VNiVERSiDAD
D SALAMANCA

Comparación de Algoritmos de *Machine Learning* para la Predicción de *Churn*: Boosting, Random Forest, Regresión Logística y Redes Neuronales

Álvaro Ramos Hernández

2023-05-02

Índice

1 Resumen

El análisis de *churn*, también conocido como tasa de cancelación o de abandono, es un aspecto en auge que desempeña un papel clave en la gestión de empresas que ofrecen servicios de suscripción. El análisis de *churn* es un proceso que tiene como objetivo comprender las razones y factores que llevan a los clientes a rescindir su contrato con la empresa. A partir de las conclusiones obtenidas, se desarrollan medidas proactivas como campañas de retención de clientes, dirigidas a aquellos en riesgo de abandonar, con el objetivo de fidelizarlos y evitar su pérdida.

En este estudio, se analiza la tasa de abandono de una entidad financiera a través del uso de datos de cancelación de clientes con el propósito de predecir cuales son aquellos individuos que se encuentran en riesgo de abandonar. Con ese objetivo se evalúan y comparan cuatro modelos predictivos: *Extreme Gradient Boosting*, *Random Forest*, Regresión Logística y Redes Neuronales. El estudio concluye que, para este problema en concreto, los mejores modelos son el *Boosting* y el *Random Forest*.

2 Introducción

Este análisis está enfocado a obtener un modelo que prioriza el *recall* frente a la precisión. El *recall* o exhaustividad nos indica la proporción de positivos capturados, por ello, en problemas como el que se está tratando en este estudio, interesa sacrificar precisión del modelo en beneficio de tener un mejor valor de exhaustividad.

Se debe sacrificar precisión ya que al tratar de aumentar el *recall* se captura una proporción mayor de casos positivos, lo que puede resultar en un mayor número de falsos positivos y, por tanto, una reducción en la precisión. En este análisis se presume que el perjuicio para la entidad financiera de capturar falsos positivos no es tan alto como el beneficio que se obtiene al tener un valor de exhaustividad alto.

El conjunto de datos utilizado en este estudio fue obtenido de la plataforma *Kaggle* (https://www.kaggle.com/datasets/shrutimechlearn/churn-modelling). El conjunto contiene la información de clientes de una entidad financiera, tanto de aquellos que rescindieron su contrato como los de los que siguen asociados a la entidad, con los cuales se tratará de predecir si un cliente está en riesgo de abandonar la empresa. El conjunto está compuesto por diez mil individuos, $n = 10000$, y catorce atributos, $p = 14$.

```
## [1] 10000    14
```

Las variables o atributos que componen el conjunto de datos son los siguientes.

```
##  [1] "RowNumber"       "CustomerId"      "Surname"         "CreditScore"
##  [5] "Geography"       "Gender"          "Age"             "Tenure"
##  [9] "Balance"         "NumOfProducts"   "HasCrCard"       "IsActiveMember"
## [13] "EstimatedSalary" "Exited"
```

donde

- ***RowNumber***: representa el índice de los individuos.
- ***CustomerId***: representa el id del cliente en el banco.
- ***Surname***: representa el apellido del cliente.
- ***CreditScore***: representa una medida numérica que califica la solvencia crediticia y el riesgo del cliente.

- ***Geography***: representa el país de la sucursal a la que esta asociada el cliente.
- ***Gender***: representa el sexo del cliente.
- ***Age***: representa la edad del cliente.
- ***Tenure***: representa en años la antigüedad o duración del vínculo entre el cliente y el banco.
- ***Balance***: representa en el valor monetario total de los activos y pasivos de un cliente en el banco.
- ***NumOfProducts***: representa el número de productos del banco utilizados por el cliente.
- ***HasCrCard***: representa la tenencia de trajeta de crédito.
- ***IsActiveMember***: representa si el cliente es un miembro activo del banco.
- ***EstimatedSalary***: representa el salario anual estimado del cliente.
- ***Exited***: representa la variable objetivo que se denominará *target*. Tendrá un valor de "0", si el cliente sigue en el banco, o de "1", si el cliente rescindió su contrato con la entidad financiera.

Se cambian los nombres de las variables para comprender mejor el conjunto de datos. Con las nuevas variables, los datos vienen estructurados conforme a como se muestra a continuación.

```
##   Índice       Id Apellido PuntuacionCredito   País Género Edad Antigüedad
## 1      1 15634602 Hargrave               619 France Female   42          2
## 2      2 15647311     Hill               608  Spain Female   41          1
## 3      3 15619304     Onio               502 France Female   42          8
## 4      4 15701354     Boni               699 France Female   39          1
## 5      5 15737888 Mitchell               850  Spain Female   43          2
## 6      6 15574012      Chu               645  Spain   Male   44          8
##     Balance NumProductos TarjetaCredito MiembroActivo   Salario target
## 1      0.00            1              1             1 101348.88      1
## 2  83807.86            1              0             1 112542.58      0
## 3 159660.80            3              1             0 113931.57      1
## 4      0.00            2              0             0  93826.63      0
## 5 125510.82            1              1             1  79084.10      0
## 6 113755.78            2              1             0 149756.71      1
```

Como se puede observar, las tres primeras variables, *Índice*, *Id* y *Apellido*, no influyen en la predicción del abandono, ya que en el caso de los dos primeros, son atributos únicos para cada cliente. En el caso del apellido, se considera que no existe una relación directa con el *Churn* ya que no hay ninguna información específica relacionada que pueda afectar. Por este motivo, se eliminan las tres primeras variables y, por consiguiente, los datos tienen ahora once atributos.

El análisis descriptivo de las variables numéricas independientes es el siguiente:

```
##  PuntuacionCredito      Edad          Antigüedad        Balance
##  Min.   :350.0     Min.   :18.00   Min.   : 0.000   Min.   :     0
##  1st Qu.:584.0     1st Qu.:32.00   1st Qu.: 3.000   1st Qu.:     0
##  Median :652.0     Median :37.00   Median : 5.000   Median : 97199
##  Mean   :650.5     Mean   :38.92   Mean   : 5.013   Mean   : 76486
##  3rd Qu.:718.0     3rd Qu.:44.00   3rd Qu.: 7.000   3rd Qu.:127644
##  Max.   :850.0     Max.   :92.00   Max.   :10.000   Max.   :250898
##   NumProductos  TarjetaCredito   MiembroActivo       Salario
##  Min.   :1.00   Min.   :0.0000   Min.   :0.0000   Min.   :    11.58
##  1st Qu.:1.00   1st Qu.:0.0000   1st Qu.:0.0000   1st Qu.: 51002.11
##  Median :1.00   Median :1.0000   Median :1.0000   Median :100193.91
##  Mean   :1.53   Mean   :0.7055   Mean   :0.5151   Mean   :100090.24
##  3rd Qu.:2.00   3rd Qu.:1.0000   3rd Qu.:1.0000   3rd Qu.:149388.25
##  Max.   :4.00   Max.   :1.0000   Max.   :1.0000   Max.   :199992.48
```

A continuación, se procede a la limpieza de la base de datos. Para ello se eliminan los individuos duplicados y con datos faltantes. En este caso no había ninguno así que se mantienen los diez mil individuos.

