



# Práctica 1

## Preprocesamiento de datos y clasificación binaria

Sistemas Inteligentes para la Gestión de la Empresa

Pedro Manuel Gómez-Portillo López

[gomezportillo@correo.ugr.es](mailto:gomezportillo@correo.ugr.es)

23 de abril de 2019

# Índice

<b>1. Introducción</b>	<b>6</b>
<b>2. Exploración</b>	<b>7</b>
<b>3. Preprocesamiento</b>	<b>8</b>
<b>4. Clasificación</b>	<b>12</b>
4.1. Random Forest	13
4.2. SVM	16
4.2.1. SVM lineal	17
2.4.2. SVM radial	19
4.3. CART	20
<b>5. Resultados</b>	<b>23</b>
<b>6. Conclusiones</b>	<b>25</b>
<b>7. Webgrafía</b>	<b>26</b>

# 1. Introducción

Kaggle es una comunidad online propiedad de Google de científicos de datos. Esta comunidad permite a sus usuarios encontrar y publicar conjuntos de datos, explorar y construir modelos en un entorno de ciencia de datos basado en la web, trabajar con otros científicos e ingenieros de aprendizaje automático y acceder en competiciones para resolver desafíos relacionados con este área<sup>1</sup>.



Logo de Kaggle

Una de estas competiciones, organizada por el banco Santander, es *Santander Customer Transaction Prediction*<sup>2</sup>, que promete premios de un total de 65.000\$ para los equipos con los mejores resultados. Esta competición proporciona un conjunto de datos de las transacciones con 200.000 clientes y 200 variables cada uno y está orientada a identificar qué clientes realizarán una transacción en el futuro, independientemente del dinero.

El objetivo de esta práctica será conseguir un resultado competente para esta competición usando RStudio.

Cabe decir que, aunque a fecha de realizar esta práctica la competición ya ha terminado y los ganadores describieron cómo habían llegado a sus resultados<sup>3,4</sup>, se ha decidido no consultarlos para obtener un resultado propio.

Los bloques de código se presentarán en gris y las salidas por consola en azul.

Aunque en esta documentación se presentarán las partes más importantes del código, la versión completa puede consultarse en el repositorio del proyecto,

<https://github.com/gomezportillo/sige>

---

<sup>1</sup> <https://en.wikipedia.org/wiki/Kaggle>

<sup>2</sup> <https://www.kaggle.com/c/santander-customer-transaction-prediction/overview>

<sup>3</sup> <https://www.kaggle.com/c/santander-customer-transaction-prediction/discussion/89003>

<sup>4</sup> <https://www.kaggle.com/c/santander-customer-transaction-prediction/discussion/88939>

## 2. Exploración

Uno de los factores que hace que esta competición tan críptica es la nula explicación por parte de su organizador de los datos; de hecho, en la propia discusión de la página de Kaggle hay un par de hilos en los los *kagglers* se quejan por ello. Por tanto, el primer paso que se dará será la exploración de los mismos.

Una vez cargados los datos en RStudio usaremos las funciones `glimpse`, `summary`, y `df_status` para empezar a entender los datos.

Tras ejecutarlas, la información obtenida más importante es que,

- Existe una columna inútil para nosotros, *ID\_code*, que tendremos que borrar.
- La variable a predecir, *target*, tiene un 90.99% de valores a 0, por lo que deberemos llevar a cabo un análisis desbalanceado.
- La mayoría de variables tiene al menos un 0.01% de valores NA<sup>5</sup>.

---

<sup>5</sup> Not Available

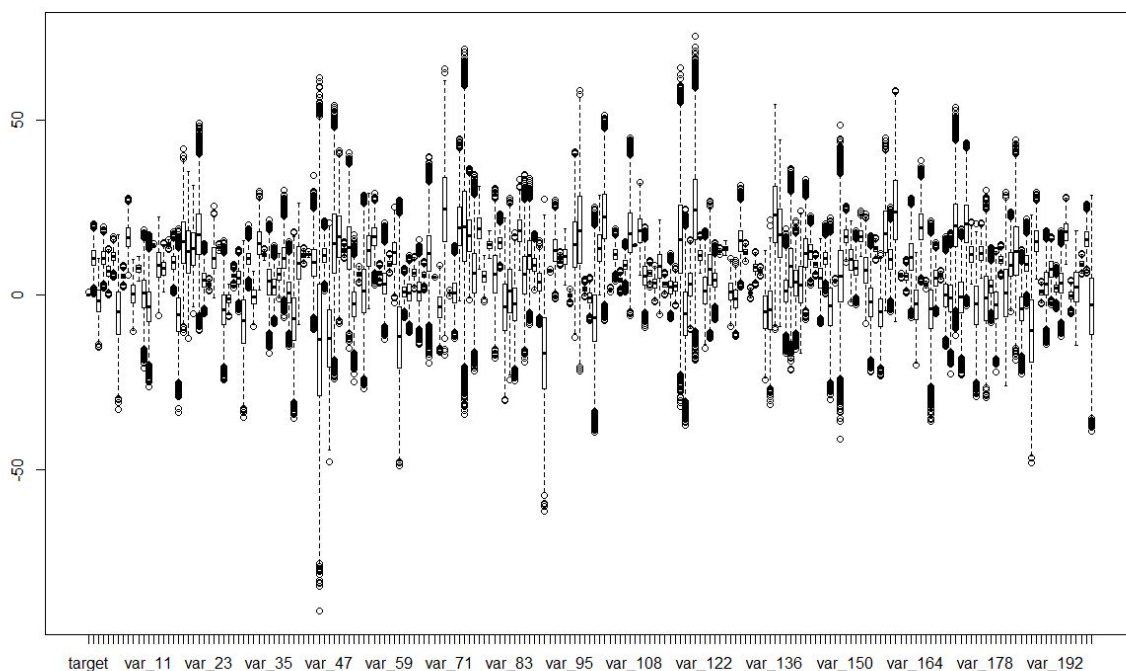
### 3. Preprocesamiento

Una vez que ya entendemos a qué datos nos estamos enfrentando, pasaremos a preprocesarlos.

