# Predicción de Churn usando Random Forest en R

AUTHOR

Nestor Montano P

# Caso: Customer Churn Prediction 2020

Esta competencia consiste en predecir si un cliente cambiará de proveedor de telecomunicaciones, algo que se conoce como "Churn".

Kostas Diamantaras. (2020). Customer Churn Prediction 2020. Kaggle. https://kaggle.com/competitions/customer-churn-prediction-2020

# EDA y Preliminares

## Librerias

```r
## R
library(tidyverse) # Conjunto de paquetes para manejo de datos
library(magrittr) # Pipe
library(tidymodels) # Machine Learning en R
library(skimr) # Descriptivas univariadas masivas
library(ranger) # Random Forest
## Estos son para hacer computacion en paralelo en Windows
library(parallel)
library(doParallel)
```

## Importar Datos

Importar y modificar los objetos para que sean del tipo correcto

```r
# Leer el archivo de excel y asignarlo al objeto data
data <- read_csv(file = "Data/train_kaggle.csv")
```

```
Rows: 4250 Columns: 20
-- Column specification --------------------------------------------------------
Delimiter: ","
chr  (5): state, area_code, international_plan, voice_mail_plan, churn
dbl (15): account_length, number_vmail_messages, total_day_minutes, total_da...

i Use `spec()` to retrieve the full column specification for this data.
i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```r
data %>% glimpse
```

```
Rows: 4,250
Columns: 20
$ state                         <chr> "OH", "NJ", "OH", "OK", "MA", "MO", "LA"~
$ account_length                <dbl> 107, 137, 84, 75, 121, 147, 117, 141, 65~
```

```
$ area_code                     <chr> "area_code_415", "area_code_415", "area_~
$ international_plan            <chr> "no", "no", "yes", "yes", "no", "yes", "~
$ voice_mail_plan               <chr> "yes", "no", "no", "no", "yes", "no", "n~
$ number_vmail_messages         <dbl> 26, 0, 0, 0, 24, 0, 0, 37, 0, 0, 0, 0, 0~
$ total_day_minutes             <dbl> 161.6, 243.4, 299.4, 166.7, 218.2, 157.0~
$ total_day_calls               <dbl> 123, 114, 71, 113, 88, 79, 97, 84, 137, ~
$ total_day_charge              <dbl> 27.47, 41.38, 50.90, 28.34, 37.09, 26.69~
$ total_eve_minutes             <dbl> 195.5, 121.2, 61.9, 148.3, 348.5, 103.1,~
$ total_eve_calls               <dbl> 103, 110, 88, 122, 108, 94, 80, 111, 83,~
$ total_eve_charge              <dbl> 16.62, 10.30, 5.26, 12.61, 29.62, 8.76, ~
$ total_night_minutes           <dbl> 254.4, 162.6, 196.9, 186.9, 212.6, 211.8~
$ total_night_calls             <dbl> 103, 104, 89, 121, 118, 96, 90, 97, 111,~
$ total_night_charge            <dbl> 11.45, 7.32, 8.86, 8.41, 9.57, 9.53, 9.7~
$ total_intl_minutes            <dbl> 13.7, 12.2, 6.6, 10.1, 7.5, 7.1, 8.7, 11~
$ total_intl_calls              <dbl> 3, 5, 7, 3, 7, 6, 4, 5, 6, 5, 2, 5, 9, 4~
$ total_intl_charge             <dbl> 3.70, 3.29, 1.78, 2.73, 2.03, 1.92, 2.35~
$ number_customer_service_calls <dbl> 1, 0, 2, 3, 3, 0, 1, 0, 4, 0, 1, 3, 4, 1~
$ churn                         <chr> "no", "no", "no", "no", "no", "no", "no"~
```

### Corregir tipos de datos

```r
# Convertir a factor
data %>%
  mutate( churn = factor(churn,
                         levels= c("yes","no"),
                         labels= c("si", "no"))
  ) -> data
```

# EDA Univariado

```r
skim(data)
```

Data summary

| | |
|---|---|
| Name | data |
| Number of rows | 4250 |
| Number of columns | 20 |
| _______________________ | |
| Column type frequency: | |
| character | 4 |
| factor | 1 |
| numeric | 15 |
| ________________________ | |
| Group variables | None |

**Variable type: character**

| skim_variable | n_missing | complete_rate | min | max | empty | n_unique | whitespace |
|---|---|---|---|---|---|---|---|
| state | 0 | 1 | 2 | 2 | 0 | 51 | 0 |
| area_code | 0 | 1 | 13 | 13 | 0 | 3 | 0 |
| international_plan | 0 | 1 | 2 | 3 | 0 | 2 | 0 |
| voice_mail_plan | 0 | 1 | 2 | 3 | 0 | 2 | 0 |

**Variable type: factor**

| skim_variable | n_missing | complete_rate | ordered | n_unique | top_counts |
|---|---|---|---|---|---|
| churn | 0 | 1 | FALSE | 2 | no: 3652, si: 598 |