Tras la limpieza del conjunto de datos, antes de comenzar a trabajar con él, se debe preparar adecuadamente. Para ello se escalan los datos y se factorizan aquellos atributos que son de tipo *"character"*. En este caso son *País* y *Género*. Tras la factorización, las variables ahora serán de tipo "factor", sin embargo, algunos algoritmos que se utilizarán en este análisis necesitan que sean valores numéricos, por lo que es necesaria su conversión. Las clases de los atributos tras las conversiones resultan de la siguiente forma.

```
## PuntuacionCredito             País           Género             Edad
##         "integer"        "numeric"        "numeric"        "integer"
##        Antigüedad          Balance     NumProductos    TarjetaCredito
##         "integer"        "numeric"        "integer"        "integer"
##     MiembroActivo          Salario           target
##         "integer"        "numeric"        "integer"
```

La proporción de los valores de las clases objetivo en nuestra variable respuesta *Exited* es la siguiente:

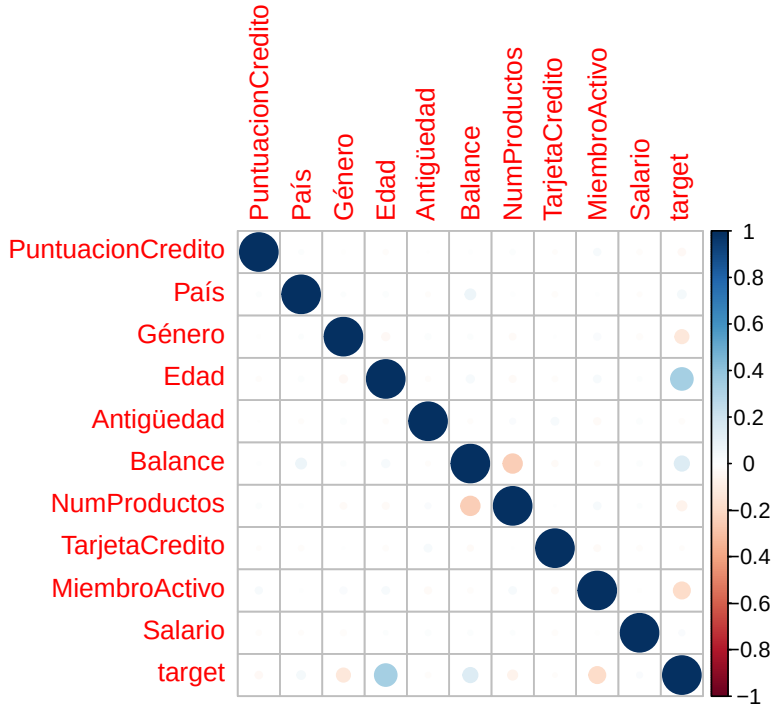
```
##
##      0      1
## 0.7963 0.2037
```

Como se observa, existe una disparidad significativa en la distribución de las clases objetivo. Esto puede tener un impacto negativo en el rendimiento de los modelos de clasificación, ya que el algoritmo puede sesgarse hacia la clase mayoritaria y tener dificultades para aprender patrones y características de las clases minoritarias.

Como el objetivo más importante de este análisis es precisamente clasificar correctamente los casos positivos, que es la clase minoritaria, se realiza un sub-muestreo de la clase mayoritaria eliminando cuatro mil individuos cuyo valor de la variable *Exited* es cero. El conjunto de datos cuenta en este momento con 6000 individuos. La distribución resultante está más equilibrada y los valores ahora son:

```
##
##      0      1
## 0.6605 0.3395
```

La relación entre las variables se pueden observar en el gráfico de correlación de abajo. La variable más correlacionada con la variable objetivo es la edad, positivamente correlacionada.



Antes de entrenar los diferentes modelos se llevó a cabo la división del conjunto de datos en tres conjuntos: conjunto de entrenamiento, 60% de los datos, conjunto de validación, 20% de los datos, y conjunto de prueba, 20% de los datos. El conjunto de entrenamiento se utilizará para crear los modelos, el de validación para buscar los hiperparámetros adecuados, y el de prueba para evaluar la precisión del modelo. Las dimensiones de los conjuntos son:

```
## [1] "Conjunto de entrenamiento:  3600  filas y  11  columnas."
```

```
## [1] "Conjunto de validación:  1200  filas y  11  columnas."
```

```
## [1] "Conjunto de prueba:  1200  filas y  11  columnas."
```

3 Métodos Estadísticos

En este estudio, se evalúan y comparan cuatro modelos predictivos de aprendizaje automático o, comúnmente denominado, *machine learning*. Alpaydin (2020) define el aprendizaje automático como la programación de ordenadores para optimizar un criterio de rendimiento utilizando datos de entrenamiento o experiencias pasadas.

El aprendizaje automático utiliza la teoría estadística en la construcción de los modelos matemáticos, y la computación en el entrenamiento con algoritmos eficientes para resolver los problemas de optimización, así como para procesar las grandes cantidades de datos (Alpaydin, 2020).

El proceso de utilizar algoritmos de *machine learning* para construir modelos a partir de los datos se denomina entrenamiento. El conjunto de los datos utilizados se denomina conjunto de entrenamiento. El proceso de hacer predicciones con un modelo entrenado se llama prueba y a los datos utilizados para ello conjunto de prueba (Zhou, 2021).

Este estudio aborda un problema de clasificación en el que hay dos clases: *"Evento"* o *"No Evento"*. La información sobre un cliente constituye la entrada para un clasificador, cuya tarea es asignar la entrada a una de las dos clases.

Para este análisis de la tasa de abandono se entrenan los modelos a través de los datos presentados anteriormente en la *introducción* (2) mediante el lenguaje de programación *R* con el entorno de desarrollo *RStudio*. En este problema, el clasificador asignará a los clientes una clase u otra en función de la variable *Exited*.