Lo primero que haremos serán tareas básicas como eliminar las columnas que no necesitamos, en este caso *ID\_code*. Luego, en un intento de reducir la cantidad de datos que tenemos, eliminaremos las filas con valores NA.

```
data <- select(data_raw, -ID_code)
data <- na.omit(data)
```

Lo siguiente que haremos será calcular el diagrama de cajas de los datos, que puede verse a continuación y, como puede verse, hay bastante dispersión en los datos. Además, con el comando `boxplot(data)$stats` pueden verse la estadísticas del mismo, como los outliers.



Ahora, como lo que intentamos es predecir la variable *target*, pasaremos a eliminar las variables que estén muy débilmente correladas con ella. Para ello, primero construiremos la matriz de correlación del conjunto de datos y la usaremos para detectar las variables que no tengan apenas relación con nuestra variable objetivo.

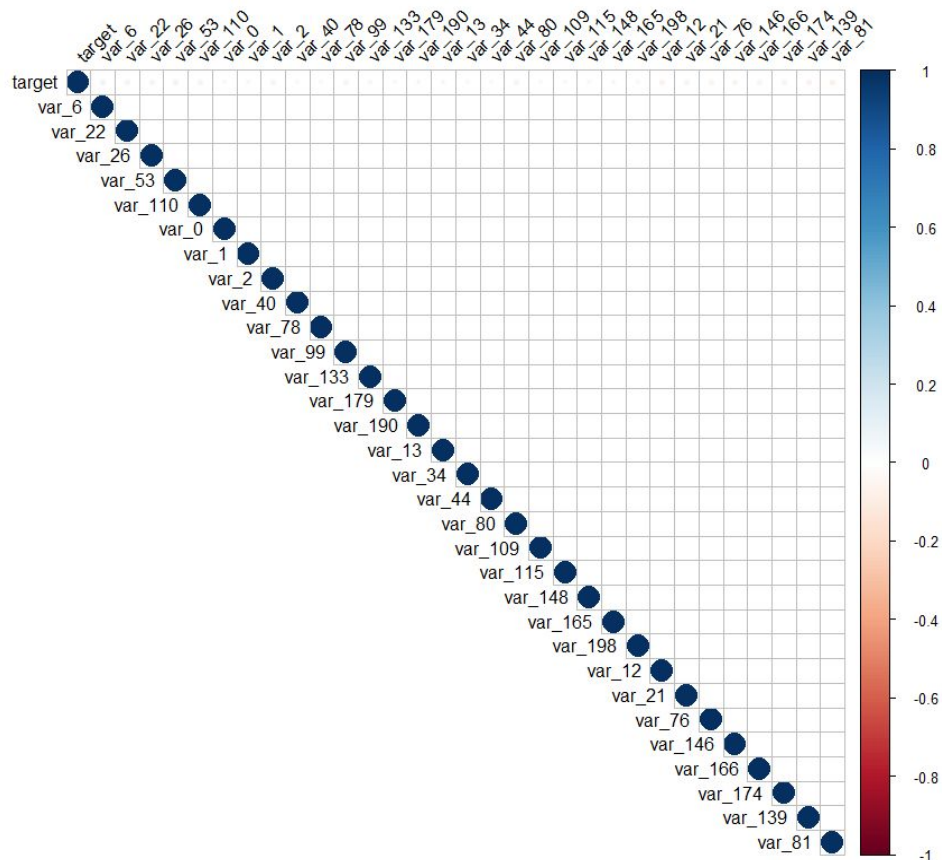
Tras ello, eliminaremos las filas con menos de un 0.05 de relación en valor absoluto con la variable objetivo.

```
## Calcular la correlación entre las variables
data_num <- data %>%
  na.exclude() %>%
  mutate_if(is.character, as.factor) %>%
  mutate_if(is.factor, as.numeric)

cor_target <- correlation_table(data_num, target = 'target')
correlated_vars <- cor_target %>%
  filter(abs(target) >= 0.05)

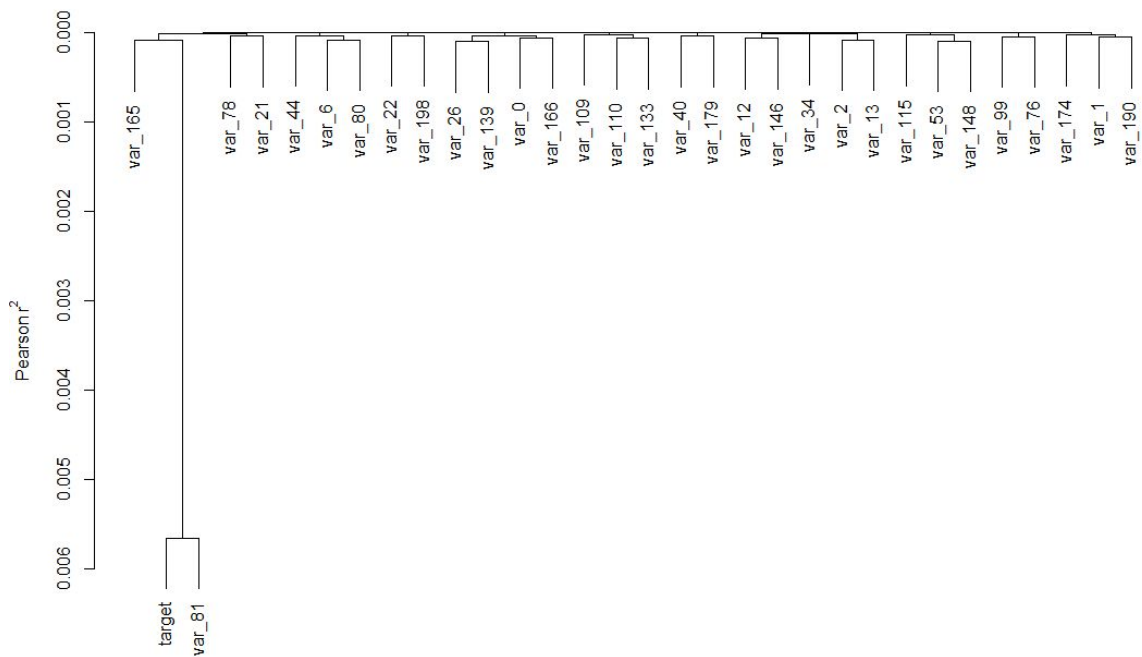
## Cuando tenemos las columnas las seleccionamos del dataset
data_num <- data_num %>%
  select(one_of(correlated_vars$Variable))
```

