TRABAJO DE FIN DE GRADO

Modelos de Aprendizaje Profundo. Aplicaciones con R y Python

Presentado por:
Raquel Beltrán Barba

Tutor:
D. RAFAEL PINO MEJÍAS

FACULTAD DE MATEMÁTICAS
Sevilla, Junio de 2022

Índice general

Resumen

La Inteligencia Artificial avanza desde hace unos años a pasos agigantados. El Aprendizaje Profundo (Deep Learning), una rama específica de ésta, no se queda atrás. Se ha desarrollado tanto que lo podemos aplicar en muchos los ámbitos de nuestra vida. Este aprendizaje es característico por estar basado en el funcionamiento de las redes neuronales que conforman el cerebro biológico. El análisis de estas redes y algunas aplicaciones concretas serán el objeto de estudio de este Trabajo de Fin de Grado.

El trabajo comienza con una introducción a la Inteligencia Artificial hasta llegar a la explicación del Aprendizaje Profundo, donde se habla de sus ventajas y aplicaciones en el día a día. Tras comentar brevemente la similitud con el cerebro humano, se pasa a la clasificación de los modelos según el aprendizaje y la topología. Explicamos también algunos conceptos relacionados. Antes de empezar a trabajar con los ejemplos, se hace una breve introducción a las librerías de los programas que se van a utilizar y sus correspondientes funciones. Los ejemplos que se resuelven son tres: Detección de transacciones fraudulentas con tarjetas de crédito, Predicción del precio del vino español y Clasificación de diferentes prendas a través de imágenes.

Abstract

Artificial Intelligence has taken quantum leaps these last years. Deep Learning, an especific branch of A.I., isn't too far behind. It has developed so much that it is useful in almost every aspect of our lives. This type of learning is special because it is based on how neural networks work inside biological brains. The analysis of these networks and its applications will be the subject of this Final Degree Project.

This dissertation starts by introducing Artificial Intelligence until reaching Deep Learning, where all its advantages and applications are explained. After a brief comment on its similarity with the biological brain, different neural networks are classified by it topology and Machine Learning methods. Some necessary concepts are also explained. Before we start working on some Deep Learning examples, a brief introduction on libraries and its functions used in Rstudio and Python are explained. We are solving three examples: Credit card fraud detection, Spanish wine price prediction and image clasification of different types of clothing.

Índice de figuras

Índice de cuadros

Capítulo 1

Introducción

1.1. De la Inteligencia Artificial al Aprendizaje Profundo

La creación de la Inteligencia Artificial surge de la necesidad de solucionar problemas de tratamiento y predicción de los datos. Se basa en una combinación de algoritmos que buscan imitar el funcionamiento de la conducta humana.

El único problema que la Inteligencia Artificial puede presentar, pero que el Machine Learning (también conocido como Aprendizaje Automático) si lo trabaja, es la capacidad de mejorar las tareas asignadas a través de la experiencia que consigue al tratar los datos. Lo que el Aprendizaje Automático hace es trabajar con un modelo que aprenda por sí mismo la estructura de los datos y consiga convertirlos de forma que nos dé un resultado apropiado. Este proceso de aprendizaje se mejora cuando contamos con la salida que los datos deberían proporcionar, así pudiendo este modelo comparar la salida esperada y la obtenida.

Aun así, el Aprendizaje Automático no llega al fondo de lo que significan los datos que recibe inicialmente, se centra en la primera o segunda "capa" de información. Esto presentaría un problema para datos complejos como pueden utilizarse en la clasificación de imágenes, la traducción de voz a texto o en la transcripción manual, para lo que necesitaríamos una forma de aprendizaje que ahonde aún más en estos datos: la solución se encontró en el Aprendizaje Profundo *(Deep Learning)*.

1.2. ¿Qué es el aprendizaje profundo?

El Aprendizaje Profundo es un subconjunto del Aprendizaje Automático que se basa en redes neuronales artificiales. Decimos que este aprendizaje es profundo porque al identificarse con la estructura de las redes de neuronas cuenta con varias capas de entrada, ocultas y de salida.
Cada una de estas capas contiene unidades que transforman los datos de entrada en la información que la siguiente capa debe utilizar para realizar la tarea asignada. Al contar con esta estructura la máquina puede aprender mientras ocurre el procesamiento de datos.