3.1 Boosting

El *boosting* es una metodología estadística moderna que mejora el rendimiento predictivo en la clasificación o regresión de los modelos de aprendizaje automático. Se originó a mediados de la década de 1990 y ha experimentado un gran progreso en las últimas décadas. Se caracteriza por el aprendizaje automático y la optimización (Bühlmann & Yu, 2010).

El *boosting* consiste en convertir clasificadores débiles en clasificadores robustos. El algoritmo más popular de boosting y del que parten el resto de algoritmos es el *AdaBoost*. A partir de un subconjunto de los datos de entrenamiento, el algoritmo comienza con el entrenamiento de un clasificador base débil (Schapire & Freund, 2012). A partir de los resultados de este primer clasificador, se ponderan los datos del subconjunto utilizado para el entrenamiento de modo que las muestras clasificadas incorrectamente reciban más atención por parte de los clasificadores posteriores. Una vez completado este paso, se entrena el segundo clasificador débil con otro subconjunto de entrenamiento que puede contener las muestras ajustadas anteriormente, y el resultado se utiliza para ajustar nuevamente las muestras del subconjunto actual. Este proceso se repite hasta que el número de clasificadores débiles alcance un valor predefinido. Finalmente, estos clasificadores débiles se ponderan y combinan según su rendimiento en el conjunto de entrenamiento, constituyendo de este modo, el clasificador robusto (Zhou, 2021).

Aunque se puede usar en regresión, el *boosting* es especialmente útil en problemas de clasificación, donde se busca predecir una variable objetivo categórica. Al construir modelos débiles y combinarlos de manera secuencial, el *boosting* tiene la capacidad de capturar relaciones más complejas entre las variables de entrada y la variable objetivo, mejorando la precisión de la predicción.

El algoritmo *AdaBoost*, es efectivo para problemas de clasificación binaria y se ha utilizado con éxito en una variedad de aplicaciones de aprendizaje automático, sin embargo, en este estudio, no se utilizará este algoritmo sino el *XGBoost*. Este algoritmo, el *Extreme Gradient Boosting*, se basa, como su propio nombre indica, en el boosting de gradiente regularizado. El entrenamiento se realiza utilizando una estrategia aditiva donde, dada una muestra i con un vector de descriptores x_i, el modelo utiliza K funciones aditivas

para predecir la salida $\widehat{y_i}$.

$$\widehat{y_i} = \phi(x_i) = \sum_{k=1}^{K} f_k(x_i), \qquad f \in \mathcal{F} \tag{1}$$

Aquí, $\mathcal{F}$ es el conjunto de todos los posibles árboles de regresión. La función f_k en cada uno de los k pasos mapea los valores de los descriptores en x_i a una determinada salida. Para aprender el conjunto de funciones utilizadas en el modelo, que contienen la estructura del árbol y las puntuaciones de las hojas, el objetivo es minimizar la expresión Eq. (2) buscando el valor adecuado para la norma del vector de puntuaciones $||\omega||$ de Eq. (3).

$$\mathcal{L}(\phi) = \sum_i l(\widehat{y_i}, y_i) + \sum_k \Omega(f_k) \tag{2}$$

donde

$$\Omega(f) = \gamma T + \frac{1}{2}\lambda ||\omega||^2 \tag{3}$$

En Eq. (2), l es una función de pérdida convexa diferenciable que mide la diferencia entre la predicción $\widehat{y_i}$ y el objetivo y_i. Se utiliza una aproximación de Taylor de segundo orden en esta función de pérdida. El segundo término, Ω, como se observa en Eq. (3), penaliza la complejidad del modelo en función del número de hojas, T, en el árbol. La regularización ayuda a suavizar los pesos finales aprendidos para evitar el sobreajuste y que el objetivo regularizado tienda a seleccionar un modelo que emplee funciones simples y predictivas. Además se añade regularización mediante el parámetro λ junto a la norma del vector de puntuaciones $||\omega||$, que es el valor de salida. XGBoost lo que busca es encontrar este valor de manera que el resultado de Eq. (2) sea mínimo. Este valor óptimo se calcula de manera distinta según si se está clasificando o realizando una regresión.

Para la regresión se calcula como:

$$\frac{\sum_i^n Residual_i}{n + \lambda} \tag{4}$$

donde n es el número de residuales.

Para la clasificación sería:

$$\frac{\sum_i^n Residual_i}{\sum [P_i \times (1 - P_i) + \lambda]} \tag{5}$$

Otras dos técnicas que tienen como objetivo reducir el sobreajuste son el encogimiento o *shrinkage* y el submuestreo de columnas. Después de cada paso del algoritmo, se ajustan los pesos recién agregados por un factor η para reducir la influencia de cada árbol y que el modelo aprenda lentamente e, idealmente, mejore. El submuestreo de columnas considera solo un subconjunto aleatorio de descriptores para construir un árbol dado. Esto también acelera el proceso de entrenamiento al reducir el número de descriptores a considerar. XGBoost también utiliza el *split-finding*, que es efectivo a la hora de controlar la dispersión para entrenar el modelo de manera más eficiente en datos dispersos (Chen & Guestrin, 2016).

3.2 Random Forest

El Random Forest es un algoritmo de aprendizaje automático que se utiliza tanto para clasificación como para regresión. Su nombre se debe a que combina múltiples árboles de decisión con muestras aleatorias para tomar una decisión final. La aleatoriedad permite que cada árbol tenga un enfoque ligeramente diferente y evita que los árboles se ajusten en exceso a patrones específicos (Breiman, 2001).

Antes de comenzar el algoritmo se debe indicar cuántos árboles se crearán y cuántas muestras se seleccionan aleatoriamente para entrenar cada árbol. El algoritmo consiste en que, para cada árbol, se selecciona aleatoriamente con reemplazo una muestra de tamaño "m" del conjunto de datos de entrenamiento con la que se construye dicho árbol. Para ello, se selecciona el mejor atributo para dividir los datos, utilizando algún criterio como la ganancia de información o el índice Gini. Posteriormente, se dividen los datos en subconjuntos más pequeños según el valor de la característica seleccionada y se continúa dividiendo recursivamente cada subconjunto hasta alcanzar una condición de parada (Liaw, Wiener, et al., 2002).

Una vez construidos todos los arboles, para cada individuo del conjunto de prueba, cada árbol genera una predicción individual basada en las características de la instancia y se le asigna a una de las clases. La clase mayoritaria en los árboles es la que se le asigna al individuo (Liaw, Wiener, et al., 2002).