**Variable type: numeric**

| skim_variable | n_missing | complete_rate | mean | sd | p0 | p25 | p50 | p75 | p100 | hist |
|---|---|---|---|---|---|---|---|---|---|---|
| account_length | 0 | 1 | 100.24 | 39.70 | 1 | 73.00 | 100.00 | 127.00 | 243.00 | |
| number_vmail_messages | 0 | 1 | 7.63 | 13.44 | 0 | 0.00 | 0.00 | 16.00 | 52.00 | |
| total_day_minutes | 0 | 1 | 180.26 | 54.01 | 0 | 143.33 | 180.45 | 216.20 | 351.50 | |
| total_day_calls | 0 | 1 | 99.91 | 19.85 | 0 | 87.00 | 100.00 | 113.00 | 165.00 | |
| total_day_charge | 0 | 1 | 30.64 | 9.18 | 0 | 24.36 | 30.68 | 36.75 | 59.76 | |
| total_eve_minutes | 0 | 1 | 200.17 | 50.25 | 0 | 165.93 | 200.70 | 233.78 | 359.30 | |
| total_eve_calls | 0 | 1 | 100.18 | 19.91 | 0 | 87.00 | 100.00 | 114.00 | 170.00 | |
| total_eve_charge | 0 | 1 | 17.02 | 4.27 | 0 | 14.10 | 17.06 | 19.87 | 30.54 | |
| total_night_minutes | 0 | 1 | 200.53 | 50.35 | 0 | 167.22 | 200.45 | 234.70 | 395.00 | |
| total_night_calls | 0 | 1 | 99.84 | 20.09 | 0 | 86.00 | 100.00 | 113.00 | 175.00 | |
| total_night_charge | 0 | 1 | 9.02 | 2.27 | 0 | 7.52 | 9.02 | 10.56 | 17.77 | |
| total_intl_minutes | 0 | 1 | 10.26 | 2.76 | 0 | 8.50 | 10.30 | 12.00 | 20.00 | |
| total_intl_calls | 0 | 1 | 4.43 | 2.46 | 0 | 3.00 | 4.00 | 6.00 | 20.00 | |
| total_intl_charge | 0 | 1 | 2.77 | 0.75 | 0 | 2.30 | 2.78 | 3.24 | 5.40 | |
| number_customer_service_calls | 0 | 1 | 1.56 | 1.31 | 0 | 1.00 | 1.00 | 2.00 | 9.00 | |

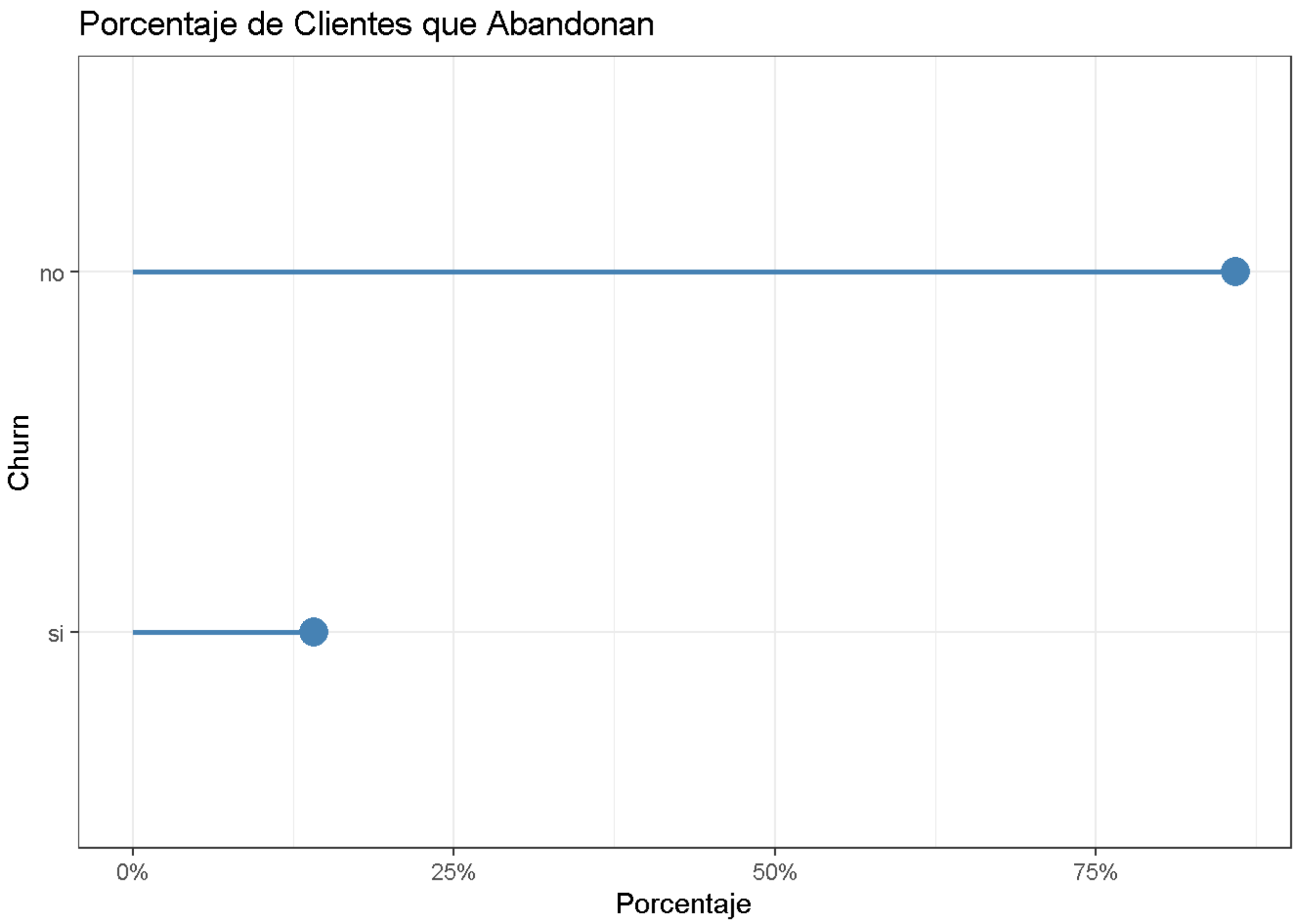
## Balanceo

```
data %>%
  group_by( churn) %>%
  count( name = 'frec') %>%
  ungroup() %>%
  mutate( Porc= frec/sum(frec))
```

```
# A tibble: 2 x 3
  churn  frec  Porc
  <fct> <int> <dbl>
```

```
1 si      598 0.141
2 no     3652 0.859
```

```
data %>%
  group_by( churn) %>%
  count( name = 'frec') %>%
  ungroup() %>%
  mutate( Porc= frec/sum(frec)) %>%
  ggplot( aes(x= churn, y= Porc)) +
  geom_segment( aes(xend= churn, y=0, yend=Porc),
                color= "steelblue", linewidth= 1) +
  geom_point( size=5, color= "steelblue") +
  coord_flip() +
  scale_y_continuous( labels = percent_format()) +
  labs(title= 'Porcentaje de Clientes que Abandonan',
       y= "Porcentaje", x= "Churn") +
  theme_bw()
```