[15]Lo que hace especial al Aprendizaje Profundo dentro del Automático es lo siguiente:

- Necesita utilizar grandes cantidades de datos para hacer buenas predicciones.
- Depende de máquinas rápidas. El Aprendizaje Automático puede trabajar en equipos más lentos, mientras que el Profundo debe realizar muchas operaciones de multiplicación de matrices y tardaría mucho más.
- Una vez introducidos los datos, el Aprendizaje Profundo aprende sus características y crea nuevas automáticamente, mientras que el Automático necesita que estén identificadas con precisión.
- El Aprendizaje Automático divide el proceso en pequeños pasos que se unen en el momento de dar la salida, mientras que el Profundo resuelve el problema de un extremo a otro, todo conectado.
- El Aprendizaje Profundo tarda más tiempo en entrenarse principalmente por lo que se explica anteriormente del proceso secuencial.
- La salida que proporciona el Aprendizaje Profundo puede ser mucho más variada que la del Automático.

1.3. Ventajas que ofrece el aprendizaje profundo

[4]Como veremos más adelante, la constitución y los fundamentos de las redes neuronales (artificiales) son muy semejantes a las del cerebro, en cuanto a que son capaces de aprender de la experiencia, de extraer características especiales que a simple vista parecen irrelevantes, etc.
Estas cualidades especiales del Aprendizaje Profundo ofrecen numerosas ventajas y serán muy útiles en distintas áreas de trabajo. Las ventajas principales son:

- **Aprendizaje adaptativo**
  Esta es una de las características más atractivas del Aprendizaje Profundo. Hace referencia a la capacidad de aprender a realizar tareas basadas en el entrenamiento o experiencia inicial. Por esta misma razón, con el Deep Learning no es necesario trabajar con modelos previos ni especificar funciones de distribución de probabilidad. Decimos que son adaptables debido a la capacidad de autoajuste de los elementos (neuronas) que componen el sistema, que constantemente están modificándose a las nuevas condiciones que se implementen.

- **Auto-organización**
  Hace referencia a la capacidad de la red neuronal de crear su propia organización o representar la información adecuadamente en la etapa de aprendizaje. Esta característica de las redes es importante porque nos explica cómo al recibir datos o información diferente a la recibida anteriormente, es capaz de reorganizarse y saber dar respuesta al problema que se plantee. Cabe destacar que esto también ocurre si la información recibida está incompleta.

- **Tolerancia a fallos**
  Debemos distinguir dos tipos de fallos: Cuando se trata de un fallo en los datos, la red neuronal es capaz de reconocer patrones con ruido, distorsionados o datos incompletos y seguir trabajando. Si se trata de un fallo en la propia red neuronal, ésta puede seguir funcionando aunque se destruya una parte (no muy grande) de la estructura, cosa que con otros sistemas computacionales tradicionales no ocurre. Esto se debería a que la información de las redes neuronales está distribuida entre todas sus neuronas, a veces de forma redundante.

- **Operación en tiempo real**
  Una necesidad importante de satisfacer en las áreas de aplicación es la necesidad de realizar los procesos de forma muy rápida. Los cómputos neuronales pueden ser realizados en paralelo, ya que se utilizan máquinas con hardware especial sólo para obtener esta cualidad.

- **Fácil insercion dentro de la tecnología actual**
  Existe la opción de entrenar y evaluar una red neuronal antes de trasladarlo a un hardware de bajo coste, así podríamos utilizar las redes neuronales para mejorar sistemas de forma incremental en ciertas tareas.

1.4. Aplicaciones

Las redes neuronales pueden solucionar un gran y variado número de problemas en distintos ámbitos. Se pueden entrenar y validar en un tiempo considerable, y pueden resolver problemas mejor que otras tecnologías. Además, al existir muchos tipos diferentes de Redes Neuronales, son capaces de solventar problemas muy distintos en la actualidad. [3] [4] Algunas de estas aplicaciones son:

- **Ámbito sanitario**
  En el ámbito sanitario, las Redes Neuronales Artificiales se utilizan para la detección de posibles enfermedades tanto por la clasificación de imágenes como por el estudio de una serie de datos del paciente. También se utilizan para la monitorización de cirugías o la predicción de efectos adversos de ciertos medicamentos en pacientes. Otro uso muy reciente y útil es en la detección de células cancerígenas. Cabe destacar que, el uso de las Redes de Neuronas Artificiales ha supuesto un gran avance en la medicina respecto a otras técnicas estadísticas utilizadas anteriormente, ya que las RNAs pueden trabajar con datos incompletos, imprecisos o con una gran cantidad de ruido.
- **Ámbito industrial y robótico**
  Las RNAs son muy útiles para el procesado de imágenes con el fin de clasificar piezas como viables para la construcción o reconocer qué componente provoca una avería (inspecciones de calidad). También se utiliza para predecir el próximo movimiento o paso que debe hacer una máquina automatizada.
- **Ámbito de la física**
  El CERN *(Conseil Europeen pour la Recherche Nucleaire)* trabaja cerca de Ginebra con el acelerador de partículas más grande que existe *(Large Hadron Collider)* y utiliza el Aprendizaje Profundo para clasificar e interpretar todos los resultados de las colisiones que se producen.
- **Ámbito aeroespacial y militar**
  En este ámbito, las RNAs son muy útiles para la detección de satélites en áreas de interés o de zonas seguras para tropas. Además, se utilizan para clasificar señales de radares, para la creación de armas inteligentes o el reconocimiento y seguimiento del tiro al blanco. En la detección de minas también encontramos otro uso muy útil para las RNAs.
- **Ámbito del Análisis Financiero**
  Otra aplicación muy útil se encuentra en el Análisis Financiero, utilizando el Aprendizaje Profundo para la previsión de la evolución de los precios, la concesión de créditos a particulares/empresas por parte de los bancos, la detección de transacciones fraudulentas con tarjetas de crédito o la interpretación de firmas.

Otros usos particulares son el reconocimiento y la síntesis de voz a través de la introducción de sonidos, la predicción meteorológica, el procesamiento del lenguaje natural, la predicción de series temporales, el reconocimiento de caracteres e imágenes, etc.

Capítulo 2

Redes Neuronales Artificiales

2.1. Fundamentos biológicos de las redes neuronales

Las RNAs tratan de emular el comportamiento del cerebro humano, que aprende a través de la experiencia y la extracción de conocimiento genérico a partir de un conjunto de datos. Estas imitan la estructura neuronal del cerebro.
Obviamente existen diferencias entre el cerebro biológico y el creado en computadores convencionales: Los computadores trabajan con microprocesadores que son muy rápidos y son capaces de ejecutar instrucciones complejas de forma fiable, mientras que el cerebro está formado por millones de procesadores elementales (neuronas) que forman redes al conectarse entre sí. Las neuronas biológicas además no necesitan ser programadas, sino que funcionan y aprenden mediante estímulos que reciben y trabajan siguiendo un esquema masivamente paralelo, diferente al procesamiento de los computadores convencionales.
El procesamiento de información que lleva a cabo cerebro biológico funciona de la siguiente forma:

1. Existe un canal de recepción de la información, que son las dendritas, quienes reciben señales de entrada (inputs) de otras neuronas o del exterior.

2. Una vez se ha recibido la información, el soma combina los inputs recibidos y emite señales de salida en forma de estímulos nerviosos.

3. La salida que emite el soma se envía a través del axón. Esta salida la puede llevar a otras neuronas para volver a repetir el proceso o a neuronas que son específicamente motoras y envían la información directamente al músculo. Para transmitir esta información, el axón se conecta a través de sus ramificaciones las dendritas de otras neuronas. Una neurona del córtex cerebral recibe información aproximadamente de unas 10.000 neuronas y envía impulsos a cuentos de ellas.

La conexión entre el axón y la dendrita de otra neurona recibe el nombre de sinapsis y

determina la fuerza y el tipo de relación entre ellas.

