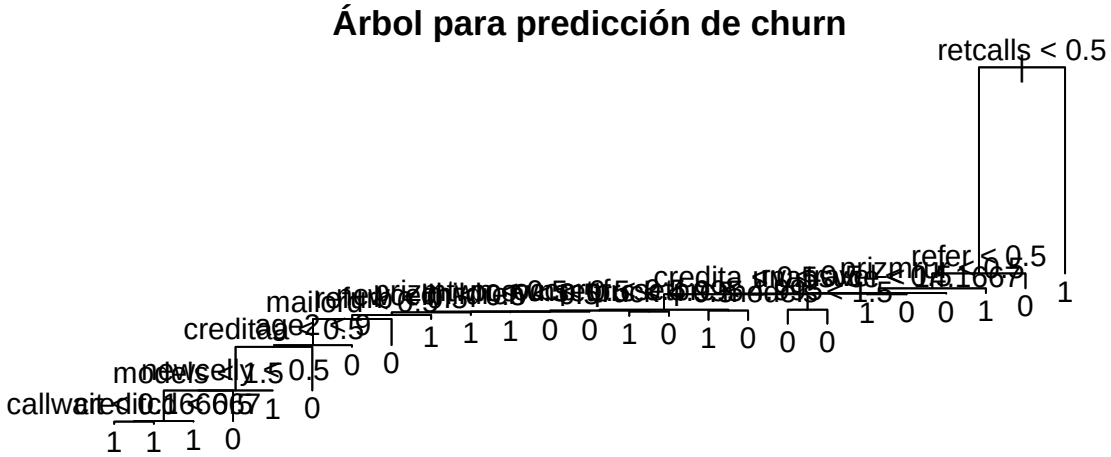
```
# A
tree_control <- tree.control(nobs = nrow(data_training_a[, -1]),
    mincut = 1000, mindev = 0.05)



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

9. Grafica el árbol resultante

```
plot(tree_estimation)
text(tree_estimation, pretty = 0)
title("Árbol para predicción de churn", line = 0.5)
```

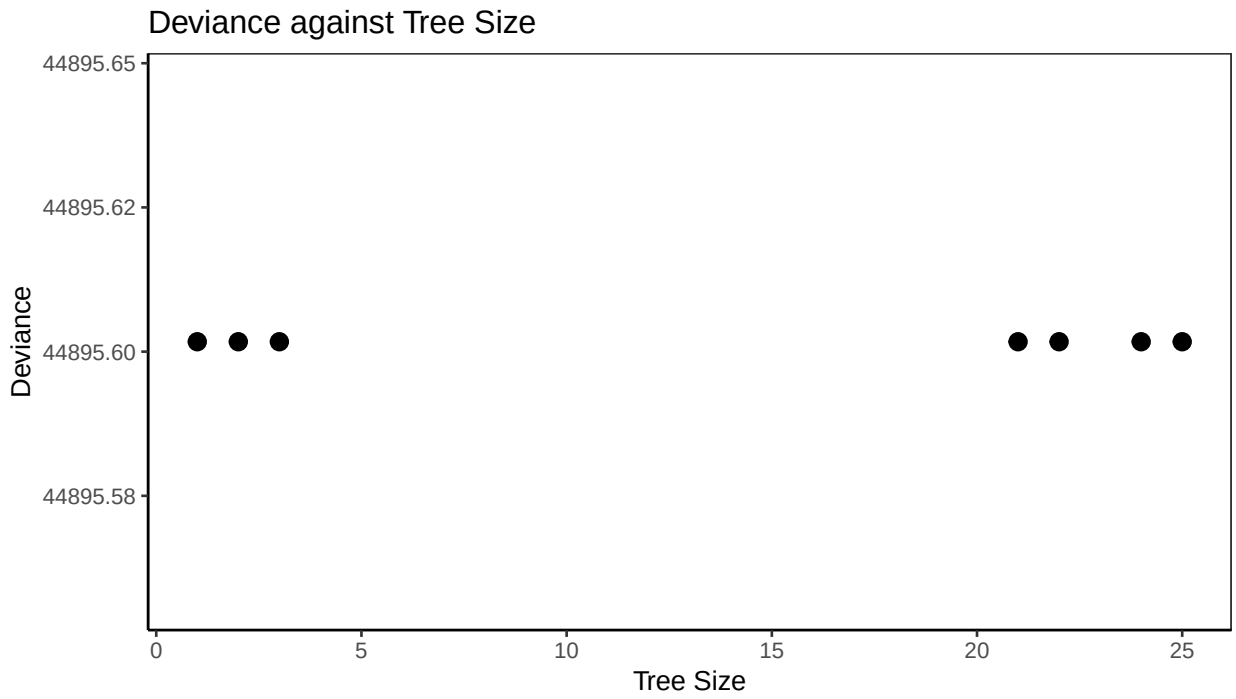


10. Poda el árbol usando CV. Muestra el resultado. Grafica Tree Size vs Binomial Deviance. Cuál es el mejor tamaño del árbol? Mejora el Error?