Se puede ver una proporción de 6 a 1 entre el "No" y el "Si"

# EDA Multivariado

```
# data %>%
#   select_if( is.numeric) %>%
#   GGally::ggscatmat()
```

Otra opción para ver la correlacion es:

```
# data %>%
#   select_if( is.numeric) %>%
#   cor %>%
#   corrplot::corrplot(
#     method ="number", type = "lower" )
```

# Modelamiento

## Train-Test Split

```
set.seed(1234) # Semilla para aleatorios
split <- data %>%
    initial_split(
      prop = 0.8, # Porcentaje al train
      strata = churn  # Estratificación del muestreo
      )
```

Con el split creado, podemos obtener nuestro train y test, así:

```
train <- training(split)
dim(train)
```

```
[1] 3399   20
```

```
test <- testing(split)
dim(test)
```

```
[1] 851  20
```

## Preprocesamiento

### Balancear usando pesos

Aqui vamos a crear una columna para la ponderacion de cada fila, esta ponderación se va a usar en la estimación del modelo y en los pasos de la receta que sean “supervisado” (cuando se usa la variable ”y” en el preprocesamiento)

```
train %>%
  mutate(
    ## crear la variable con los pesos
    case_wts = ifelse(churn == "si", 6, 1),
    ## crea el vector de importancia ponderada
    case_wts = importance_weights(case_wts)
  ) ->
  train
```

## Receta de preprocesamiento

```r
receta <- train %>%
  recipe(churn ~ . ) %>% ## Crea la receta
  ## Eliminar variables que no usaremos
  # step_rm() %>%
  ## Crear nuevas variables (insight desde el EDA)
  # step_mutate( account_length_anio= account_length/12 )
  ## Imputar los datos
  # step_impute_mean()
  step_impute_knn( all_predictors() ) %>%
  ## Estandarizacion/Normalizacion de numericas
  step_normalize( all_numeric(), -all_outcomes()) %>%
  ## Crear una categoría "otros" que agrupe a categorias pequeñas
  step_other(all_nominal(), -all_outcomes() , threshold = 0.07, other = "otros") %>%
  ## Crear una categoría "new" para observaciones con labels "no muestreados"
  step_novel(all_nominal(), -all_outcomes() , new_level = "new") %>%
  ## Crear variables indicadoras para cada categoría
  step_dummy(all_nominal(), -all_outcomes() ) %>% # Dummy
  ## Eliminar automáticamente variables con alta correlacion
  ## para evitar la multicolinealidad xi ~ xj
  # step_corr(all_numeric(), -all_outcomes(), threshold = 0.9) %>%
  ## Tambien podemos eliminar variables con multicolinealidad "a mano"
  step_rm(total_day_charge, total_eve_charge,
          total_night_charge, total_intl_charge) %>% # Eliminar
  ## Eliminar columnas con varianza cercana a cero
  step_nzv(all_predictors())
```

```r
receta
```

```
-- Recipe ----------------------------------------------------------------------


-- Inputs

Number of variables by role

outcome:       1
predictor:    19
case_weights:  1


-- Operations

* K-nearest neighbor imputation for: all_predictors()

* Centering and scaling for: all_numeric(), -all_outcomes()

* Collapsing factor levels for: all_nominal(), -all_outcomes()

* Novel factor level assignment for: all_nominal(), -all_outcomes()
```

```
* Dummy variables from: all_nominal(), -all_outcomes()

* Variables removed: total_day_charge, total_eve_charge, ...

* Sparse, unbalanced variable filter on: all_predictors()
```