3.3 Regresión Logística

La regresión es un algoritmo de aprendizaje supervisado que predice una variable en función de otras variables adicionales. La variable explicada se denomina variable dependiente o respuesta, mientras que las otras variables utilizadas para explicar la respuesta se denominan variables independientes. Muchas veces, las variables independientes se conocen como predictores (Hilbe, 2009).

El modelo logístico fue introducido a mediados del siglo XX y es ampliamente utilizado para predecir la probabilidad de que una variable categórica dependiente tome uno de los dos valores posibles. Utiliza una función logística, también conocida como sigmoide, para modelar la relación entre las variables independientes y la probabilidad de ocurrencia de un evento. La expresión es la siguiente:

$$g(z_i) = \frac{1}{1 + e^{-z_i}} \tag{6}$$

donde

$$z_i = \beta_0 + \beta_1 x_1 + \ldots + \beta_i x_i \tag{7}$$

En estas expresiones:

- $g(z_i)$ es la probabilidad de que la variable dependiente valga 1.
- β_0 es la constante del modelo o sesgo.
- β_i es el coeficiente de regresión asociado a la variable independiente i.
- x_i es el valor de la variable i.

El entrenamiento del modelo de regresión logística implica encontrar los valores óptimos para los coeficientes de regresión β_i que maximicen la verosimilitud de los datos observados. Este proceso se logra mediante el uso de técnicas de optimización, como el método de máxima verosimilitud (Hosmer Jr et al., 2013).

La verosimilitud es una medida de qué tan probable es que los datos observados se generen a partir del modelo propuesto. En el caso del modelo de regresión logística, la función de verosimilitud se define de la siguiente manera:

$$L(\beta_0, \beta_1, \ldots, \beta_p) = \prod_{i=1}^{n} \left(g(z_i)\right)^{y_i} \left(1 - g(z_i)\right)^{1-y_i} \tag{8}$$

El objetivo es maximizar esta función de verosimilitud, lo cual se puede hacer encontrando los valores de los coeficientes que maximizan el logaritmo natural de la verosimilitud, es decir, la expresión Eq. (9). Esto se puede lograr utilizando técnicas de optimización, como el descenso de gradiente o métodos más avanzados como el algoritmo de Newton-Raphson (Hosmer Jr et al., 2013).

$$l(\beta_0, \beta_1, \ldots, \beta_p) = \sum_{i=1}^{n} \left(y_i \log(g(z_i)) + (1 - y_i) \log(1 - g(z_i))\right) \tag{9}$$

El algoritmo itera y va maximizando la expresión hasta alcanzar los valores óptimos.

3.4 Redes Neuronales

Las redes neuronales se han convertido en la familia de algoritmos de *machine learning* más populares de estas últimas décadas. Como modelo computacional, existen desde mediados del siglo pasado, pero no ha sido hasta hace unos años, con la mejora de la técnica y la tecnología, que su uso se ha multiplicado exponencialmente.

Las redes neuronales son modelos compuestos por unidades de menor complejidad, las neuronas. Una neurona es una función matemática que utiliza todos los valores de entrada para realizar una suma ponderada de ellos sumado a un valor conocido como sesgo o *bias*, esta suma ponderada se denomina como Z. La ponderación de cada una de las entradas viene dada por el peso que se le asigna a cada una de las conexiones de entrada, estos pesos junto al sesgo, serán los valores que se pueden ajustar para que nuestra red neuronal aprenda y sea capaz de predecir. Una vez calculada la suma, el resultado pasa por una función $a(x)$ que deforma la salida de la suma ponderada, es la función de activación[1]. Por tanto la expresión para calcular la salida de una neurona es la siguiente (Santana, 2018).

$$y_i = a(b + \sum w_i x_i) \tag{10}$$

donde b es el sesgo, x_i los datos de entrada y w_i los pesos asignados.

[1] Es necesario deformar la salida de la neurona con una función no lineal ya que es demostrable que la suma de varias operaciones de regresión lineal es equivalente a una única operación y no tendría sentido tener más de una neurona.

Las neuronas forman capas dentro de la red neuronal de manera que cada neurona de la capa n recibe los mismos datos de entrada de la capa $n-1$ y pasarán sus cálculos a la capa $n+1$ que serán sus datos de entrada. La primera capa de la red neuronal se denomina capa de entrada, y la última, capa de salida, todas las capas entre estas dos son las capas ocultas (Nielsen, 2015).

El algoritmo para entrenar una red neuronal y reducir su función de coste C se llama *backpropagation* o retropropagación. Este algoritmo se basa en el descenso del gradiente, donde se ajustan los parámetros de un modelo de forma iterativa de manera que se reduce el valor de una función de costo. Para su cálculo, hay que realizar la derivada parcial de la función de coste respecto a cada parámetro del modelo, pesos y sesgo.

Sin embargo, en una red neuronal estas derivadas parciales no son tan sencillas de calcular, ya que la variación de un parámetro afectará a los resultados de las capas posteriores. Como indica el nombre del algoritmo, retropropagación se empieza derivando desde la capa de salida, la última. Suponiendo que se tienen L capas habrá que calcular:

$$\frac{\partial C}{\partial w^L} \qquad \frac{\partial C}{\partial b^L} \tag{11}$$

Para realizar la derivada parcial hay que descomponer, aplicando la regla de la cadena, la expresión en otras derivadas parciales para poder llegar del coste, a los pesos. La función del coste de la última capa resulta tras realizar el cálculo $C(a(Z^L))$, es decir, se hace la suma ponderada de los pesos, Z^L, se pasa por una función de activación $a(Z^L)$ y se calcula su coste $C(a(Z^L))$. Por tanto, las expresiones de Eq. (11) descompuestas quedaran de la siguiente manera.

$$\frac{\partial C}{\partial w^L} = \frac{\partial C}{\partial a^L} \cdot \frac{\partial a^L}{\partial z^L} \cdot \frac{\partial z^L}{\partial w^L} \qquad \frac{\partial C}{\partial b^L} = \frac{\partial C}{\partial a^L} \cdot \frac{\partial a^L}{\partial z^L} \cdot \frac{\partial z^L}{\partial b^L} \tag{12}$$

Ahora, con una función de coste y de activación determinada, simplemente hay que calcular como varía el coste según la función de activación, la función de activación respecto a la suma ponderada, y la suma ponderada respecto a los pesos o el *bias*. Las dos primeras derivadas de la descomposición nos indican cuanto varia el coste en función de la suma de la neurona, es lo que se denomina como el error imputado a la neurona, δ.