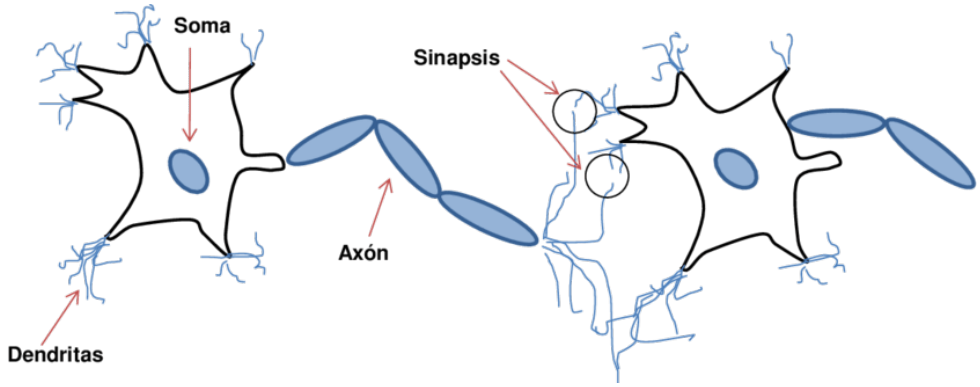


Figura 2.1: Partes de una neurona biológica

Una vez hemos visto cómo trabajan las redes de neuronas en el cerebro, podemos ver las similitudes con el procesamiento de información que realizan los computadores convencionales a través de las capas de entrada, ocultas y de salida.

2.2. Redes Neuronales Artificiales

2.2.1. Conceptos básicos

Los principales conceptos a conocer de una RNA son los siguientes:

- **Capa de entrada:** Se llama capa de entrada a la que recibe directamente la información que viene de fuentes externas a la red.
- **Capa oculta:** Estas capas no tienen ningún contacto con el exterior y pueden contar con varios niveles. La forma en la que se conectan las neuronas de esta capa oculta define las distintas topologías de las redes neuronales.
- **Capa de salida:** Esta capa devuelve la información que buscábamos predecir tras aplicar la función de activación.

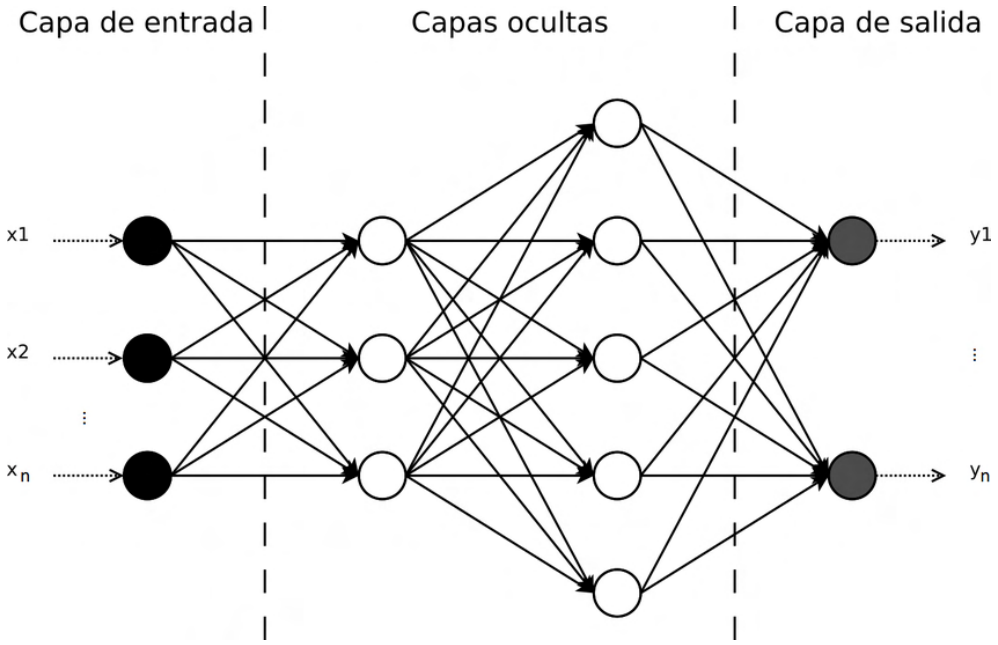


Figura 2.2: Representacion gráfica de las 3 capas de una Red Neuronal Artificial

- **Función de entrada o regla de propagación s:** [4] Una RNA trata muchos valores de entrada, y debe enfrentarse al problema de cómo combinar estas diferentes entradas y cómo asignarles la importancia que debe tener cada una.
  Para ello contaremos con el concepto del peso sináptico, que al multiplicar cada valor de entrada con el correspondiente conseguiremos cambiar la influencia de cada valor en la capa de entrada.
  Tipos de funciones de entrada conocidos:

1. *Sumatorio de las entradas ponderadas*, que suma el producto de cada valor de entrada por su peso correspondiente.

$$\sum_{j=1}^{n}(n_{ij}w_{ij}), \tag{2.2.1}$$

2. *Producto de las entradas ponderadas*, que es el producto de las multiplicaciones entre los valores de entrada y los pesos.

$$\prod_{j=1}^{n}(n_{ij}w_{ij}) \tag{2.2.2}$$

3. *Máximo de las entradas ponderadas*, que sólo elige el máximo del producto de los valores de entrada por el peso correspondiente.

$$Max_j(n_{ij}w_{ij}), \;\; con\; j = 1, 2, ..., n \tag{2.2.3}$$

La notación de las funciones de entrada es la siguiente: n_{ij} hace referencia al valor de entrada número *j* en la neurona *i*; y w_{ij} es el peso correspondiente a la neurona *i* en la entrada *j*.