# Entrenamiento y ajuste de Hiperparámetros

Para ajustar los hiperparámetros usaremos la estrategia de definir mallas de búsqueda y evaluar dichas combinaciones sobre un remuestreo (crossvalidation), para ello necesitamos:
- Remuestreo
- Métricas para evaluar y comparar los modelos

## Remuestreo

Se define una estrategia de remuestreo (para poder ajustar hiperparámetros)

```
set.seed(1234)
cv <- vfold_cv(train, v = 5, repeats = 1, strata = churn)
cv
```

```
#  5-fold cross-validation using stratification
# A tibble: 5 x 2
  splits             id
  <list>             <chr>
1 <split [2718/681]> Fold1
2 <split [2719/680]> Fold2
3 <split [2719/680]> Fold3
4 <split [2720/679]> Fold4
5 <split [2720/679]> Fold5
```

## Métricas

Así también definimos las métricas que queremos que se ejecuten en cada remuestreo

```
metricas <- metric_set(accuracy, sens, spec, bal_accuracy)
metricas
```

```
# A tibble: 4 x 3
  metric       class        direction
  <chr>        <chr>        <chr>
1 accuracy     class_metric maximize
2 sens         class_metric maximize
3 spec         class_metric maximize
4 bal_accuracy class_metric maximize
```

## Especificacion del modelo

Fuente:
* https://www.tidymodels.org/find/parsnip/

* https://parsnip.tidymodels.org/reference/rand_forest.html
* https://parsnip.tidymodels.org/reference/details_rand_forest_ranger.html

```
rf_sp <-
  rand_forest(
    mtry = tune(), trees = tune(), min_n = tune() ) %>%
  set_engine("ranger", importance = "impurity") %>%
  set_mode("classification")
```

Notar que no se ha usado todos los hiperparámetros

## workflow

```
rf_wflow <-
  workflow() %>%
  add_recipe(receta) %>%
  add_model(rf_sp) %>%
  add_case_weights(case_wts) ## Aquí agregamos los pesos

rf_wflow
```

```
== Workflow ====================================================================
Preprocessor: Recipe
Model: rand_forest()

-- Preprocessor ----------------------------------------------------------------
7 Recipe Steps

* step_impute_knn()
* step_normalize()
* step_other()
* step_novel()
* step_dummy()
* step_rm()
* step_nzv()

-- Case Weights ----------------------------------------------------------------
case_wts

-- Model -----------------------------------------------------------------------
Random Forest Model Specification (classification)

Main Arguments:
  mtry = tune()
  trees = tune()
  min_n = tune()

Engine-Specific Arguments:
  importance = impurity

Computational engine: ranger
```

# Afinamiento de hiperparametros

## Malla de Busqueda

```
set.seed(123)
rf_grid <- rf_sp %>%
  ## preguntamos los parametros tuneables del modelo
  parameters() %>%
  ## Vamos a definir un rango para el min_n y mtry
  update(min_n= min_n( range= c(70, 170)),
         mtry= mtry( range= c(4, 7))) %>%
  grid_latin_hypercube(size = 10)
```

```
Warning: `parameters.model_spec()` was deprecated in tune 0.1.6.9003.
i Please use `hardhat::extract_parameter_set_dials()` instead.
```

## Paralelizacion

```
parallel::detectCores(logical=FALSE)
```

```
[1] 4
```

```
cl <- makePSOCKcluster(4)
registerDoParallel(cl)
# parallel::stopCluster(cl) ## Esto se debe ejecutar al final
```

## Entrenamiento de Malla de Busqueda en la Crossvalidation

```
set.seed(123)
rf_tuned <- tune_grid(
  rf_wflow, ## Modelo
  resamples= cv, ## Crossvalidation
  grid = rf_grid, ## Malla de Busqueda
  metrics = metricas, ## Metricas
  control= control_grid(allow_par = T, save_pred = T) ## Paralel y Pred
  )
rf_tuned
```

```
# Tuning results
# 5-fold cross-validation using stratification
# A tibble: 5 x 5
  splits             id    .metrics          .notes           .predictions
  <list>             <chr> <list>            <list>           <list>
1 <split [2718/681]> Fold1 <tibble [40 x 7]> <tibble [0 x 3]> <tibble>
2 <split [2719/680]> Fold2 <tibble [40 x 7]> <tibble [0 x 3]> <tibble>
3 <split [2719/680]> Fold3 <tibble [40 x 7]> <tibble [0 x 3]> <tibble>
4 <split [2720/679]> Fold4 <tibble [40 x 7]> <tibble [0 x 3]> <tibble>
5 <split [2720/679]> Fold5 <tibble [40 x 7]> <tibble [0 x 3]> <tibble>
```