$$\frac{\partial z^L}{\partial b^L} = 1 \quad \frac{\partial z^L}{\partial b^L} = a_i^{L-1} \quad \frac{\partial C}{\partial z^L} = \delta^L \quad \Rightarrow \quad \frac{\partial C}{\partial w^L} = \delta^L a_i^{L-1} \quad y \quad \frac{\partial C}{\partial b^L} = \delta^L \tag{13}$$

Para generalizar esto al resto de capas hacia atrás, hay que tener en cuenta la δ de las capas posteriores multiplicándola a la derivada parcial de la función de activación de la capa que estamos derivando respecto a la suma ponderada y a la derivada parcial de la suma ponderada respecto al parámetro. Sin embargo, para llegar de la δ de una capa a la capa anterior hay que realizar la derivada parcial de la suma ponderada de la capa actual respecto a la función de activación de la capa anterior. La derivada para la capa $L-1$ quedaría de la siguiente forma:

$$\frac{\partial C}{\partial w^{L-1}} = \frac{\partial C}{\partial a^L} \cdot \frac{\partial a^L}{\partial z^L} \cdot \frac{\partial z^L}{\partial a^{L-1}} \cdot \frac{\partial a^{L-1}}{\partial z^{L-1}} \cdot \frac{\partial z^{L-1}}{\partial w^{L-1}} \tag{14}$$

$$\frac{\partial C}{\partial b^{L-1}} = \frac{\partial C}{\partial a^L} \cdot \frac{\partial a^L}{\partial z^L} \cdot \frac{\partial z^L}{\partial a^{L-1}} \cdot \frac{\partial a^{L-1}}{\partial z^{L-1}} \cdot \frac{\partial z^{L-1}}{\partial b^{L-1}} \tag{15}$$

De igual manera para la capa $L-2$ será:

$$\frac{\partial C}{\partial w^{L-1}} = \frac{\partial C}{\partial a^L} \cdot \frac{\partial a^L}{\partial z^L} \cdot \frac{\partial z^L}{\partial a^{L-1}} \cdot \frac{\partial a^{L-1}}{\partial z^{L-1}} \cdot \frac{\partial z^{L-1}}{\partial a^{L-2}} \cdot \frac{\partial a^{L-2}}{\partial z^{L-2}} \cdot \frac{\partial z^{L-2}}{\partial w^{L-2}} \tag{16}$$

$$\frac{\partial C}{\partial b^{L-1}} = \frac{\partial C}{\partial a^L} \cdot \frac{\partial a^L}{\partial z^L} \cdot \frac{\partial z^L}{\partial a^{L-1}} \cdot \frac{\partial a^{L-1}}{\partial z^{L-1}} \cdot \frac{\partial z^{L-1}}{\partial a^{L-2}} \cdot \frac{\partial a^{L-2}}{\partial z^{L-2}} \cdot \frac{\partial z^{L-2}}{\partial b^{L-2}} \tag{17}$$

(Chauvin & Rumelhart, 2013)

Y así sucesivamente para el resto de capas. De esta manera, se puede calcular el vector de gradientes, ∇f, que es lo que utilizará el algoritmo del descenso del gradiente para minimizar el error y mejorar nuestra red neuronal en la predicción del conjunto de entrenamiento e, idealmente, también en el de prueba.

La arquitectura óptima de una red neuronal depende del problema específico y de los datos manejados. Aumentar el número de capas y neuronas en una red neuronal puede aumentar su capacidad de aprendizaje y representación, lo que puede ser beneficioso para problemas complejos o con grandes conjuntos de datos. Sin embargo, un exceso de capas y neuronas puede llevar a problemas como el sobreajuste.

4 Resultados

El objetivo del entrenamiento de estos cuatro modelos es obtener el mejor en cuanto a exhaustividad pero que también sea lo más preciso posible. Para ello, se modifican los umbrales de clasificación y los hiperparámetros con el objetivo de conseguir un valor para la exhaustividad de al menos 0.9 en el conjunto de validación, es decir, se obtienen modelos que capturen al menos un 90% de los positivos verdaderos.

4.1 Boosting

Para la creación del modelo que hace uso de la técnica de *boosting*, en concreto *XGBoost*, se utiliza el código mostrado abajo. En este código se entrenan quinientos modelos distintos con el objetivo de encontrar el mejor posible respecto al *recall* y el mejor posible respecto al *accuracy*. Para ello, se modifican los hiperparámetros *nrounds* y *max.depth*. El umbral de clasificación, en este caso 0.432, ha sido elegido en función de las predicciones en el conjunto de validación para obtener una exhaustividad mayor a 0.9.

```
for (i in 1:5) {
  for (s in 1:100) {
    xgb <- xgb.train(data = data_train_Dmat,
                objective = "binary:logistic",
                nrounds = s,
                max.depth = i,
                eta = 0.3,
                nthread = 2)


    #Se predicen los resultados en el conjunto de entrenamiento.
    predicciones_train <- predict(xgb, data_train_Dmat)
```

```
    predicciones_train <- as.factor(predicciones_train > 0.432)
    levels(predicciones_train) <- c("No evento", "Evento")

    matriz_train <- confusionMatrix(predicciones_train,
                                    train_Y_f,
                                    positive = "Evento")


    #Se predicen los resultados en el conjunto de validación.
    predicciones_validation <- predict(xgb, data_validation_Dmat)

    predicciones_validation <- as.factor(predicciones_validation > 0.432)
    levels(predicciones_validation) <- c("No evento", "Evento")

    matriz_validation <- confusionMatrix(predicciones_validation,
                                         validation_Y_f,
                                         positive = "Evento")


    #Se recogen los valores del "Accuracy"  y "Recall" del modelo
    #entrenado en cada iteración cuando se predice el conjunto de
    #entrenamiento y validación.
    recall_train[i, s] <- matriz_train$byClass["Recall"]
    accuracy_train[i, s] <- matriz_train$overall["Accuracy"]

    recall_validation[i, s] <- matriz_validation$byClass["Recall"]
    accuracy_validation[i, s] <- matriz_validation$overall["Accuracy"]



    #Se guarda el modelo si tiene mayor "Recall" que el modelo anterior.
    if (matriz_validation$byClass["Recall"] > recall_max){
      modelo_xgb1 <-  xgb
      n_rounds1 <- s
      max_depth1 <- i
      accuracy_xgb1 <- matriz_validation$overall["Accuracy"]
      recall_max <- matriz_validation$byClass["Recall"]
    }


    #Se guarda el modelo si tiene mayor "Accuracy" que el modelo anterior.
    if (matriz_validation$overall["Accuracy"] > accuracy_max){
      modelo_xgb2 <-  xgb
      n_rounds2 <- s
      max_depth2 <- i
      accuracy_max <- matriz_validation$overall["Accuracy"]
    }
  }
}
```

Como el objetivo es encontrar un modelo que prioriza la exhaustividad pero se desea que tenga la mejor precisión posible, se puede modificar el umbral de clasificación del segundo modelo decrementándolo. Esto provocará un aumento del valor del *recall* pero una disminución del valor de precisión.