A continuación se adjunta la matriz de correlación de las 32 variables resultantes.



Como no se aprecian del todo bien los colores, también se adjunta el árbol de correlación, en el que puede verse cómo la variable más relacionada con *target* es *var\_81*.

```
v <- varclus(as.matrix(data_num), similarity = "pearson")
plot(v)
```



Ahora utilizaremos de detección de ruido para limpiar aún más los datos usando el paquete `NoiseFiltersR` y la función `AENN`<sup>6</sup>.

```
noise_filter <- AENN(target ~ ., data_num)
summary(noise_filter)
identical(noise_filter$cleanData,
          data_num[setdiff(1:nrow(data_num),
                           noise_filter$remIdx), ])
```

O al menos esa era la teoría; aún dejando la función `AENN` 24 horas seguidas con los datos preprocesados (es decir, solo con 32 variables) no se consiguió que terminara de ejecutarse, y al necesitar el ordenador para poder seguir trabajando en la práctica se decidió prescindir a efectos prácticos del tratamiento del ruido, aunque a nivel teórico estaba todo preparado para ello.

Tras el fracaso del tratamiento del ruido se valoró utilizar un servicio IaaS como Azure o AWS para dejar la función ejecutando en paralelo mientras se seguía trabajando en la práctica, pero como los créditos gratis que tenemos son limitados y los necesitamos para otras asignaturas finalmente se ha tenido que optar por no hacerlo.

<sup>6</sup> <https://www.rdocumentation.org/packages/NoiseFiltersR/versions/0.1.0/topics/AENN>

Por último sustituiremos los valores 0 y 1 de la columna *target* por 'Yes' y 'No' como pide el enunciado.

```
data_num <-  
  data_num %>%  
  mutate(target = as.factor(ifelse(target == 1, 'Yes', 'No')))
```

Finalmente se han guardado los datos ya preprocesados en un archivo para poder leerlos directamente sin tener que volver a ejecutar el preprocesamiento cada vez que se necesite.

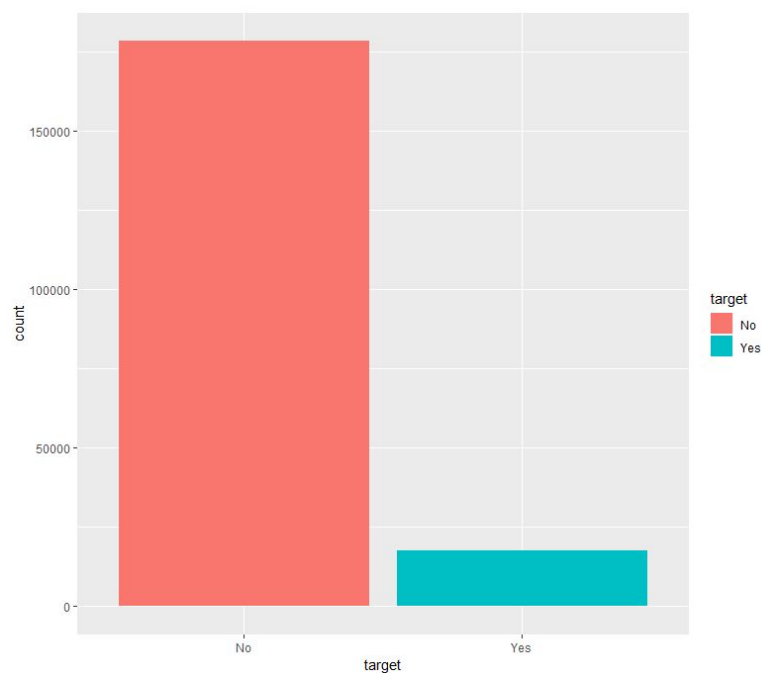
```
write_csv(data_num, "out/processed.csv")
```



## 4. Clasificación

Una vez que ya tenemos los datos preprocesados, pasaremos a clasificarlos. Aunque ya hemos visto en la exploración que nos enfrentamos a datos desbalanceados, mostraremos el porcentaje de unos y ceros en un diagrama para verlo de manera más gráfica.

```
ggplot(data) +  
  geom_histogram(aes(x = target, fill = target), stat = 'count')
```



A la vista del gráfico podemos elegir dos opciones; o bien balancear los datos prescindiendo de una buena parte del conjunto predominante o bien llevar a cabo una clasificación desbalanceada. Para este proyecto se ha elegido la segunda opción, ya que ha sido lo estudiado en clase.