Evaluemos que tal es cada combinacion segun las principales metricas

```r
show_best(rf_tuned, metric = 'accuracy', n = 10)
```

```
# A tibble: 10 x 9
    mtry trees min_n .metric  .estimator  mean     n std_err .config
   <int> <int> <int> <chr>    <chr>      <dbl> <int>   <dbl> <chr>
 1     7   773    80 accuracy binary     0.932     5 0.00469 Preprocessor1_Mode~
 2     6   467    91 accuracy binary     0.925     5 0.00645 Preprocessor1_Mode~
 3     5   325    83 accuracy binary     0.923     5 0.00719 Preprocessor1_Mode~
 4     5  1441   106 accuracy binary     0.919     5 0.00644 Preprocessor1_Mode~
 5     6   950   130 accuracy binary     0.914     5 0.00539 Preprocessor1_Mode~
 6     4  1370   116 accuracy binary     0.911     5 0.00539 Preprocessor1_Mode~
 7     5  1904   126 accuracy binary     0.910     5 0.00557 Preprocessor1_Mode~
 8     6  1737   158 accuracy binary     0.909     5 0.00550 Preprocessor1_Mode~
 9     6  1124   162 accuracy binary     0.907     5 0.00585 Preprocessor1_Mode~
10     5    70   144 accuracy binary     0.906     5 0.00572 Preprocessor1_Mode~
```

```r
show_best(rf_tuned, metric = 'sens', n = 10)
```

```
# A tibble: 10 x 9
    mtry trees min_n .metric .estimator  mean     n std_err .config
   <int> <int> <int> <chr>   <chr>      <dbl> <int>   <dbl> <chr>
 1     6  1737   158 sens    binary     0.856     5  0.0135 Preprocessor1_Model~
 2     6  1124   162 sens    binary     0.854     5  0.0137 Preprocessor1_Model~
 3     5  1904   126 sens    binary     0.851     5  0.0147 Preprocessor1_Model~
 4     5  1441   106 sens    binary     0.851     5  0.0147 Preprocessor1_Model~
 5     5    70   144 sens    binary     0.849     5  0.0128 Preprocessor1_Model~
 6     4  1370   116 sens    binary     0.849     5  0.0151 Preprocessor1_Model~
 7     6   950   130 sens    binary     0.849     5  0.0159 Preprocessor1_Model~
 8     6   467    91 sens    binary     0.847     5  0.0109 Preprocessor1_Model~
 9     5   325    83 sens    binary     0.847     5  0.0158 Preprocessor1_Model~
10     7   773    80 sens    binary     0.845     5  0.0143 Preprocessor1_Model~
```

```r
show_best(rf_tuned, metric = 'spec', n = 10)
```

```
# A tibble: 10 x 9
    mtry trees min_n .metric .estimator  mean     n std_err .config
   <int> <int> <int> <chr>   <chr>      <dbl> <int>   <dbl> <chr>
 1     7   773    80 spec    binary     0.946     5 0.00546 Preprocessor1_Model~
 2     6   467    91 spec    binary     0.938     5 0.00717 Preprocessor1_Model~
 3     5   325    83 spec    binary     0.936     5 0.00737 Preprocessor1_Model~
 4     5  1441   106 spec    binary     0.929     5 0.00704 Preprocessor1_Model~
 5     6   950   130 spec    binary     0.924     5 0.00656 Preprocessor1_Model~
 6     4  1370   116 spec    binary     0.921     5 0.00599 Preprocessor1_Model~
 7     5  1904   126 spec    binary     0.920     5 0.00660 Preprocessor1_Model~
 8     6  1737   158 spec    binary     0.917     5 0.00643 Preprocessor1_Model~
 9     6  1124   162 spec    binary     0.916     5 0.00691 Preprocessor1_Model~
10     5    70   144 spec    binary     0.916     5 0.00768 Preprocessor1_Model~
```

```r
show_best(rf_tuned, metric = 'bal_accuracy', n = 10)
```

```
# A tibble: 10 x 9
    mtry trees min_n .metric      .estimator  mean     n std_err .config
```

```
   <int> <int> <int> <chr>        <chr>      <dbl> <int>   <dbl> <chr>
 1     7   773    80 bal_accuracy binary     0.896     5 0.00707 Preprocessor1_~
 2     6   467    91 bal_accuracy binary     0.893     5 0.00669 Preprocessor1_~
 3     5   325    83 bal_accuracy binary     0.891     5 0.00944 Preprocessor1_~
 4     5  1441   106 bal_accuracy binary     0.890     5 0.00819 Preprocessor1_~
 5     6   950   130 bal_accuracy binary     0.887     5 0.00759 Preprocessor1_~
 6     6  1737   158 bal_accuracy binary     0.886     5 0.00691 Preprocessor1_~
 7     5  1904   126 bal_accuracy binary     0.886     5 0.00729 Preprocessor1_~
 8     4  1370   116 bal_accuracy binary     0.885     5 0.00784 Preprocessor1_~
 9     6  1124   162 bal_accuracy binary     0.885     5 0.00697 Preprocessor1_~
10     5    70   144 bal_accuracy binary     0.883     5 0.00507 Preprocessor1_~
```

## Malla de Busqueda

```
set.seed(123)
rf_grid_2 <-  crossing(
  min_n = seq(80, 92, 3),
  mtry = c(5, 6),
  trees= seq(500, 800, 100)
)
rf_grid_2
```

```
# A tibble: 40 x 3
   min_n  mtry trees
   <dbl> <dbl> <dbl>
 1    80     5   500
 2    80     5   600
 3    80     5   700
 4    80     5   800
 5    80     6   500
 6    80     6   600
 7    80     6   700
 8    80     6   800
 9    83     5   500
10    83     5   600
# i 30 more rows
```