- **Función de activación:** Una neurona biológica tiene dos estados: estará activa o inactiva. Las neuronas artificiales pueden tener estos dos, o varios estados a la vez. Para saber en qué estado de la actividad se encuentra una neurona necesitaremos la función de activación.
[1] Esta puede ser de los siguientes tipos:

1. *Función lineal*, que es la función de activación más simple, ya que devuelve como output el valor introducido. Esta función de activación se utiliza más en las capas de salida de las regresiones.

$$f(x) = x \tag{2.2.4}$$

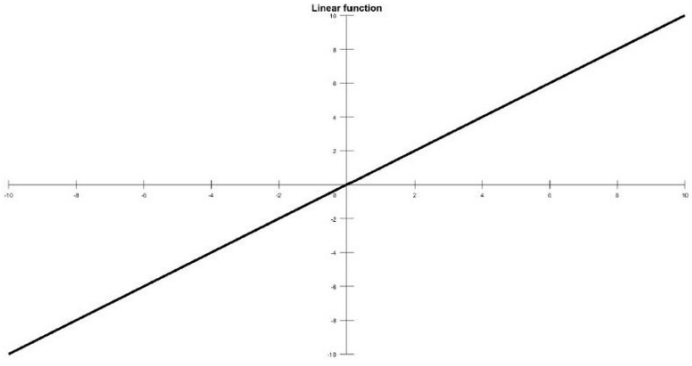


Figura 2.3: Representación gráfica de la función de activación lineal

2. *Función paso*, que es otra función de activación simple, donde se devuelve el valor 1 (o verdadero) cuando los datos recibidos superan un cierto umbral (normalmente 0 ó 0.5).

$$f(x) = \begin{cases} 1 & si \quad x \geq 0{,}5 \\ 0 & si \quad x < 0{,}5 \end{cases} \tag{2.2.5}$$

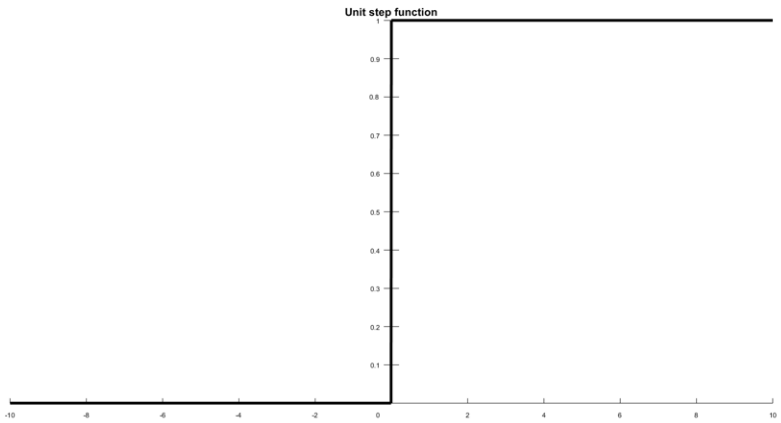


Figura 2.4: Representación gráfica de la función de activación paso

3. *Función sigmoidea*, que la utilizamos para asegurarnos de que los valores se encontrarán en un rango relativamente pequeño.