La estrategia implementada en el código que se muestra a continuación, consiste en disminuir el umbral de clasificación del segundo modelo, el que tenía mayor precisión pero menor exhaustividad, hasta que el valor del *recall* sea igual al del primer modelo, y comparar en ese momento que modelo tiene mayor precisión. El modelo que obtenga mayor precisión será el modelo seleccionado para realizar la prueba. En este caso, decrementando el umbral se consigue que el segundo modelo iguale en exhaustividad al primer modelo y tenga mejor precisión en las predicciones del conjunto de validación, por tanto, se seleccionará este modelo y su umbral para probar el modelo. El valor del umbral que se tendrá que usar en la prueba es 0.12.

```
umbral <- 0.44
recall_xgb2 <- 0

#Predicciones del segundo modelo en el conjunto de validación
pred_val_xgb2 <- predict(modelo_xgb2, data_validation_Dmat)

#Se decrementa el umbral de clasificación para el segundo modelo hasta
#que su recall iguale al del primer modelo
while (recall_max > recall_xgb2) {

    pred_val_xgb2_f <- as.factor(pred_val_xgb2 > umbral)
    levels(pred_val_xgb2_f) <- c("No evento", "Evento")

    matriz_validation_xgb2 <- confusionMatrix(pred_val_xgb2_f,
                                        validation_Y_f,
                                        positive = "Evento")

    recall_xgb2 <- matriz_validation_xgb2$byClass["Recall"]
    accuracy_xgb2 <- matriz_validation_xgb2$overall["Accuracy"]
    umbral <- umbral-0.01
}

#Se selecciona el modelo con mayor accuracy
if(accuracy_xgb1 > accuracy_xgb2 || umbral == 0.01){
  modelo_xgb <- modelo_xgb1
} else {
  modelo_xgb <- modelo_xgb2
}
```

El gráfico de la Figura (1) muestra como evoluciona la precisión del modelo cuando se predice el conjunto de entrenamiento, líneas azules, frente a cuando se predice el conjunto de validación, líneas rojas, en función del valor de los hiperparámetros *nrounds* y *max.depth*. Del mismo modo, la Figura (2) muestra la evolución del valor de la exhaustividad.

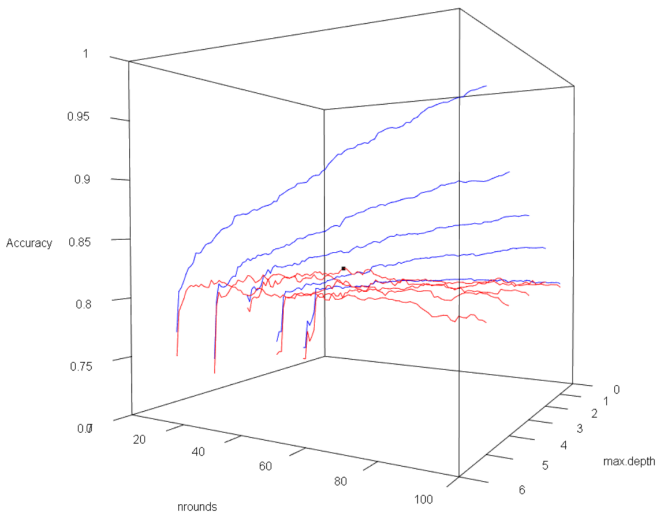


Figura 1: Búsqueda de hiperparámetros respecto al *accuracy*.

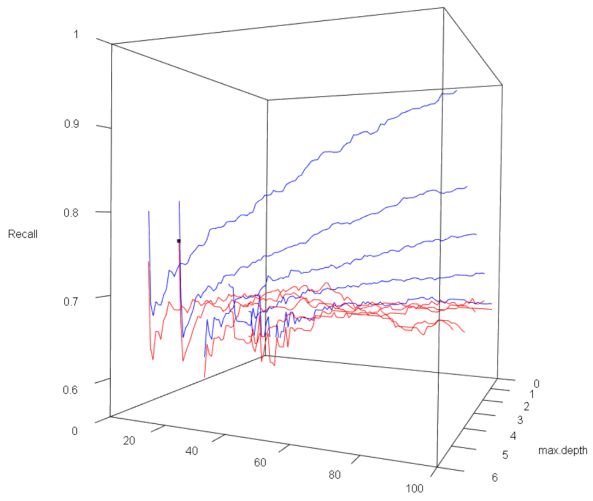


Figura 2: Búsqueda de hiperparámetros respecto al *recall*

Los puntos donde mayor precisión (Figura 1) y mayor recall (Figura 2) tiene el modelo al predecir el conjunto de validación, son los marcados en negro. Los hiperparámetros que entrenan el modelo con mayor precisión son $nrounds = 40$ y $max.depth = 3$ y para el modelo de mayor *recall*, $nrounds = 1$ y $max.depth = 4$.

Buscando los hiperparámetros de esta manera se consigue que el modelo no esté ni infraajustado ni sobreajustado, ya que como se observa en ambas imágenes, a mayor numero de rondas y profundidad, la precisión y la exhaustividad aumentan en el conjunto de entrenamiento pero, en el conjunto de validación, llega un punto en el que estos valores comienzan a disminuir.