## Entrenamiento de Malla de Busqueda en la Crossvalidation

```
set.seed(123)
rf_tuned_2 <- tune_grid(
  rf_wflow, ## Modelo
  resamples= cv, ## Crossvalidation
  grid = rf_grid_2, ## Malla de Busqueda
  metrics = metricas, ## Metricas
  control= control_grid(allow_par = T, save_pred = T) ## Paralel y Pred
  )
rf_tuned_2
```

```
# Tuning results
# 5-fold cross-validation using stratification
# A tibble: 5 x 5
```

```
  splits             id    .metrics           .notes           .predictions
  <list>             <chr> <list>             <list>           <list>
1 <split [2718/681]> Fold1 <tibble [160 x 7]> <tibble [0 x 3]> <tibble>
2 <split [2719/680]> Fold2 <tibble [160 x 7]> <tibble [0 x 3]> <tibble>
3 <split [2719/680]> Fold3 <tibble [160 x 7]> <tibble [0 x 3]> <tibble>
4 <split [2720/679]> Fold4 <tibble [160 x 7]> <tibble [0 x 3]> <tibble>
5 <split [2720/679]> Fold5 <tibble [160 x 7]> <tibble [0 x 3]> <tibble>
```

Evaluemos que tal es cada combinacion segun las principales metricas

```
show_best(rf_tuned_2, metric = 'accuracy', n = 10)
```

```
# A tibble: 10 x 9
    mtry trees min_n .metric  .estimator  mean     n std_err .config
   <dbl> <dbl> <dbl> <chr>    <chr>      <dbl> <int>   <dbl> <chr>
 1     6   500    83 accuracy binary     0.930     5 0.00588 Preprocessor1_Mode~
 2     6   500    80 accuracy binary     0.930     5 0.00576 Preprocessor1_Mode~
 3     6   700    80 accuracy binary     0.929     5 0.00522 Preprocessor1_Mode~
 4     6   800    80 accuracy binary     0.929     5 0.00676 Preprocessor1_Mode~
 5     6   700    83 accuracy binary     0.929     5 0.00650 Preprocessor1_Mode~
 6     6   600    80 accuracy binary     0.929     5 0.00615 Preprocessor1_Mode~
 7     6   800    83 accuracy binary     0.929     5 0.00530 Preprocessor1_Mode~
 8     5   700    80 accuracy binary     0.928     5 0.00608 Preprocessor1_Mode~
 9     6   600    83 accuracy binary     0.927     5 0.00582 Preprocessor1_Mode~
10     6   500    89 accuracy binary     0.927     5 0.00737 Preprocessor1_Mode~
```

```
show_best(rf_tuned_2, metric = 'sens', n = 10)
```

```
# A tibble: 10 x 9
    mtry trees min_n .metric .estimator  mean     n std_err .config
   <dbl> <dbl> <dbl> <chr>   <chr>      <dbl> <int>   <dbl> <chr>
 1     5   600    80 sens    binary     0.854     5  0.0137 Preprocessor1_Model~
 2     5   800    83 sens    binary     0.854     5  0.0137 Preprocessor1_Model~
 3     5   800    92 sens    binary     0.854     5  0.0137 Preprocessor1_Model~
 4     5   500    83 sens    binary     0.851     5  0.0142 Preprocessor1_Model~
 5     5   600    83 sens    binary     0.851     5  0.0117 Preprocessor1_Model~
 6     5   500    89 sens    binary     0.851     5  0.0117 Preprocessor1_Model~
 7     5   800    89 sens    binary     0.851     5  0.0142 Preprocessor1_Model~
 8     5   700    80 sens    binary     0.851     5  0.0147 Preprocessor1_Model~
 9     6   500    83 sens    binary     0.851     5  0.0147 Preprocessor1_Model~
10     5   500    86 sens    binary     0.851     5  0.0147 Preprocessor1_Model~
```

```
show_best(rf_tuned_2, metric = 'spec', n = 10)
```

```
# A tibble: 10 x 9
    mtry trees min_n .metric .estimator  mean     n std_err .config
   <dbl> <dbl> <dbl> <chr>   <chr>      <dbl> <int>   <dbl> <chr>
 1     6   500    80 spec    binary     0.943     5 0.00662 Preprocessor1_Model~
 2     6   500    83 spec    binary     0.943     5 0.00657 Preprocessor1_Model~
 3     6   800    80 spec    binary     0.943     5 0.00720 Preprocessor1_Model~
 4     6   700    80 spec    binary     0.942     5 0.00569 Preprocessor1_Model~
 5     6   600    80 spec    binary     0.942     5 0.00687 Preprocessor1_Model~
 6     6   700    83 spec    binary     0.941     5 0.00702 Preprocessor1_Model~
```

```
 7     6   800    83 spec    binary      0.941      5 0.00566 Preprocessor1_Model~
 8     6   600    83 spec    binary      0.941      5 0.00633 Preprocessor1_Model~
 9     5   700    80 spec    binary      0.940      5 0.00665 Preprocessor1_Model~
10     6   700    86 spec    binary      0.940      5 0.00674 Preprocessor1_Model~
```

```r
show_best(rf_tuned_2, metric = 'bal_accuracy', n = 10)
```

```
# A tibble: 10 x 9
    mtry trees min_n .metric      .estimator  mean     n std_err .config
   <dbl> <dbl> <dbl> <chr>        <chr>      <dbl> <int>   <dbl> <chr>
 1     6   500    83 bal_accuracy binary     0.897     5 0.00781 Preprocessor1_~
 2     5   600    80 bal_accuracy binary     0.896     5 0.00814 Preprocessor1_~
 3     5   700    80 bal_accuracy binary     0.896     5 0.00803 Preprocessor1_~
 4     6   700    80 bal_accuracy binary     0.896     5 0.00698 Preprocessor1_~
 5     6   500    89 bal_accuracy binary     0.895     5 0.00850 Preprocessor1_~
 6     6   700    83 bal_accuracy binary     0.895     5 0.00846 Preprocessor1_~
 7     6   800    83 bal_accuracy binary     0.895     5 0.00834 Preprocessor1_~
 8     6   800    92 bal_accuracy binary     0.895     5 0.00810 Preprocessor1_~
 9     6   500    80 bal_accuracy binary     0.895     5 0.00732 Preprocessor1_~
10     5   800    83 bal_accuracy binary     0.895     5 0.00851 Preprocessor1_~
```