```
cv_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"
```

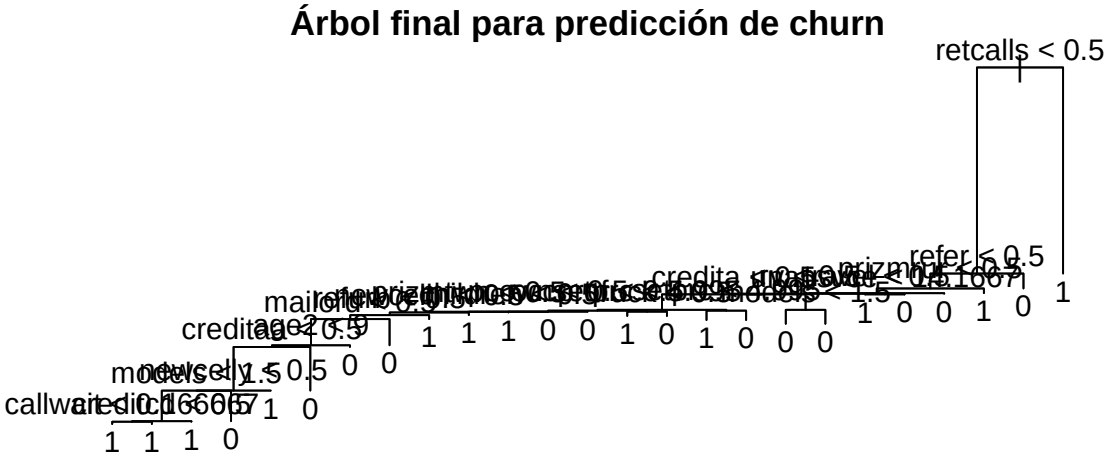
```
# Size con menor deviance
min_dev_size <- cv_tree$size[match(min(cv_tree$dev), cv_tree$dev)]


# 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 Si
        xlab("Tree Size") + ylab("Deviance") + theme_bw() + theme(axis.line = element_line(colo
        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 y elegimos el modelo de 25 nodos. 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*

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

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


plot(tree_cut)
text(tree_cut, pretty = 0)
title("Árbol final para predicción de churn", line = 0.5)
```



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

```
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")
```

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`

```
# 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

a <- Sys.time()  # para observar el tiempo de estimacion

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,
        )

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

    rf <- NULL
    k <- k + 1
}

Sys.time() - a
```

```
Time difference of 54.88877 secs
```

```
stopCluster(cl)

rf <- data.frame(trees, error)

kable(rf, col.names = c("Trees", "Error")) %>% kable_styling(position = "center",
    latex_options = "HOLD_position")
```

Trees	Error
100	0.3968187
200	0.3884722
300	0.3858111
500	0.3786440
700	0.3833011
800	0.3820612

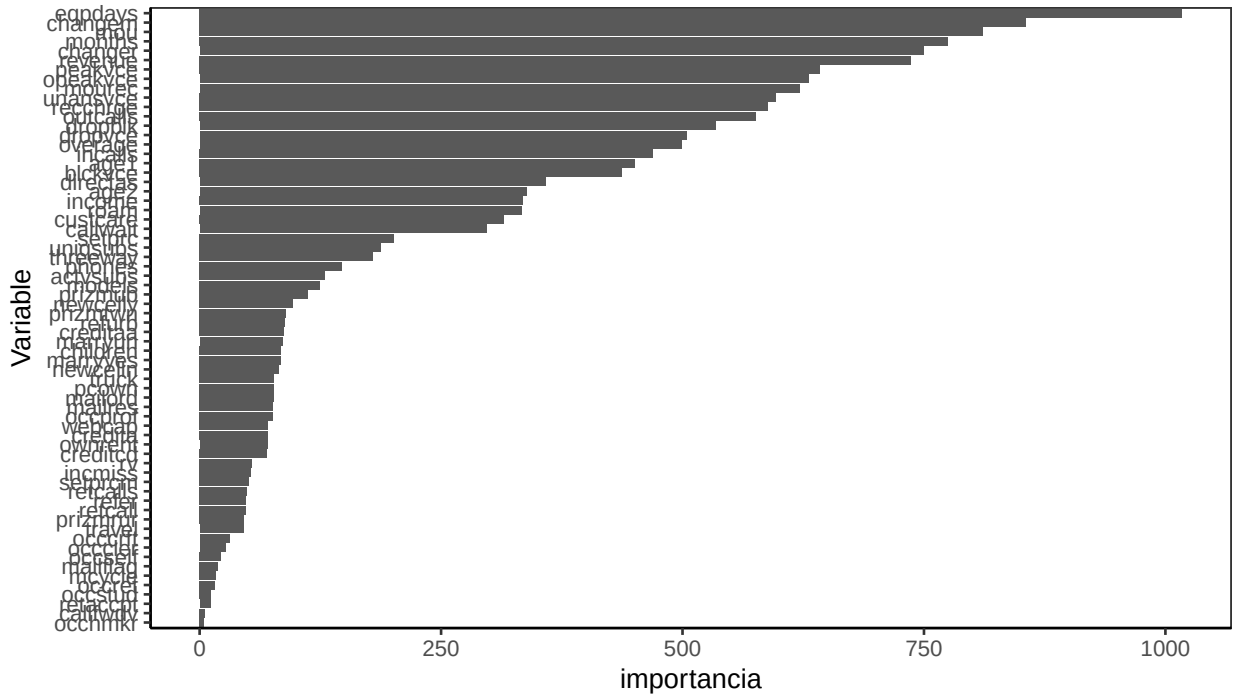
Parece ser que con una $B = 200$, el error de predicción ya no parece reducirse mucho. Cabe mencionar que el error in sample era mucho menor con las bases resultantes de oversampling, pero lo opuesto pasaba en OOS.

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