Lo primero que haremos será crear dos particiones de datos aleatorias, una para entrenamiento y otra para comprobación. Además, se utilizará la función `now()`<sup>7</sup> del paquete `lubridate` para medir el tiempo que ha tardado en ejecutarse.

---

<sup>7</sup> <https://www.rdocumentation.org/packages/lubridate/versions/1.7.4/topics/now>

```
part_index <- createDataPartition(data$target,
                                   p = .75,
                                   list = FALSE,
                                   times = 1)

train_data <- data[part_index,]
val_data    <- train_data[-part_index,]
```

Esto lo haremos así para que las dos técnicas que utilizemos comparen los mismos datos y así podamos comparar los resultados de manera más fiable.

Por el mismo motivo, ambas técnicas compartirán la misma configuración, que puede verse a continuación.

```
train_Ctrl <- trainControl(
  verboseIter = F,
  classProbs = TRUE,
  method = "repeatedcv",
  number = 5,
  repeats = 1,
  summaryFunction = twoClassSummary
)

train_TuneGrid <- expand.grid(.mtry = c(sqrt(ncol(train_data))))
```

- **Train control.** Sirve para controlar la ejecución del método `train`. Se utilizará el método de resampling `repeatedcv`, que dividirá el conjunto de datos de entrenamiento en 5 partes al azar y luego usará cada una de esas partes como conjunto de datos de prueba para el modelo entrenado en otras 9, y tomará la media de los resultados obtenidos<sup>8</sup>. Por otro lado, el argumento `summaryFunction` se usa para pasar una función que toma los valores observados y predichos y estima una medida del rendimiento<sup>9</sup>.
- **Tune grid.** Este parámetro nos permite decidir qué valores tomará el parámetro principal<sup>10</sup>.

Se usará `caret` para entrenar los modelos, por lo que entre todas las técnicas que tiene<sup>11</sup> se ha elegido utilizar Random Forest y SVM.

<sup>8</sup> <https://discuss.analyticsvidhya.com/t/what-is-repeated-cv-in-caret/12222/2>

<sup>9</sup> <https://cran.r-project.org/web/packages/caret/vignettes/caret.html>

<sup>10</sup> [http://www.rpubs.com/Mentors\\_Ubiquum/tunegrid\\_tunelength](http://www.rpubs.com/Mentors_Ubiquum/tunegrid_tunelength)

<sup>11</sup> <http://topepo.github.io/caret/train-models-by-tag.html>

## 4.1. Random Forest

La primera técnica de clasificación que utilizaremos será un Random Forest. Este algoritmo fue desarrollado por Leo Breiman y Adele Cutler. Dicho término aparece de la primera propuesta de *Random decision forests*, hecha por Tin Kam Ho en 1995. Es un algoritmo predictivo que usa la técnica de *bagging* para combinar diferentes árboles donde cada uno de ellos es construido con observaciones y variables aleatorias<sup>12,13</sup>.

Aunque inicialmente se pensó en utilizar la implementación del paquete de R `RandomForest`<sup>14</sup>, finalmente se ha optado por utilizar el paquete `caret` como se ha visto en clase, ya que permite variar cómodamente de técnica de entrenamiento. Construiremos el modelo de la siguiente forma, usando curvas ROC como métrica para representar la sensibilidad como se vió en clase.

```
time_prev <- now()

rfModel <-
  train(
    target ~ .,
    data = train_data,
    method = "rf",
    metric = "ROC",
    trControl = train_Ctrl
    tuneGrid = train_TuneGrid
  )

(elapsed_time <- now() - time_prev)
```

Tras entrenar el modelo durante ~57 minutos se obtuvo el siguiente resultado al imprimirlo por pantalla.

---

<sup>12</sup> <http://apuntes-r.blogspot.com/2014/11/ejemplo-de-random-forest.html>

<sup>13</sup> [https://es.wikipedia.org/wiki/Random\\_forest](https://es.wikipedia.org/wiki/Random_forest)

<sup>14</sup> <https://www.rdocumentation.org/packages/randomForest/versions/4.6-14/topics/randomForest>

```
Random Forest

147031 samples
  31 predictor
  2 classes: 'No', 'Yes'

No pre-processing
Resampling: Cross-Validated (5 fold, repeated 1 times)
Summary of sample sizes: 117625, 117624, 117625, 117625, 117625
Resampling results:

ROC          Sens          Spec
0.7809761    0.9998804    0.004755515

Tuning parameter 'mtry' was held constant at a value of 5.656854
```

Tras ello, se ha usado la función `predict` con el modelo y la partición de validación para hacer una matriz de confusión como se indica en este<sup>15</sup> tutorial para poder obtener el porcentaje de *accuracy* del modelo, que será discutido en la sección [Resultados](#).

```
prediction_p <- predict(rfModel, val_data, type = "prob")
prediction_r <- predict(rfModel, val_data, type = "raw")
(conf_matrix <- confusionMatrix(prediction_r, val_data$target))
```

#### Confusion Matrix and Statistics

	Reference	
Prediction	No	Yes
No	44589	4392
Yes	5	24

**Accuracy : 0.9103**

95% CI : (0.9077, 0.9128)

No Information Rate : 0.9099

P-Value [Acc > NIR] : 0.386

Kappa : 0.0096

Mcnemar's Test P-Value : <2e-16

Sensitivity : 0.999888

Specificity : 0.005435

Pos Pred Value : 0.910333

Neg Pred Value : 0.827586

Prevalence : 0.909896

<sup>15</sup> <https://machinelearningmastery.com/how-to-estimate-model-accuracy-in-r-using-the-caret-package/>

```
Detection Rate : 0.909794
Detection Prevalence : 0.999408
Balanced Accuracy : 0.502661

'Positive' Class : No
```

Además, para comparar los valores totales de las predicción contra resultado se ha construido una tabla de la siguiente forma.