Como se puede comprobar viendo los resultados de la predicción en el conjunto de validación, el segundo modelo es mejor.

```
##   Modelo Accuracy Recall Specificity Precision     F1
## 1   xgb1   0.5742 0.9107      0.4040    0.4359 0.5896
## 2   xgb2   0.6458 0.9156      0.5094    0.4855 0.6346
```

La predicción en el conjunto de prueba con el segundo modelo se realiza de la siguiente manera:

```
predicciones_xgb <- predict(modelo_xgb, data_test_Dmat)

predicciones_xgb_f = as.factor(predicciones_xgb > umbral)
levels(predicciones_xgb_f)=c("No evento", "Evento")

matriz_xgb <- confusionMatrix(predicciones_xgb_f, test_Y_f,
                              positive = "Evento")
```

Los resultados para la predicción del modelo en el conjunto de prueba son los siguientes.

```
##     Modelo Accuracy Recall Specificity Precision     F1
## 1 Boosting   0.6217 0.9109      0.4749    0.4682 0.6185
```

4.2 Random Forest

Para el modelo de *Random Forest* se procede de la misma manera que el caso anterior. En el código que se muestra a continuación, se entrenan veinte modelos distintos de los cuales se selecciona el mejor respecto a la precisión y el mejor respecto a la exhaustividad. Para ello, se modifica el hiperparámetro *ntree* y se establece el umbral de clasificación en 0.185.

```
for(i in seq(100,1000,50)){

  set.seed(100)
  RF <- randomForest(target ~.,
                     data = train_data,
                     ntree=i,
                     importance=TRUE)

  predicciones_RF <- predict(RF,validation_data)

  predicciones_RF_f = as.factor(predicciones_RF > 0.185)
  levels(predicciones_RF_f)=c("No evento", "Evento")

  matriz_RF <- confusionMatrix(predicciones_RF_f, validation_Y_f,
                               positive = "Evento")

  #Se guarda el modelo si tiene mayor "Recall" que el modelo anterior.
  if (matriz_RF$byClass["Recall"] > recall_max){
    modelo_RF1 <-  RF
    ntrees1 <- i
    accuracy_rf1 <- matriz_RF$overall["Accuracy"]
    recall_max <- matriz_RF$byClass["Recall"]
  }


  #Se guarda el modelo si tiene mayor "Accuracy" que el modelo anterior.
  if (matriz_RF$overall["Accuracy"] > accuracy_max){
    modelo_RF2 <-  RF
    ntrees2 <- i
    accuracy_max <- matriz_RF$overall["Accuracy"]
  }
}
```

De nuevo, se comprueba si es mejor el modelo de mayor exhaustividad, el primer modelo, o si resulta mejor el segundo modelo al bajar el umbral de clasificación. En este caso, nuevamente el mejor modelo resulta ser el segundo, se consigue con un umbral de 0.13.

```
umbral <- 0.19
recall_rf <- 0

pred_val_rf2 <- predict(modelo_xgb2, data_validation_Dmat)


while (recall_max > recall_rf) {
```

```
    pred_val_rf2_f = as.factor(pred_val_rf2 > umbral)
    levels(pred_val_rf2_f)=c("No evento", "Evento")

    matriz_validation_rf2 <- confusionMatrix(pred_val_rf2_f,
                                             validation_Y_f,
                                             positive = "Evento")

    recall_rf <- matriz_validation_rf2$byClass["Recall"]
    accuracy_rf2 <- matriz_validation_rf2$overall["Accuracy"]
    umbral <- umbral-0.01
}

if(accuracy_rf1 > accuracy_rf2 || umbral == 0.01){
  modelo_RF <- modelo_RF1
} else {
  modelo_RF <- modelo_RF2
}
```

Las predicciones del segundo modelo en el conjunto de prueba se realizan mediante el código mostrado a continuación.

```
predicciones_RF <- predict(modelo_RF,test_data)

predicciones_RF_f = as.factor(predicciones_RF > umbral)
levels(predicciones_RF_f)=c("No evento", "Evento")

matriz_RF <- confusionMatrix(predicciones_RF_f, test_Y_f,
                             positive = "Evento")
```

Los resultados de este modelo son los siguientes:

```
##          Modelo Accuracy Recall Specificity Precision     F1
## 1 Random Forest   0.5883 0.9381      0.4108    0.4469 0.6054
```

4.3 Regresión Logística

En este caso, el modelo de regresión logística, solo es posible construir uno. Para ello se utilizó el siguiente código.

```
set.seed(100)
modeloRegLog <- glm(target~., data=train_data, family=binomial)
```

Mediante el código que se muestra a continuación se modificó el umbral de clasificación hasta que se obtuvo un valor de exhaustvidad o *recall* mayor a 0.9 en las predicciones del conjunto de validación. En este caso, el umbral es 0.17.

```
predicciones_RegLog <- predict(modeloRegLog, validation_data, type="response")
predicciones_RegLog_f <- as.factor(predicciones_RegLog>0.17)
levels(predicciones_RegLog_f) <- c("No evento", "Evento")
```

```
matriz_RegLog <- confusionMatrix(predicciones_RegLog_f,
                                 validation_Y_f,
                                 positive = "Evento")
```

Las predicciones en el conjunto de prueba se realizan mediante el siguiente código.

```
predicciones_RegLog <- predict(modeloRegLog, test_data, type="response")
predicciones_RegLog_f <- as.factor(predicciones_RegLog>0.17)
levels(predicciones_RegLog_f) <- c("No evento", "Evento")


matriz_RegLog <- confusionMatrix(predicciones_RegLog_f,
                                 test_Y_f,
                                 positive = "Evento")
```

El resultado de la predicción del modelo con el conjunto de prueba es:

```
##          Modelo Accuracy Recall Specificity Precision     F1
## 1 Reg Logistica   0.5458 0.9132      0.3601    0.4191 0.5746
```

4.4 Redes Neuronales

La red neuronal entrenada consta de diez neuronas en la capa de entrada, tres neuronas en la capa oculta, y una neurona en la capa de salida. Para su entrenamiento se utilizó el código que se muestra a continuación.

```
set.seed(100)
nn <- neuralnet(target ~., data=train_data, hidden=3,linear.output = FALSE)
```

La estructura de la red neuronal es la siguiente, donde las conexiones en negro indican conexiones positivas entre las neuronas y las grises, negativas.