```
# 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, aes(x = reorder(names, importance), y = importance)) +
    geom_bar(stat = "identity") + xlab("Variable") + ylab("importancia") +
    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())
```



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

```
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"
```

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

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

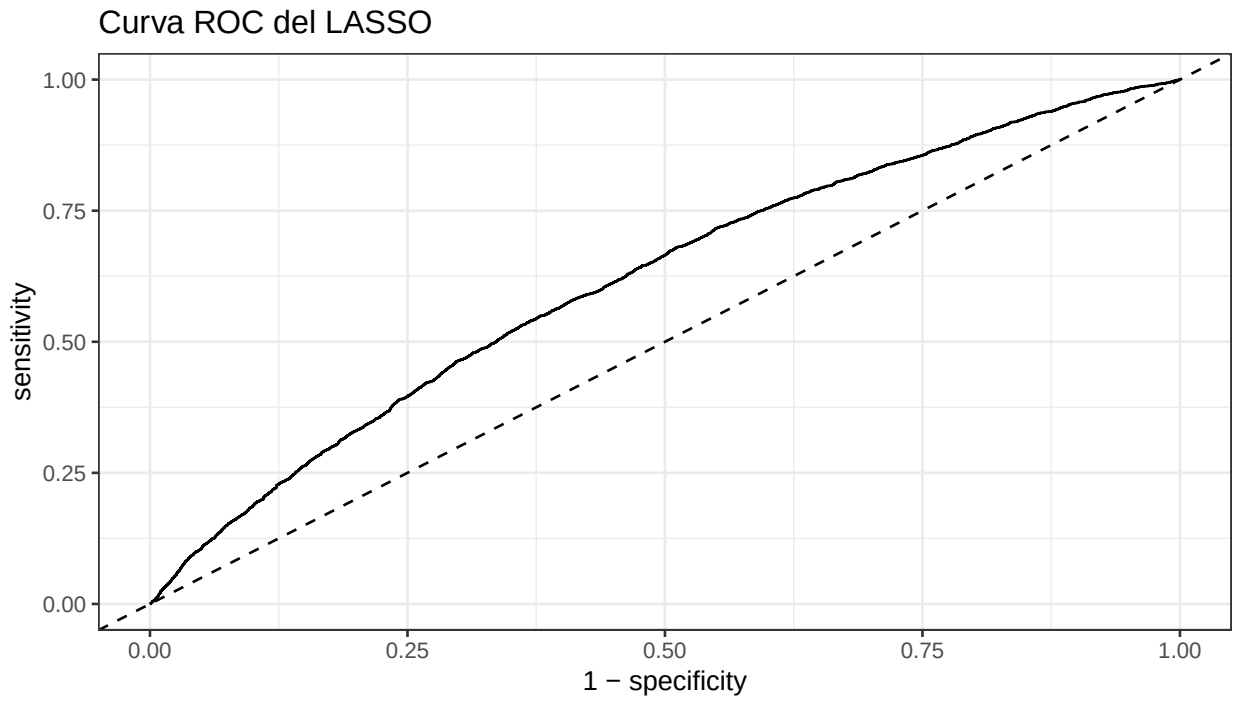
```
socket cluster with 8 nodes on host 'localhost'
```

```
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])

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

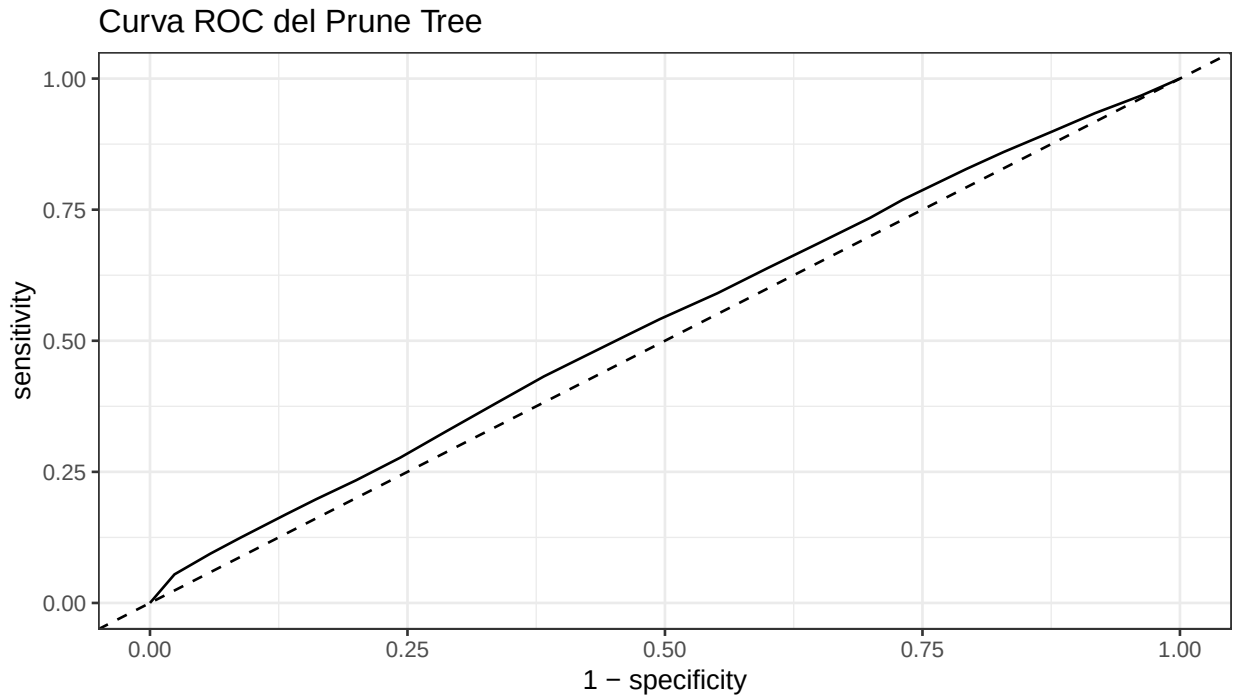
17 (4 pts). Genera graficas de las curvas ROC para los tres modelos. Cual parece ser mejor?

```
# Lasso
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()
```