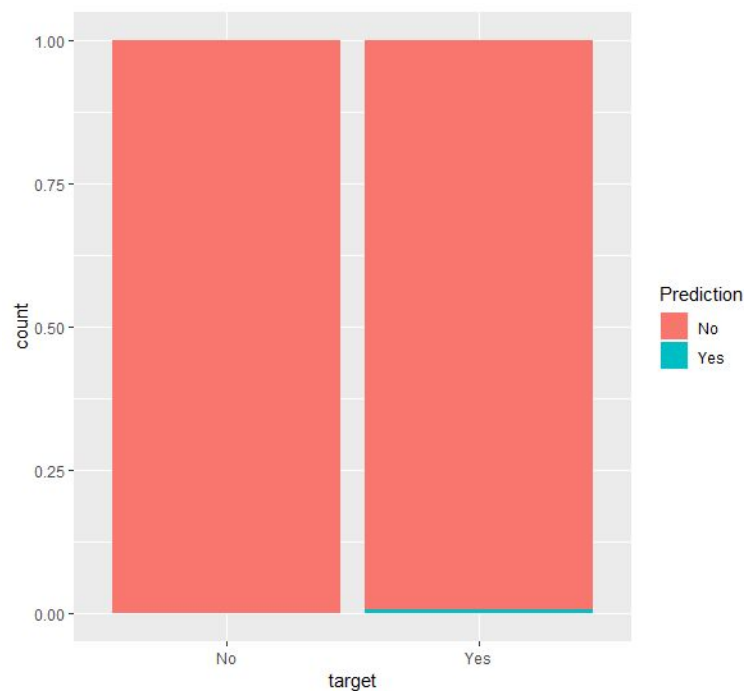
```
plotdata <- val_data %>%
  select(target) %>%
  bind_cols(prediction_p) %>%
  bind_cols(Prediction = prediction_r)

table(plotdata$target, plotdata$Prediction)
```

	No	Yes
No	44589	5
Yes	4392	24

Por último, se ha generado una gráfica a partir de la tabla anterior.

```
ggplot(plotdata) +
  geom_bar(aes(x = target,
               fill = Prediction),
           position = position_fill())
```



## 4.2. SVM

Las SVM o *Support Vector Machine* (Máquinas de vectores de soporte en español) son un conjunto de algoritmos de aprendizaje supervisado relacionados con problemas de clasificación y regresión desarrollados por Vladimir Vapnik y Alexey Chervonenkis en 1963<sup>16</sup> que permite construir un modelo que prediga la clase de una nueva muestra a partir de un conjunto de datos de entrenamiento.

### 4.2.1.SVM lineal

Inicialmente se utilizó la versión con el kernel lineal. Se ha usado la implementación del paquete `caret` de la siguiente forma.

```
time_prev <- now()

rfModel <-
  train(
    target ~ .,
    data = train_data,
    method = "svmRadial",
    metric = "ROC",
    trControl = train_Ctrl
  )

(elapsed_time <- now() - time_prev)
```

Pero tras más de 3 horas de entrenamiento la ejecución falló por no tener suficiente memoria RAM disponible con el siguiente error por pantalla.

```
Error: cannot allocate vector of size 133.3 Gb
```

A continuación se adjunta la captura de pantalla de RStudio.

---

<sup>16</sup> [https://en.wikipedia.org/wiki/Support-vector\\_machine#History](https://en.wikipedia.org/wiki/Support-vector_machine#History)

```

RStudio
File Edit Code View Plots Session Build Debug Profile Tools Help
~/
>
> # SVM lineal
>
> time_prev <- now()
>
> rfModel <-
+   train(
+     target ~ .,
+     data = train_data,
+     method = "svmLinear",
+     metric = "ROC",
+     trControl = train_ctrl
+   )
Error: cannot allocate vector of size 133.3 Gb
In addition: Warning messages:
1: model fit failed for Fold1.Rep1: C=1 Error : cannot allocate vector of size 85.3 Gb
2: model fit failed for Fold2.Rep1: C=1 Error : cannot allocate vector of size 85.3 Gb
3: model fit failed for Fold3.Rep1: C=1 Error : cannot allocate vector of size 85.3 Gb
4: model fit failed for Fold4.Rep1: C=1 Error : cannot allocate vector of size 85.3 Gb
5: model fit failed for Fold5.Rep1: C=1 Error : cannot allocate vector of size 85.3 Gb
6: In nominalTrainWorkflow(x = x, y = y, wts = weights, info = trainInfo, :

Error: cannot allocate vector of size 133.3 Gb
Show Traceback
Rerun with Debug

Timing stopped at: 2492 0.36 2492
>
> (elapsed_time <- now() - time_prev)
Time difference of 3.056901 hours
>
> saveRDS(rfModel, file = "out/svmLinearModel.rds")
Error in saveRDS(rfModel, file = "out/svmLinearModel.rds") :
  object 'rfModel' not found
> print(rfModel)
Error in print(rfModel) : object 'rfModel' not found
>
> prediction_p <- predict(rfModel, val_data, type = "prob")
Error in predict(rfModel, val_data, type = "prob") :
  object 'rfModel' not found
> prediction_r <- predict(rfModel, val_data, type = "raw")
Error in predict(rfModel, val_data, type = "raw") :
  object 'rfModel' not found
> (conf_matrix <- confusionMatrix(prediction_r, val_data$target))
Error in confusionMatrix(prediction_r, val_data$target) :
  object 'prediction_r' not found
>
> plotdata <- val_data %>%
+   select(target) %>%
+   bind_cols(prediction_p) %>%
+   bind_cols(prediction_r)
Error in dots_values(...): object 'prediction_p' not found
>
> table(plotdata$target, plotdata$prediction)
Error in table(plotdata$target, plotdata$prediction) :