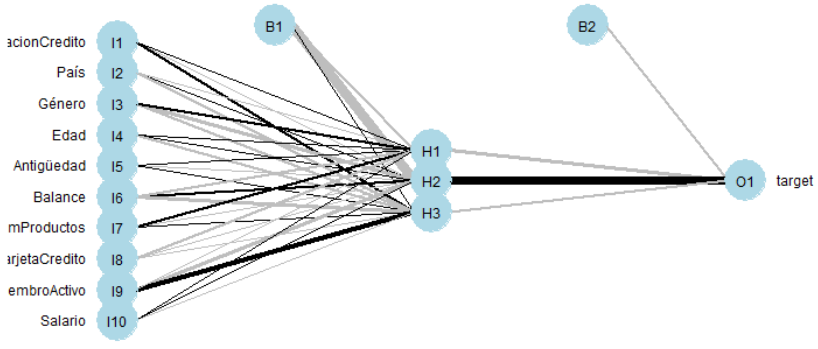


Figura 3: Estructura de la red neuronal

Con la red neuronal entrenada, se busca con el conjunto de validación el valor del umbral adecuado, en este caso 0.15. Las predicciones se realizan con el código mostrado a continuación.

```
nn_predicciones <- predict(nn, test_data)
nn_predicciones_f <- factor(nn_predicciones > 0.15)
levels(nn_predicciones_f) <- c("No evento", "Evento")


matriz_nn <- confusionMatrix(nn_predicciones_f, test_Y_f,
                             positive = "Evento")
```

Los resultados de este modelo son:

```
##         Modelo Accuracy Recall Specificity Precision     F1
## 1 Red Neuronal   0.5625 0.9156      0.3839    0.4291 0.5843
```

5 Discusión

En este estudio, se analiza el *churn* o tasa de abandono, lo que supone un aspecto crítico para las entidades financieras, ya que puede afectar su rentabilidad y crecimiento. Como se comentó anteriormente, el objetivo es encontrar un modelo que tenga un valor alto de exhaustividad o *recall* ya que es el valor que mide la capacidad del modelo de identificar correctamente los casos positivos. En el contexto de una entidad financiera, un caso positivo sería un cliente que está en riesgo de abandonar o dejar de utilizar los servicios financieros ofrecidos.

Es más importante tener un valor alto de exhaustividad que de precisión ya que el objetivo del análisis de *churn*, es identificar los casos positivos para poder tomar medidas proactivas para evitar su pérdida como podría ser desarrollar campañas de retención de estos clientes. El coste de capturar mayor número de positivos es capturar mayor número de falsos positivos, pero este coste es asumible, ya que si a estos falsos positivos se les aplican las medidas, como campañas de retención, no les causará ningún perjuicio. Lo único a tener en cuenta sería el coste que le provocaría a la empresa aplicar estas medidas a los falsos positivos.

Los resultados de las predicciones en el conjunto de prueba de todos los modelos son los siguientes:

```
##           Modelo Accuracy Recall Specificity Precision     F1
## 1       Boosting   0.6217 0.9109      0.4749    0.4682 0.6185
## 2 Random Forest   0.5883 0.9381      0.4108    0.4469 0.6054
## 3 Reg Logistica   0.5458 0.9132      0.3601    0.4191 0.5746
## 4  Red Neuronal   0.5625 0.9156      0.3839    0.4291 0.5843
```

El modelo con mayor exhaustividad es el que utiliza *Random Forest*, clasifica correctamente un 93,81% de los casos positivos. El modelo con mayor precisión es el que hace uso de *XGBoost*, clasifica bien el 62,17% de los datos.

El valor del "*Specifity*", especifidad, indica la proporción de verdaderos negativos capturados, en el caso de este estudio, representa la proporción de clientes que no tienen riesgo de abandonar la entidad financiera que se han clasificado correctamente. Para el primer modelo se capturan un 47.49% de los verdaderos negativos correctamente mientras que, para el segundo modelo, este valor baja hasta un 41.08% de los verdaderos negativos.

El valor de "*Precision*" indica que proporción de los positivos capturados son verdaderos positivos, lo que significa que para el primer modelo, un 46.82% de los casos clasificados como positivos son verdaderos positivos, y el resto, falsos positivos. Para el segundo modelo, este valor es de 44.69%.

"*F1*" es una métrica de evaluación que combina el valor de "*precision*" y el "*recall*" en un solo valor, proporcionando una medida balanceada del rendimiento de un modelo de clasificación. Se utiliza comúnmente cuando se busca un equilibrio entre la precisión y el recall. Un valor cercano a uno indica un modelo con una precisión y un recall muy buenos. Es especialmente útil cuando la precisión como el recall son igualmente importantes en el problema, por lo que, para este caso de análisis no es un valor determinante.

Para elegir cual es el mejor modelo dependerá del perjuicio le cause a la entidad financiera capturar falsos positivos. Si no tiene demasiado coste o el beneficio es superior a este coste, se eligirá el segundo modelo, *Random Forest*, ya que tiene mayor exhaustividad. Sin embargo, si sucede al contrario, se eligirá el primer modelo, *XGBoost*, que tiene mayor valor de "*Accuracy*" y "*Precision*".

6 Bibliografía

[1]. Alpaydin, E. (2020). *Introduction to machine learning*. MIT press.

[2]. Zhou, Z.-H. (2021). *Machine learning*. Springer Nature.

[3]. Schapire, R. E., & Freund, Y. (2012). *Boosting: Foundations and algorithms*. MIT press.

[4]. Yu, B. (2010). Boosting. *Wiley Interdisciplinary Reviews: Computational Statistics*, 2 (1), 69-74

[5]. Sheridan, R. P., Wang, W. M., Liaw, A., Ma, J., & Gifford, E. M. (2016). Extreme gradient boosting as a method for quantitative structure-activity relationships. *Journal of chemical information and modeling, 56* (12), 2353-2360.

[6]. Chen, T., & Guestrin, C. (2016). Xgboost: *A scalable tree boosting system. En Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining* (pp. 785-794).

[7]. *XGBoost Documentation — xgboost 2.0.0-dev documentation.* (s.f.). https://xgboost.readthedocs.io/en/latest/index.html

[8]. *XGBoost Part 3 (of 4): Mathematical Details* [Vídeo]. YouTube. https://www.youtube.com/watch?v=ZVFeW798-2I

[9]. Breiman, L. (2001). *Random forests*. Machine learning, 45, 5-32.

[10]. Liaw, A., Wiener, M., & others. (2002). *Classification and regression by randomForest*. R news, 2 (3), 18-22

[11]. Hilbe, J. M. (2009). *Logistic regression models*. CRC press.

[12]. Hosmer Jr, D. W., Lemeshow, S., & Sturdivant, R. X. (2013). *Applied logistic regression* (Vol. 398). John Wiley & Sons.

[13]. Nielsen, M. A. (2015). *Neural networks and deep learning* (Vol. 25). Determination press San Francisco, CA, USA.

[14]. Dot CSV. (2018). *¿Qué es una Red Neuronal? Parte 1: La Neurona | DotCSV* [Vídeo]. YouTube. https://www.youtube.com/watch?v=MRIv2IwFTPg

[15]. Chauvin, Y., & Rumelhart, D. E. (2013). *Backpropagation: theory, architectures, andapplications*. Psychology press.