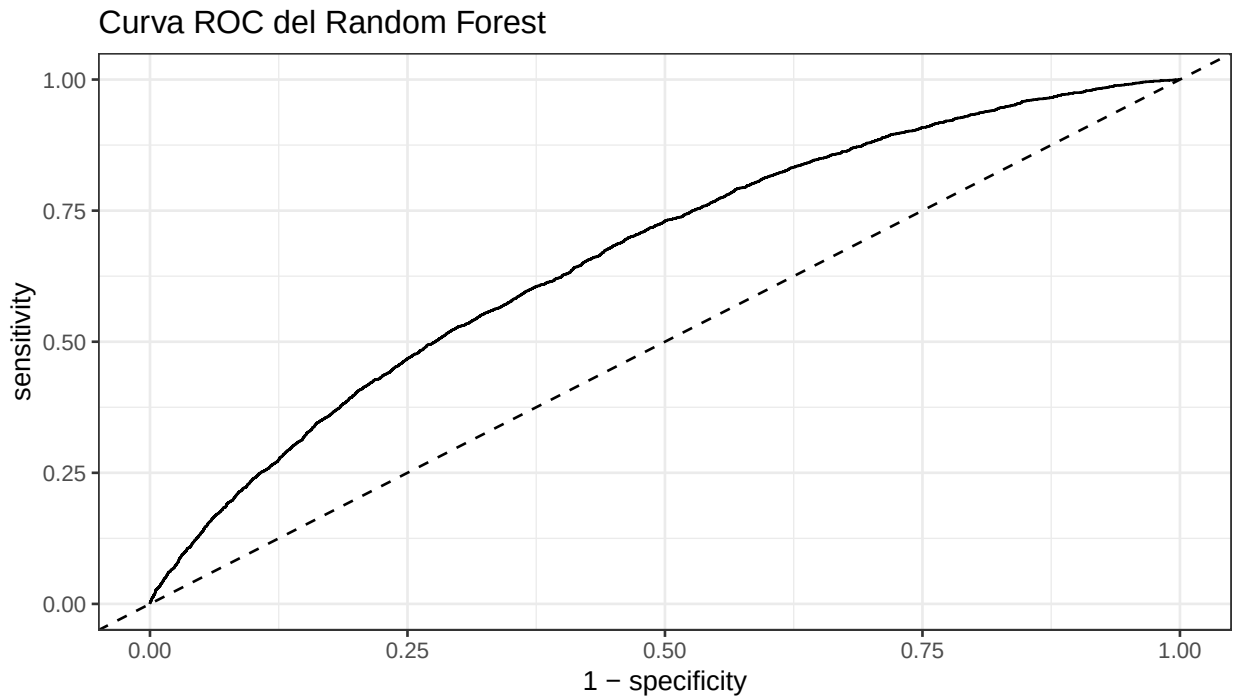



```
# Trees
roc_curve(data = A, truth = as.factor(churn_validation), tree_score,
```

```
    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
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()
```



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.

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

```
# 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")) %>% kable_styling(position = "center",
    latex_options = "HOLD_position")
```

Modelos	AUC
CV-LASSO	0.6157044
Pruned Tree	0.5346934
Random Forest	0.6647805

Confirmamos que el mejor modelo es el Random Forest.

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?