```

Por ello, en un intento de reducir la cantidad de memoria principal que necesita R se volvió a ejecutar el algoritmo con la mitad de los datos, para lo que tras leerlos del archivo *processed.csv* se ejecutó el siguiente fragmento de código; es decir, en ese momento el vector data tenía 196,041\*32 variables.

```

# reducir datos al 50% para pruebas más rápidas
tmp_index <- createDataPartition(data$target,
                                p = .5,
                                list = FALSE,
                                times = 1)

data <- data[tmp_index,]

```

Desafortunadamente, tras otra hora de entrenamiento un error similar apareció.

```
Error: cannot allocate vector of size 33.3 Gb
```

```

RStudio
File Edit Code View Plots Session Build Debug Profile Tools Help
~/GitHub/SIGE/
> # tune grid lets us decide which values the main parameter will take
> train_TuneGrid <- expand.grid(.mtry = c(sqrt(ncol(train_data))))
> time_prev <- now()
>
> rfModel <-
+   train(
+     target ~ .,
+     data = train_data,
+     method = "svmLinear",
+     metric = "ROC",
+     trControl = train_Ctrl
+   )
Error: cannot allocate vector of size 33.3 Gb
In addition: warning messages:
1: model fit failed for Fold1.Rep1: C=1 Error : cannot allocate vector of size 21.3 Gb
2: model fit failed for Fold2.Rep1: C=1 Error : cannot allocate vector of size 21.3 Gb
3: model fit failed for Fold3.Rep1: C=1 Error : cannot allocate vector of size 21.3 Gb
4: model fit failed for Fold4.Rep1: C=1 Error : cannot allocate vector of size 21.3 Gb
5: model fit failed for Fold5.Rep1: C=1 Error : cannot allocate vector of size 21.3 Gb
6: In nominalTrainWorkflow(x = x, y = y, wts = weights, info = trainInfo, :
Error: cannot allocate vector of size 33.3 Gb
Timing stopped at: 753.3 0.26 759
>
> (elapsed_time <- now() - time_prev)
Time difference of 49.95645 mins
>
> saveRDS(rfModel, file = 'out/svmLinearModel.rds')
Error in saveRDS(rfModel, file = "out/svmLinearModel.rds") :
  object 'rfModel' not found
> print(rfModel)
Error in print(rfModel) : object 'rfModel' not found
>
> prediction_p <- predict(rfModel, val_data, type = "prob")
Error in predict(rfModel, val_data, type = "prob") :
  object 'rfModel' not found
> prediction_r <- predict(rfModel, val_data, type = "raw")
Error in predict(rfModel, val_data, type = "raw") :
  object 'rfModel' not found
> (conf_matrix <- confusionMatrix(prediction_r, val_data$target))

```

Tras esto se valoró seguir reduciendo el vector de datos hasta un 25% de su tamaño original, pero al usar tan pocos se temió obtener resultados poco fiables, por lo que se intentó cambiar el algoritmo de entrenamiento.

## 2.4.2.SVM radial

Tras fracasar con las SVM con kernels lineales se probó a ejecutar la versión con kernel radial de la siguiente forma.

```

time_prev <- now()

rfModel <-
  train(
    target ~ .,
    data = train_data,
    method = "svmRadial",
    metric = "ROC",
    trControl = train_Ctrl
  )

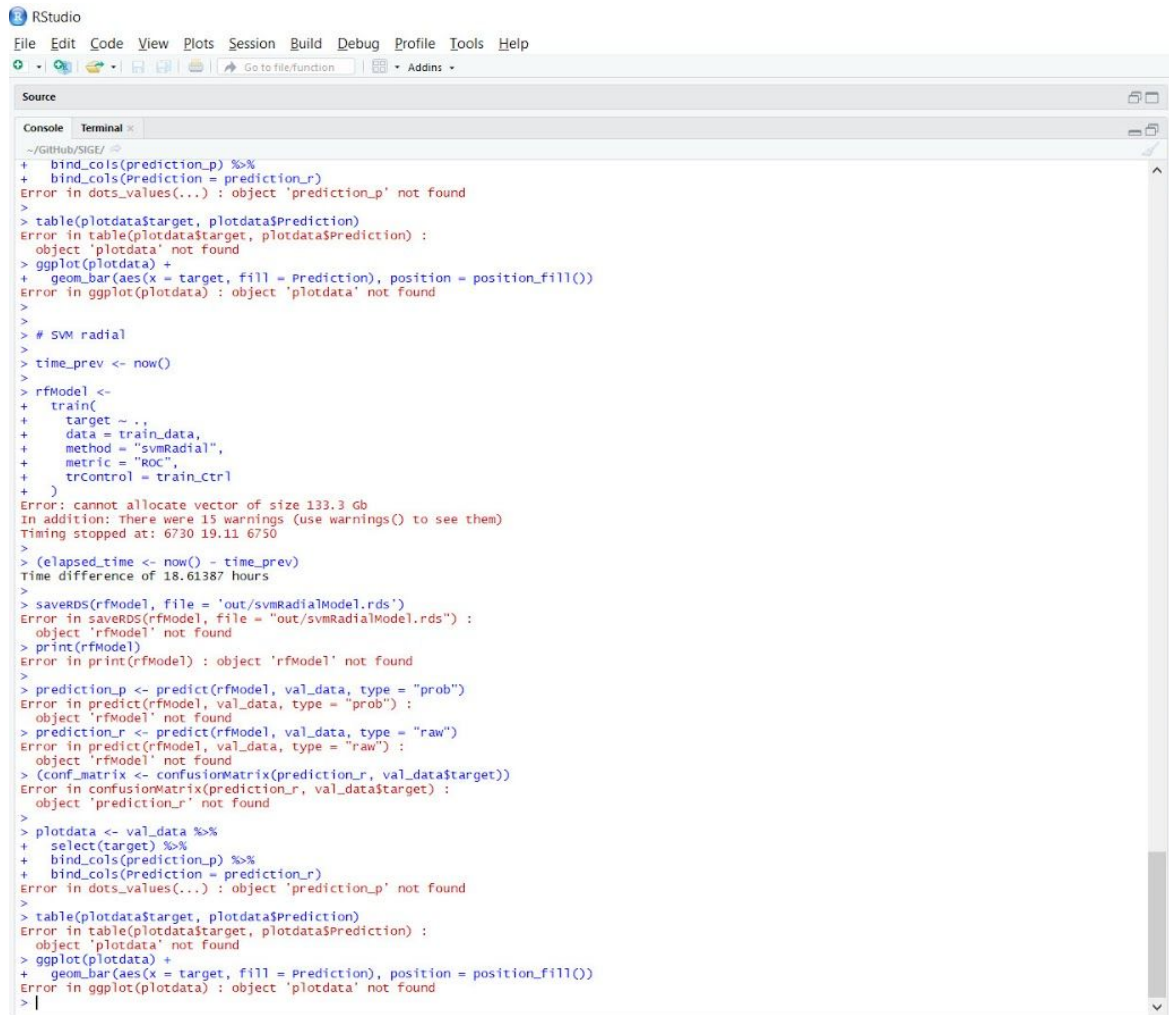
(elapsed_time <- now() - time_prev)