$$f(x) = \frac{1}{1 + e^{-x}} \tag{2.2.6}$$

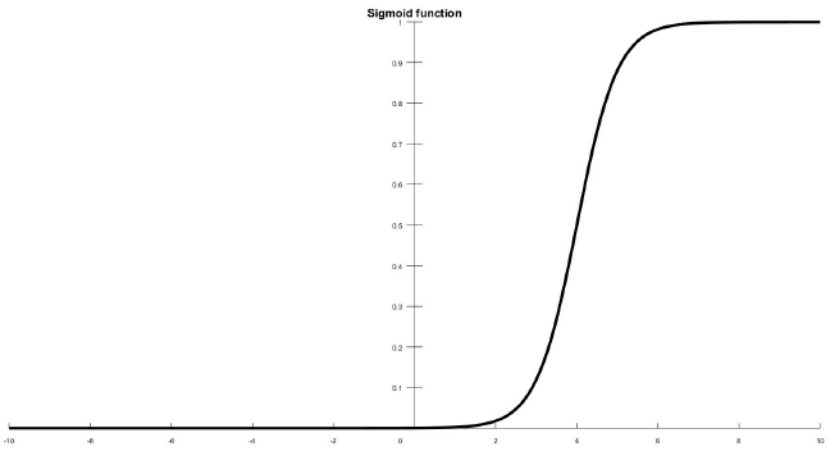


Figura 2.5: Representación gráfica de la función de activación sigmoidea

4. *Función tangente hiperbólica*, muy útil para las RNAs que necesitan como output valores entre -1 y 1.

$$f(x) = tanh(x) \tag{2.2.7}$$

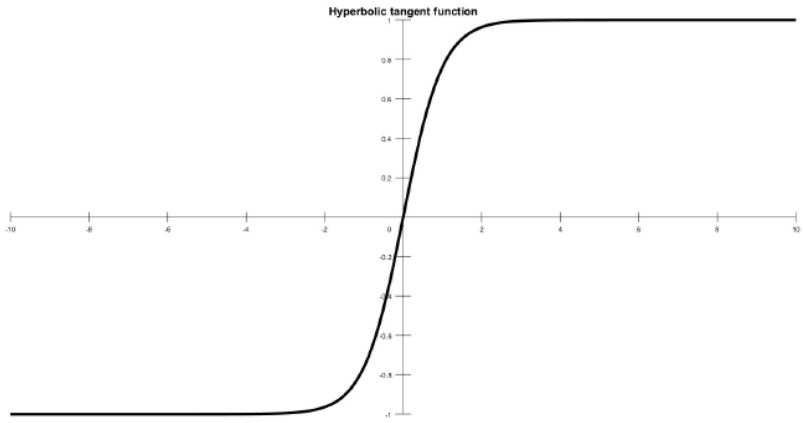


Figura 2.6: Representación gráfica de la función de activación tangente hiperbólica

5. *Función ReLU (Rectified Linear Unit)*, introducida en el año 2000 por Teh y Hinton, sustituyó en muchos lugares a la función tangente hiperbólica debido a que tenía mejores resultados en el conjunto de entrenamiento, sobre todo para trabajar en las capas ocultas.

$$f(x) = max(0, x) = \begin{cases} 0 & si \quad x < 0 \\ x & si \quad x \geq 0 \end{cases} \tag{2.2.8}$$

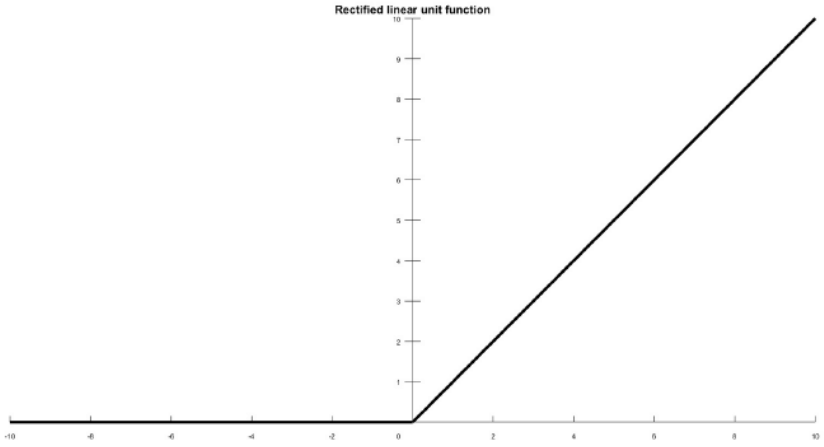


Figura 2.7: Representación gráfica de la función de activación ReLU

6. *Función Softmax*, que la encontraremos principalmente como función de activación en la capa de salida para RNAs de clasificación. Con esta función de activación la salida es un conjunto de probabilidades asociadas a cada tipo de clasificación, eligiendo la mayor probabilidad como el grupo en que lo clasificamos. Si no utilizáramos el método Softmax, la salida serían un grupo de valores numéricos, y tendríamos que elegir el mayor.

$$f(i) = \frac{e^{x_i}}{\sum_{j=0}^{n} e^{x_j}}, \tag{2.2.9}$$

donde x representa los valores de las neuronas (j) de la capa de salida.