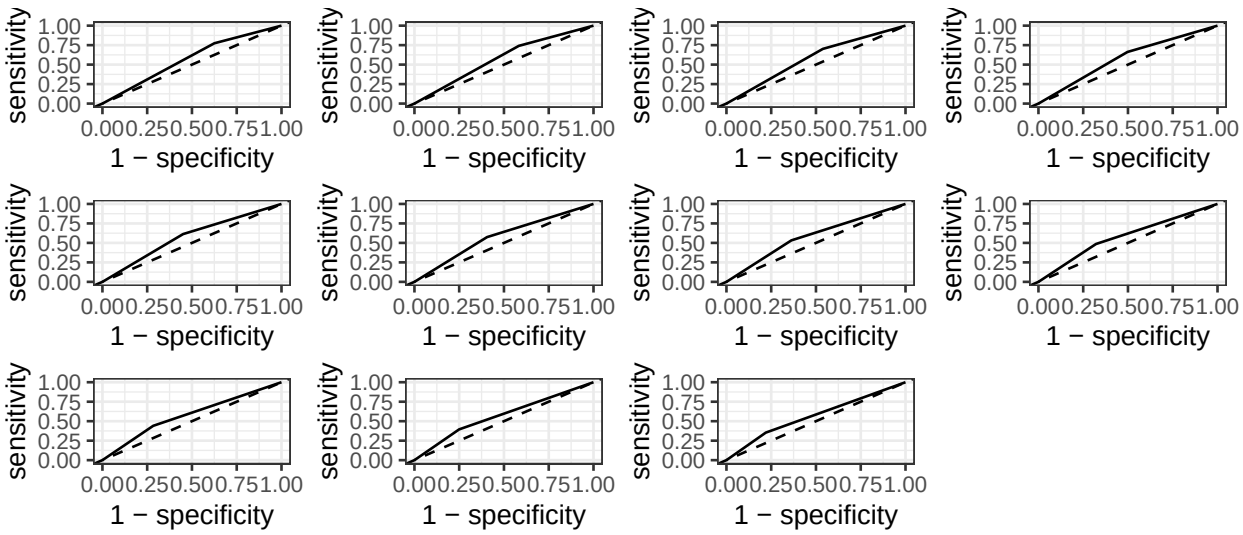
Para elegir el punto de corte, graficamos la Curva ROC para diferentes puntos de corte. Partimos de lo más general y lo fuimos acotando, con el 0.5 como punto central del intervalo.

```
# ------------------------------ Puntos de corte
# ------------------------------ Nota (FVG): Ya hice el punto
# de corte para LASSO desde que se definió inicialmente el df
# A en la sección de LASSO.  Lo hice en 0.5, y me parece que
# el resto de los modelos toman el mismo criterio.

# Nota (ALF): Moví esa predicción hasta acá. Sí parece ser la
# más adecuada pero # creo que en esta pregunta queda mejor
# justificarla y antes no se usa.

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 grafique curva
# ROC sobre el predict

gglist <- NULL
auc_vector <- 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)

    curva <- roc_curve(data = A, truth = as.factor(churn_validation),
        lasso_1, event_level = "second") %>% ggplot(aes(x = 1 -
        specificity, y = sensitivity)) + geom_abline(slope = 1,
        intercept = 0, linetype = "dashed") + geom_path() + theme_bw()

    auc_vector[it] <- roc_auc(data = A, truth = factor(churn_validation),
        lasso_1, event_level = "second")[1, 3]

    gglist[[it]] <- curva
    corte[it] <- x
    curva <- NULL
    it <- it + 1
}


grid.arrange(grobs = gglist, ncol = 4, nrow = 4)
```




```
data.frame(corte, unlist(auc_vector))
```

```
   corte unlist.auc_vector.
```

```
1   0.45          0.5744871
2   0.46          0.5789149
3   0.47          0.5817287
4   0.48          0.5824983
5   0.49          0.5813174
6   0.50          0.5844272
7   0.51          0.5848566
8   0.52          0.5823133
9   0.53          0.5785329
10  0.54          0.5726817
11  0.55          0.5668697
```

```
# punto de corte elegido
A$lasso_predict <- as.numeric(A$lasso_score > 0.51)
```

El máximo es 0.51, así que elegimos ese como nuestro punto de corte.

```
# ------------------------------ Matrices de confusión
# ------------------------------

# 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))


kable(matconf_lasso$table, caption = "Lasso") %>% kable_styling(position = "center",
    latex_options = "HOLD_position")
```

Table 3. Lasso

	0	1
0	6458	1905
1	3677	2170

```
kable(matconf_tree$table, caption = "Tree") %>% kable_styling(position = "center",
    latex_options = "HOLD_position")
```

Table 4. Tree

	0	1
0	5435	1992
1	4700	2083

```
kable(matconf_rf$table, caption = "Random Forest") %>% kable_styling(position = "center",
    latex_options = "HOLD_position")
```

Table 5. Random Forest

	0	1
0	6015	1481
1	4120	2594

Como vimos en clase, 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	0.637	0.533	0.772	0.371	0.772
Tree	0.536	0.511	0.732	0.307	0.732
Random Forest	0.593	0.637	0.802	0.386	0.802

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* .