```



Pero de nuevo, tras casi 19 horas de entrenamiento se obtuvo el mismo error.

Error: cannot allocate vector of size 133.3 Gb



```
~/GitHub/SICE/
+ bind_cols(prediction_p) %>%
+ bind_cols(Prediction = prediction_r)
Error in dots_values(...): object 'prediction_p' not found
>
> table(plotdata$target, plotdata$Prediction)
Error in table(plotdata$target, plotdata$Prediction):
  object 'plotdata' not found
> ggplot(plotdata) +
+   geom_bar(aes(x = target, fill = Prediction), position = position_fill())
Error in ggplot(plotdata): object 'plotdata' not found
>
> # SVM radial
> time_prev <- now()
>
> rfModel <-
+   train(
+     target ~ ..
+     data = train_data,
+     method = "svmRadial",
+     metric = "ROC",
+     trControl = train_ctr1
+   )
Error: cannot allocate vector of size 133.3 Gb
In addition: There were 15 warnings (use warnings() to see them)
Timing stopped at: 6730 19.11 6750
>
> (elapsed_time <- now() - time_prev)
Time difference of 18.61387 hours
>
> saveRDS(rfModel, file = "out/svmRadialModel.rds")
Error in saveRDS(rfModel, file = "out/svmRadialModel.rds"):
  object 'rfModel' not found
> print(rfModel)
Error in print(rfModel): object 'rfModel' not found
>
> prediction_p <- predict(rfModel, val_data, type = "prob")
Error in predict(rfModel, val_data, type = "prob"):
  object 'rfModel' not found
> prediction_r <- predict(rfModel, val_data, type = "raw")
Error in predict(rfModel, val_data, type = "raw"):
  object 'rfModel' not found
> (conf_matrix <- confusionMatrix(prediction_r, val_data$target))
Error in confusionMatrix(prediction_r, val_data$target):
  object 'prediction_r' not found
>
> plotdata <- val_data %>%
+   select(target) %>%
+   bind_cols(prediction_p) %>%
+   bind_cols(Prediction = prediction_r)
Error in dots_values(...): object 'prediction_p' not found
>
> table(plotdata$target, plotdata$Prediction)
Error in table(plotdata$target, plotdata$Prediction):
  object 'plotdata' not found
> ggplot(plotdata) +
+   geom_bar(aes(x = target, fill = Prediction), position = position_fill())
Error in ggplot(plotdata): object 'plotdata' not found
> |
```

A estas alturas, aunque se valoró volver a ejecutar el modelo con la mitad de datos se temía volver a obtener un tiempo de entrenamiento desorbitadamente alto y un error parecido, por lo que se decidió probar un último método de entrenamiento.

## 4.3. CART

Los CART, o *Classification And Regression Trees*, son una técnica de aprendizaje de árboles de decisión no paramétrica que produce árboles de clasificación o regresión, dependiendo de si la variable dependiente es categórica o numérica, respectivamente<sup>17</sup>.

Para usarlos en el paquete `caret` deberemos usar el parámetro `rpart` como método.

```
time_prev <- now()

cartModel <-
  train(
    target ~ .,
    data = train_data,
    method = "rpart",
    metric = "ROC",
    trControl = train_Ctrl
  )

(elapsed_time <- now() - time_prev)
```

Al ejecutar el fragmento de código anterior, tras solo 2 minutos habremos entrenado el modelo. Ahora al ejecutar `print(cartModel)` obtendremos un resumen del mismo.

```
CART

147031 samples
  31 predictor
   2 classes: 'No', 'Yes'

No pre-processing
Resampling: Cross-Validated (5 fold, repeated 1 times)
Summary of sample sizes: 117625, 117624, 117625, 117625, 117625
Resampling results across tuning parameters:

   cp          ROC          Sens          Spec
0.0004528986  0.5370217  0.9985574  0.012831271
0.0004906401  0.5369563  0.9985648  0.012831300
0.0006416063  0.5276023  0.9991927  0.006641994
```

---

<sup>17</sup> *Análisis predictivo: técnicas y modelos utilizados y aplicaciones del mismo*, C. Espino, 2017

ROC was used to select the optimal model using the largest value. The final value used for the model was  $cp = 0.0004528986$ .