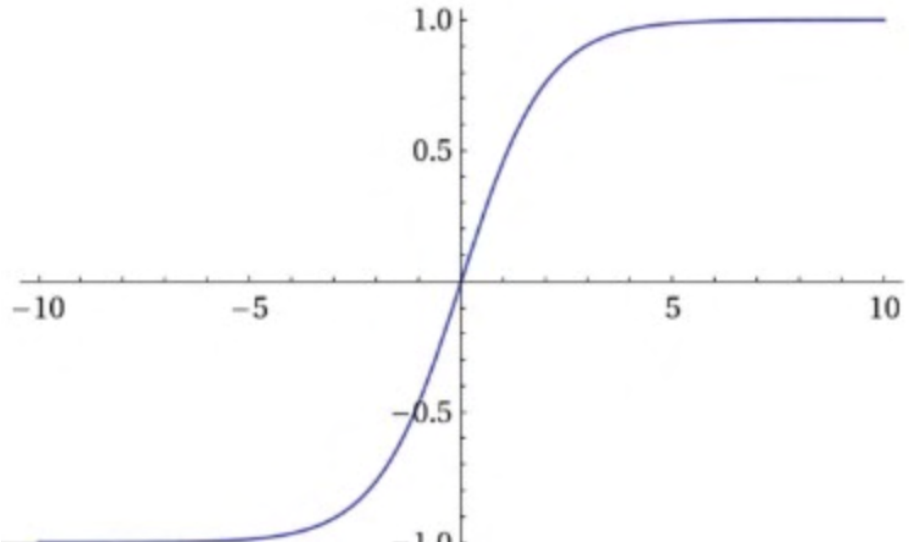


Figura 2.8: Representación gráfica de la función de activación Softmax

2.2.2. Mecanismos de aprendizaje

[4]Una red neuronal debe conseguir sacar una salida lo más correcta posible para cada conjunto de datos recibido. Para ello se debe llevar a cabo un proceso de entrenamiento o acondicionamiento a través de lo que denominaremos *conjunto de datos de entrenamiento*.
Este aprendizaje consiste en ir modificando los pesos sinápticos según la información recibida. Diremos que este proceso de aprendizaje habrá terminado cuando los valores de los pesos se mantengan estables.
Para determinar cuándo se debe acabar el proceso de aprendizaje, se necesita establecer una condición de detención, que por lo general será cuando la diferencia entre lo que la red ha proporcionado como salida y el resultado verdadero este por debajo de un umbral establecido o se haya alcanzado un número concreto de ciclos/pasos de entrenamiento.
Para trabajar con la modificación de estos pesos podemos utilizar dos métodos:

- **Aprendizaje supervisado**
  Con el aprendizaje supervisado el proceso es vigilado por un agente externo, que determina la respuesta que debería generar la red al proporcionarle una entrada concreta. Si la red falla y no da la salida que proporciona el agente externo deberán modificarse los pesos hasta conseguir la salida correcta.
  Esta corrección de pesos se puede hacer ajustando los pesos en función de la diferencia entre los valores deseados y los obtenidos *(aprendizaje por corrección de error)*, modificando los pesos si la red no acierta el resultado –que no habrá recibido previamente- *(aprendizaje por refuerzo)* o cambiando aleatoriamente los valores de los pesos hasta encontrar la combinación que nos dé el resultado más parecido al correcto *(aprendizaje estocástico)*.

- **Aprendizaje no supervisado**
  Las redes con este aprendizaje no requieren de un agente externo que ayude a regular los pesos de las conexiones entre neuronas. Aquí las redes deben relacionar los datos que se reciben de entrada según características, regularidades o categorías. Cuando consiga generar la salida, no sabrá si es la correcta o no.
  Además, el aprendizaje no supervisado se puede dividir en *aprendizaje hebbiano* (permite medir la familiaridad o extraer las características de los datos recibidos), *aprendizaje competitivo y comparativo* (que permite realizar clasificaciones de los datos de entrada) y *aprendizaje por refuerzo* (similar al aprendizaje por corrección de errores, pero sin tener un conjunto completo de los datos exactos de salida).