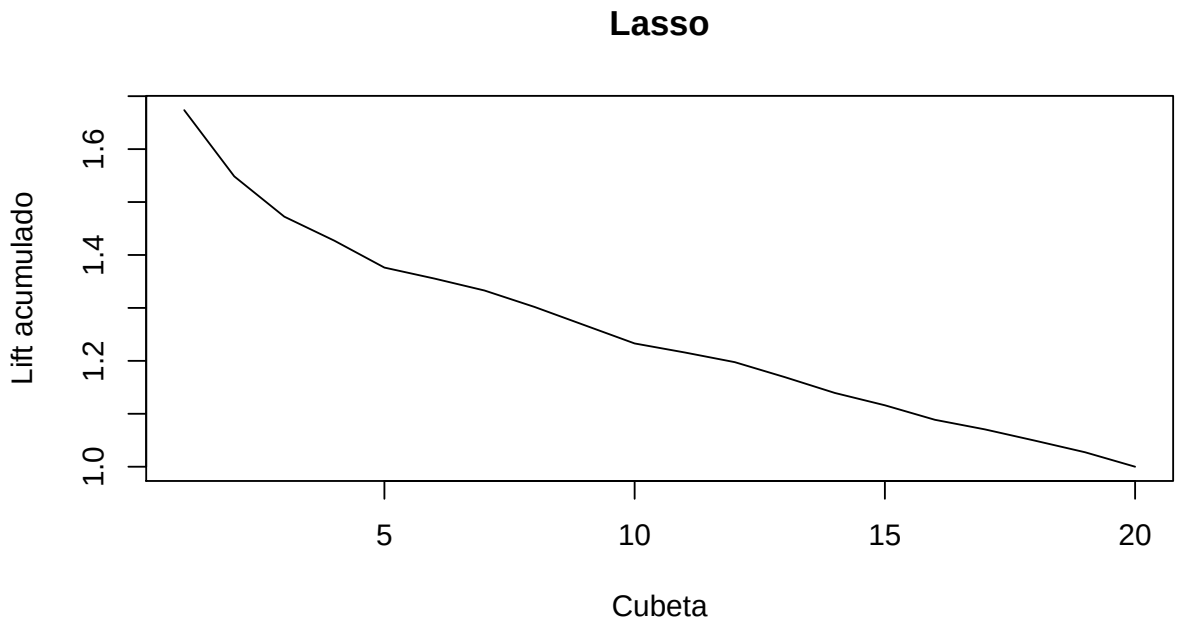
20 (2pts). Finalmente, construye una lift table. Esto es, para 20 grupos del score predecido, genera 1) El promedio de las predicciones, 2) el promedio del churn observado. Existe monotonía? El mejor algoritmo es monotónico? (Tip: usa `ntile` para generar los grupos a partir de las predicciones)

```
# 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, col.names = c("Percentil", "Valor esperado",
    "Promedio Lasso", "Promedio Tree", "Promedio RF")) %>% kable_styling(position = "center",
    latex_options = "HOLD_position")
```