Ahora usaremos la matriz de confusión para obtener el porcentaje de aciertos del modelo sobre el conjunto de prueba.

```
prediction_p <- predict(cartModel, val_data, type = "prob")
prediction_r <- predict(cartModel, val_data, type = "raw")
(conf_matrix <- confusionMatrix(prediction_r, val_data$target))
```

#### Confusion Matrix and Statistics

	Reference	
Prediction	No	Yes
No	44562	4389
Yes	32	27

**Accuracy : 0.9098**

95% CI : (0.9072, 0.9123)

No Information Rate : 0.9099

P-Value [Acc > NIR] : 0.5354

Kappa : 0.0097

Mcnemar's Test P-Value : <2e-16

Sensitivity : 0.999282

Specificity : 0.006114

Pos Pred Value : 0.910339

Neg Pred Value : 0.457627

Prevalence : 0.909896

Detection Rate : 0.909243

Detection Prevalence : 0.998796

Balanced Accuracy : 0.502698

'Positive' Class : No

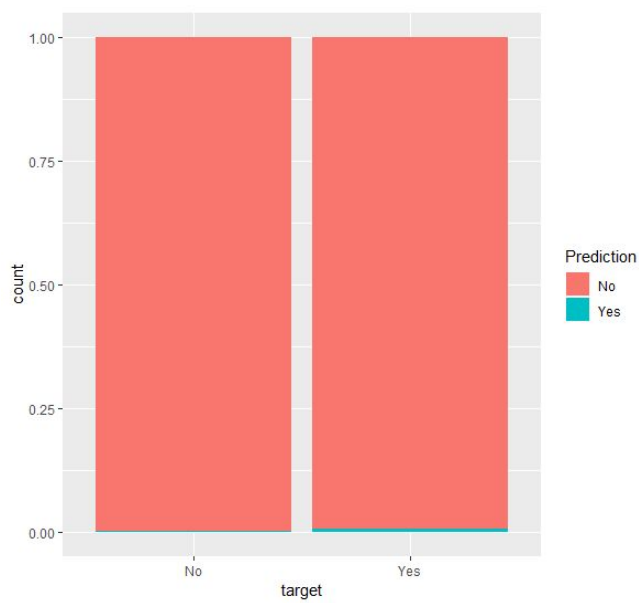
Por último usaremos los resultados obtenidos para generar un diagrama y poder verlos de manera más gráfica.

```
plotdata <- val_data %>%
  select(target) %>%
  bind_cols(prediction_p) %>%
  bind_cols(Prediction = prediction_r)

table(plotdata$target, plotdata$Prediction)
```

	No	Yes
No	44562	32
Yes	4389	27

```
ggplot(plotdata) +  
  geom_bar(aes(x = target, fill = Prediction), position =  
  position_fill())
```



## 5. Resultados

En esta sección se analizarán los resultados obtenidos de la ejecución de los algoritmos de entrenamiento.

Lo primero que se hará será resumir el *accuracy* y el tiempo de entrenamiento de todos los algoritmos ejecutados.

Técnica de clasificación	<i>Accuracy</i>	Tiempo de ejecución (min)
Random Forest	91.03%	~57
SVM lineal	/	~183
SVM radial	/	~1117
CART	90.98%	~2

Como solo hemos podido obtener resultados propiamente dichos usando Random Foresy y CART, a continuación compararemos las predicciones de síes y noes realizadas por cada uno.

Random Forest		Valores	
		No	Sí
Predicción	No	44589	5
	Sí	4392	24

CART		Valores	
		No	Sí
Predicción	No	44562	32
	Sí	4389	27

Como puede verse, ambos modelos fallan en lo mismo; debido seguramente a un sobreaprendizaje, tienden a predecir con mucha más frecuencia noes que síes, lo que explica su alto porcentaje de aciertos, ya que estábamos haciendo clasificación desbalanceada. Si aplicásemos estos modelos ya entrenados a un conjunto de datos balanceados, seguramente el porcentaje de aciertos sería bastante menor.

## 6. Conclusiones

Las técnicas y algoritmos de clasificación son un campo muy complejo y profundo y, aunque esta práctica solo me haya servido de introducción al mismo, me alegro de haberla realizado.

Además, creo que ha sido una buena manera de aplicar los conceptos aprendidos en la parte teórica de la asignatura, aunque el conjunto de datos elegido para la práctica haya sido demasiado grande, lo que ha hecho que algunos de los tiempos de entrenamiento hayan llegado a ser muy largos.

## 7. Webgrafía

Principales recursos web consultados, aparte de los mencionados a pie de página a lo largo de las secciones.

- Kaggle
  - <https://www.kaggle.com/>
- Ejemplo visto en clase de análisis predictivo de clases desbalanceadas
  - <https://github.com/jgromero/sige2019/blob/master/pr%C3%A1cticas/03.%20An%C3%A1lisis%20predictivo/loans-unbalanced.R>
- Documentación de R
  - <https://www.rdocumentation.org/>
- Documentación de `caret`
  - [http://topepo.github.io/caret/train-models-by-tag.html#Linear\\_Classifier](http://topepo.github.io/caret/train-models-by-tag.html#Linear_Classifier)
  - <http://caret.r-forge.r-project.org/>