Bien, podríamos probar mallas más extensas o tomar una decisión ya con las pruebas realizadas.

## Modelo final

```r
## Definir la mejor combinacion
rf_pars_fin <- select_best(rf_tuned_2, metric = 'sens')

## Finalizar (darle valores a parametros tuneables) el workflow
rf_wflow_fin <-
  rf_wflow %>%
  finalize_workflow(rf_pars_fin)
rf_wflow_fin
```

```
== Workflow ================================================================
Preprocessor: Recipe
Model: rand_forest()

-- Preprocessor ------------------------------------------------------------
7 Recipe Steps

* step_impute_knn()
* step_normalize()
* step_other()
* step_novel()
* step_dummy()
* step_rm()
* step_nzv()

-- Case Weights ------------------------------------------------------------
case_wts

-- Model -------------------------------------------------------------------
```

```
Random Forest Model Specification (classification)

Main Arguments:
  mtry = 5
  trees = 600
  min_n = 80

Engine-Specific Arguments:
  importance = impurity

Computational engine: ranger
```

Ahora sí, se entrena el modelo final

```r
## Entrenar el modelo final
rf_fitted <- fit(rf_wflow_fin, train)
rf_fitted
```

```
== Workflow [trained] ==========================================================
Preprocessor: Recipe
Model: rand_forest()

-- Preprocessor ----------------------------------------------------------------
7 Recipe Steps

* step_impute_knn()
* step_normalize()
* step_other()
* step_novel()
* step_dummy()
* step_rm()
* step_nzv()

-- Case Weights ----------------------------------------------------------------
case_wts

-- Model -----------------------------------------------------------------------
Ranger result

Call:
 ranger::ranger(x = maybe_data_frame(x), y = y, mtry = min_cols(~5,      x), num.trees = ~600, min.node.size = min_rows(~80, x), importance = ~"impurity",      num.threads = 1, verbose = FALSE, seed = sample.int(10^5,          1), probability = TRUE, case.weights = weights)

Type:                             Probability estimation
Number of trees:                  600
Sample size:                      3399
Number of independent variables:  14
Mtry:                             5
Target node size:                 80
Variable importance mode:         impurity
Splitrule:                        gini
OOB prediction error (Brier s.):  0.06769222
```

Notar que `arbol_fitted` sigue siendo un workflow, si por algún motivo queremos sólo trabajar con el modelo, podemos:

```
rf_model_fin  <- extract_fit_parsnip(rf_fitted)
```

## Evaluacion del modelo

Vamos a comparar las metricas del modelo en el train como en el test

```
train %>%
  predict(rf_fitted , new_data = . ) %>%
  mutate(Real= train$churn) %>%
  conf_mat(truth = Real, estimate = .pred_class ) %>%
  summary
```

```
# A tibble: 13 x 3
   .metric              .estimator .estimate
   <chr>                <chr>          <dbl>
 1 accuracy             binary         0.949
 2 kap                  binary         0.800
 3 sens                 binary         0.879
 4 spec                 binary         0.961
 5 ppv                  binary         0.787
 6 npv                  binary         0.980
 7 mcc                  binary         0.802
 8 j_index              binary         0.840
 9 bal_accuracy         binary         0.920
10 detection_prevalence binary         0.157
11 precision            binary         0.787
12 recall               binary         0.879
13 f_meas               binary         0.830
```

```
test %>%
  predict(rf_fitted, new_data = . ) %>%
  mutate(Real= test$churn) %>%
  conf_mat(truth = Real, estimate = .pred_class ) %>%
  summary
```

```
# A tibble: 13 x 3
   .metric              .estimator .estimate
   <chr>                <chr>          <dbl>
 1 accuracy             binary         0.931
 2 kap                  binary         0.735
 3 sens                 binary         0.85
 4 spec                 binary         0.944
 5 ppv                  binary         0.713
 6 npv                  binary         0.975
 7 mcc                  binary         0.739
 8 j_index              binary         0.794
 9 bal_accuracy         binary         0.897
10 detection_prevalence binary         0.168
```

```
11 precision             binary         0.713
12 recall                binary         0.85
13 f_meas                binary         0.776
```