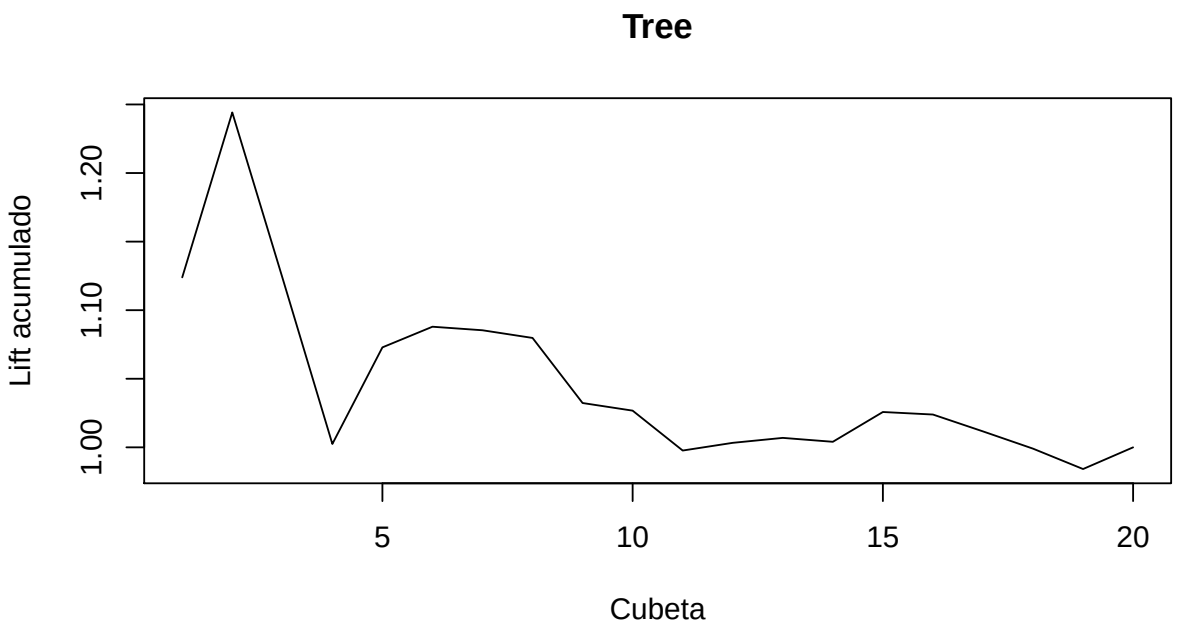
Percentil	Valor esperado	Promedio Lasso	Promedio Tree	Promedio RF
5	0.2867699	0.4802817	0.4338028	0.5436620
10	0.2867699	0.4433498	0.3638283	0.5080929
15	0.2867699	0.4218677	0.3345847	0.4819334
20	0.2867699	0.4088670	0.3216045	0.4616467
25	0.2867699	0.3944257	0.3147523	0.4487613
30	0.2867699	0.3882243	0.3140981	0.4325592
35	0.2867699	0.3818621	0.3130907	0.4186608
40	0.2867699	0.3731527	0.3128079	0.4030612
45	0.2867699	0.3633094	0.3087269	0.3900532
50	0.2867699	0.3531316	0.3059817	0.3801548
55	0.2867699	0.3485605	0.3026232	0.3726168
60	0.2867699	0.3433028	0.3004926	0.3621863
65	0.2867699	0.3353183	0.2992638	0.3544825
70	0.2867699	0.3267317	0.2973761	0.3458329
75	0.2867699	0.3200713	0.2964249	0.3362109
80	0.2867699	0.3121042	0.2955665	0.3264426
85	0.2867699	0.3069217	0.2935089	0.3170227
90	0.2867699	0.3008054	0.2912659	0.3075299
95	0.2867699	0.2946144	0.2888362	0.2980221
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 <- as.integer(as.character(variable_objetivo))
    if (is.factor(score_prediccion))
        score_prediccion <- as.integer(as.character(score_prediccion))
    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
```

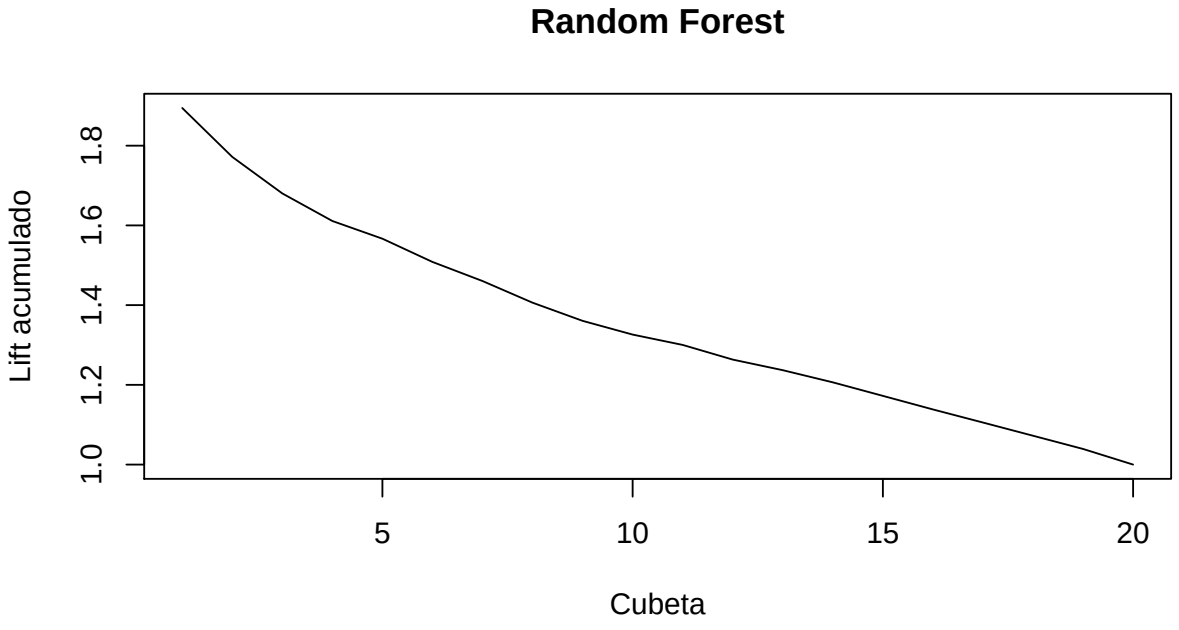
```
lift_lasso <- tabla_lift(A$churn_validation, A$lasso_score, groups = 20)
graphics::plot(lift_lasso$bucket, lift_lasso$Cumlift, type = "l",
    ylab = "Lift acumulado", xlab = "Cubeta", main = "Lasso")
```




```
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")
```



EXTRA: XGB

```
# 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)
```

```
[1] train-auc:0.650117  eval-auc:0.621607
Multiple eval metrics are present. Will use eval_auc for early stopping.
Will train until eval_auc hasn't improved in 10 rounds.

[2] train-auc:0.666514  eval-auc:0.634619
[3] train-auc:0.669490  eval-auc:0.637811
[4] train-auc:0.673285  eval-auc:0.641212
[5] train-auc:0.678104  eval-auc:0.643719
[6] train-auc:0.682485  eval-auc:0.645962
[7] train-auc:0.685446  eval-auc:0.647391
[8] train-auc:0.687835  eval-auc:0.649248
[9] train-auc:0.691503  eval-auc:0.651482
[10]    train-auc:0.693612  eval-auc:0.653245
[11]    train-auc:0.694385  eval-auc:0.653290
[12]    train-auc:0.698615  eval-auc:0.655706
[13]    train-auc:0.700314  eval-auc:0.656205
[14]    train-auc:0.701126  eval-auc:0.656180
[15]    train-auc:0.702442  eval-auc:0.655774
```

```
[16]    train-auc:0.704004  eval-auc:0.656690
[17]    train-auc:0.704654  eval-auc:0.657166
[18]    train-auc:0.705217  eval-auc:0.657607
[19]    train-auc:0.706729  eval-auc:0.657647
[20]    train-auc:0.707939  eval-auc:0.657853
[21]    train-auc:0.709233  eval-auc:0.657925
[22]    train-auc:0.710540  eval-auc:0.658479
[23]    train-auc:0.712020  eval-auc:0.659106
[24]    train-auc:0.714319  eval-auc:0.660129
[25]    train-auc:0.715647  eval-auc:0.660557
[26]    train-auc:0.717105  eval-auc:0.660943
[27]    train-auc:0.718102  eval-auc:0.661658
[28]    train-auc:0.718954  eval-auc:0.661678
[29]    train-auc:0.721077  eval-auc:0.663104
[30]    train-auc:0.721712  eval-auc:0.663083
[31]    train-auc:0.722820  eval-auc:0.663191
[32]    train-auc:0.724610  eval-auc:0.663481
[33]    train-auc:0.725823  eval-auc:0.663539
[34]    train-auc:0.727076  eval-auc:0.663912
[35]    train-auc:0.728255  eval-auc:0.664286
[36]    train-auc:0.729910  eval-auc:0.664977
[37]    train-auc:0.731060  eval-auc:0.665487
[38]    train-auc:0.732321  eval-auc:0.666014
[39]    train-auc:0.733178  eval-auc:0.666226
[40]    train-auc:0.734077  eval-auc:0.666440
[41]    train-auc:0.735093  eval-auc:0.666480
[42]    train-auc:0.735967  eval-auc:0.666628
[43]    train-auc:0.737072  eval-auc:0.667246
[44]    train-auc:0.738253  eval-auc:0.667484
[45]    train-auc:0.739832  eval-auc:0.668142
[46]    train-auc:0.741538  eval-auc:0.668576
[47]    train-auc:0.742804  eval-auc:0.668683
[48]    train-auc:0.743856  eval-auc:0.668710
[49]    train-auc:0.745149  eval-auc:0.669005
[50]    train-auc:0.746172  eval-auc:0.669229
[51]    train-auc:0.747036  eval-auc:0.669675
[52]    train-auc:0.748274  eval-auc:0.669845
[53]    train-auc:0.749247  eval-auc:0.670043
[54]    train-auc:0.749915  eval-auc:0.670108
[55]    train-auc:0.751254  eval-auc:0.670179
```

```
[56]    train-auc:0.751923  eval-auc:0.670588
[57]    train-auc:0.752310  eval-auc:0.670691
[58]    train-auc:0.752783  eval-auc:0.670665
[59]    train-auc:0.754133  eval-auc:0.671220
[60]    train-auc:0.754935  eval-auc:0.671329
[61]    train-auc:0.755627  eval-auc:0.671577
[62]    train-auc:0.756864  eval-auc:0.671662
[63]    train-auc:0.758214  eval-auc:0.671627
[64]    train-auc:0.758843  eval-auc:0.671720
[65]    train-auc:0.759664  eval-auc:0.671869
[66]    train-auc:0.760753  eval-auc:0.671986
[67]    train-auc:0.761279  eval-auc:0.672088
[68]    train-auc:0.761951  eval-auc:0.672278
[69]    train-auc:0.762608  eval-auc:0.672472
[70]    train-auc:0.763193  eval-auc:0.672764
[71]    train-auc:0.764293  eval-auc:0.673265
[72]    train-auc:0.764853  eval-auc:0.673514
[73]    train-auc:0.765465  eval-auc:0.673697
[74]    train-auc:0.766366  eval-auc:0.673902
[75]    train-auc:0.767126  eval-auc:0.673981
[76]    train-auc:0.767710  eval-auc:0.674287
[77]    train-auc:0.768686  eval-auc:0.674514
[78]    train-auc:0.769628  eval-auc:0.674773
[79]    train-auc:0.770127  eval-auc:0.674832
[80]    train-auc:0.770772  eval-auc:0.674858
[81]    train-auc:0.771364  eval-auc:0.674803
[82]    train-auc:0.772048  eval-auc:0.675038
[83]    train-auc:0.772798  eval-auc:0.675086
[84]    train-auc:0.773537  eval-auc:0.674975
[85]    train-auc:0.774271  eval-auc:0.674980
[86]    train-auc:0.774768  eval-auc:0.675326
[87]    train-auc:0.775303  eval-auc:0.675394
[88]    train-auc:0.775600  eval-auc:0.675686
[89]    train-auc:0.776298  eval-auc:0.675528
[90]    train-auc:0.776730  eval-auc:0.675550
[91]    train-auc:0.776990  eval-auc:0.675661
[92]    train-auc:0.777914  eval-auc:0.675599
[93]    train-auc:0.778946  eval-auc:0.675653
[94]    train-auc:0.779649  eval-auc:0.675812
[95]    train-auc:0.780551  eval-auc:0.676116
```

```
[96]    train-auc:0.781341  eval-auc:0.676027
[97]    train-auc:0.781925  eval-auc:0.676067
[98]    train-auc:0.782268  eval-auc:0.675961
[99]    train-auc:0.782970  eval-auc:0.676039
[100]   train-auc:0.783607  eval-auc:0.676134
```

```r
# Actualmente le pego al 0.676 con 6, 0.06, 0.85, 0.7, 10,
# 100

# Predicción
pred <- predict(xgb_model, sparse_test)
```

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

# Gráfico
xgb.plot.importance(importance_matrix)
```