Podemos ver que no existen mucha diferencia, por lo que se puede concluir que el modelo **no se ha sobreajustado**

### Finalizar Paralelizacion

```
# parallel::detectCores(logical=FALSE)
# cl <- makePSOCKcluster(4)
# registerDoParallel(cl)
parallel::stopCluster(cl) ## Esto se debe ejecutar al final
```

---

## Análisis Posteriores

#### ¿Qué variables parecen estar más relacionadas con el abandono del cliente?

```
library(vip)
```

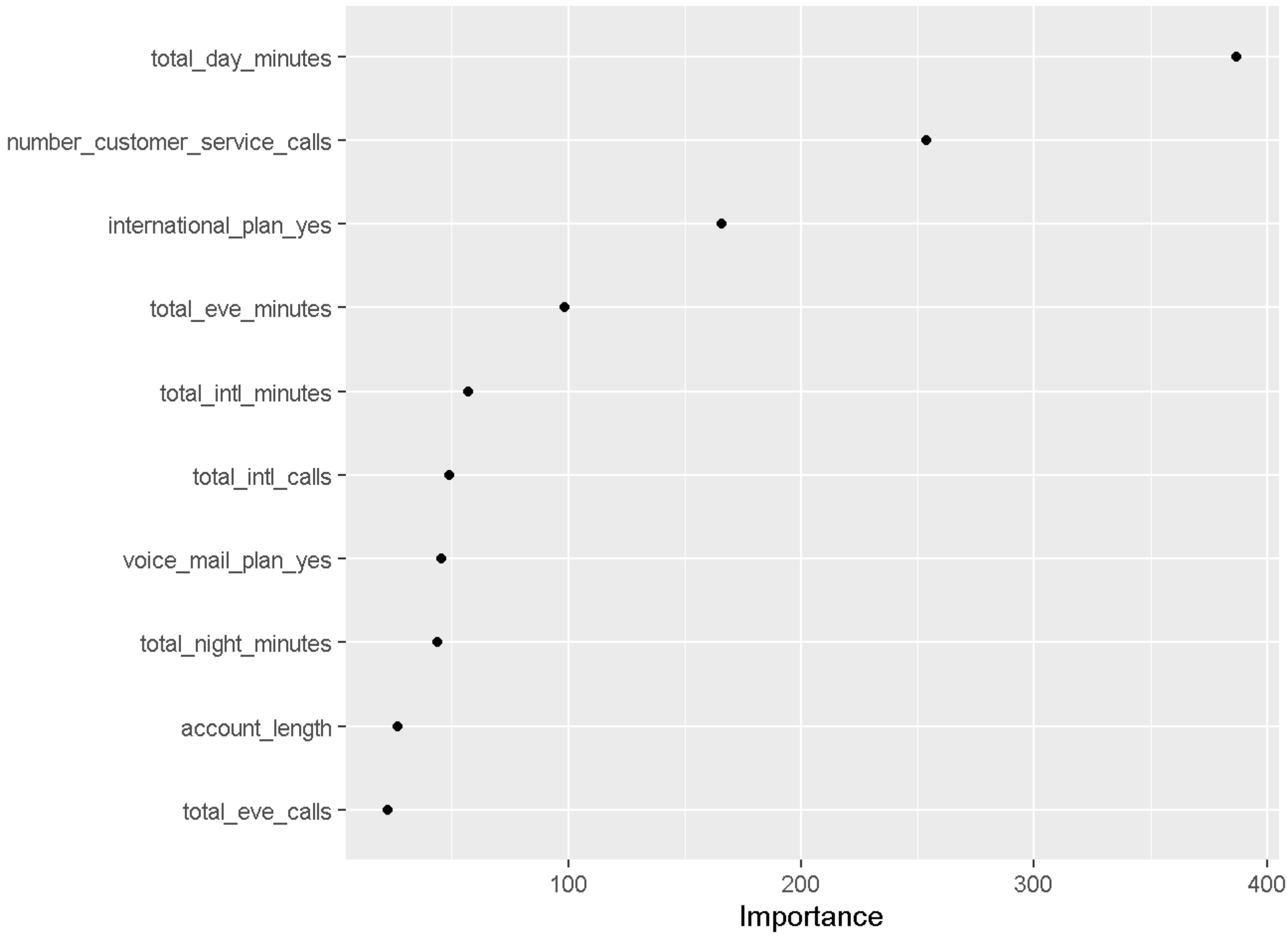
```
Warning: package 'vip' was built under R version 4.0.5


Attaching package: 'vip'

The following object is masked from 'package:utils':

    vi
```

```
rf_model_fin %>%
  vip(geom = "point")
```


total_day_minutes
number_customer_service_calls
international_plan_yes
total_eve_minutes
total_intl_minutes
total_intl_calls
voice_mail_plan_yes
total_night_minutes
account_length
total_eve_calls
100
200
300
400
Importance