2.2.3. Tipos de redes

Según la topología de la red, podemos clasificar las Redes de Neuronas Artificiales en los siguientes grupos:

- **Red Neuronal Monocapa o Perceptrón Simple**, compuestas por un solo nivel de neuronas que se unen mediante conexiones laterales.

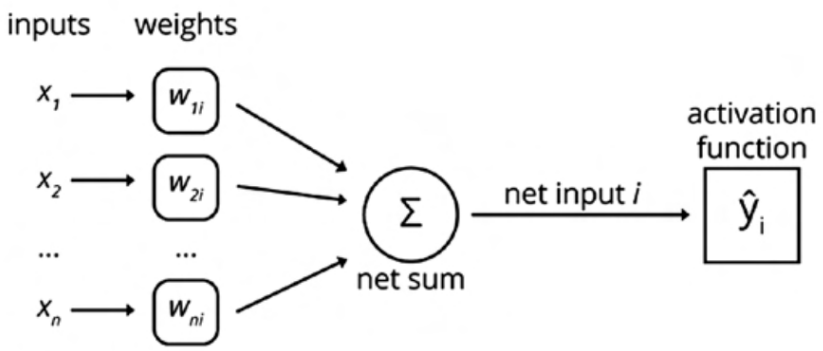


Figura 2.9: Estructura de un perceptrón simple

- **Red Neuronal Multicapa o Perceptrón Multicapa**, donde las neuronas se disponen en dos o más capas.

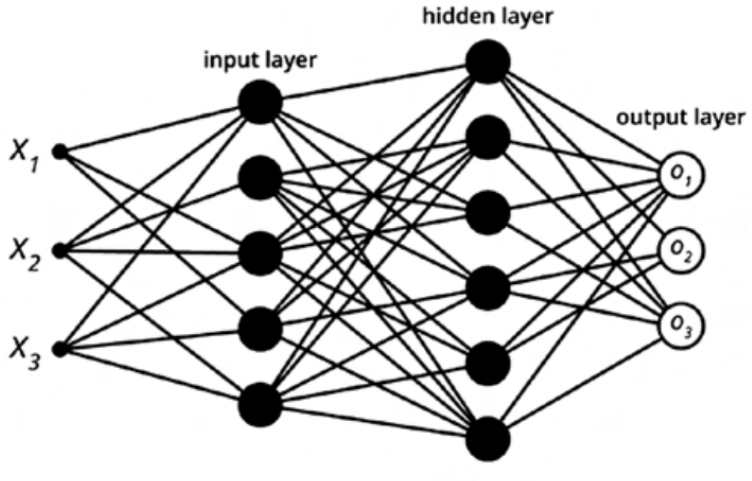


Figura 2.10: Estructura de un perceptrón multicapa

- **Red Neuronal Convolucional (CNN)**, similar al Perceptrón Multicapa con la excepción de que cada neurona no se une con todas las capas que debería, sino con un subgrupo, con el fin de reducir la complejidad computacional.
  Este tipo de modelos son muy buenos para el procesado de imágenes, ya que están

diseñados para imitar la estructura de la corteza visual, con sus neuronas organizadas para tratar con las tres dimensiones: altura, anchura y profundidad.

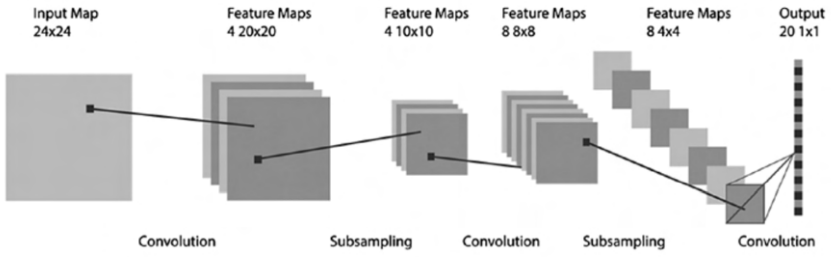


Figura 2.11: Estructura de una Red Neuronal Convolucional

- **Red Neuronal Recurrente (RNN)**, que no funciona por capas, sino que permiten conexiones arbitrarias con otras neuronas que a veces crean ciclos (así la red puede conseguir tener memoria). Estos modelos se utilizan principalmente en reconocimiento de voz y escritura.

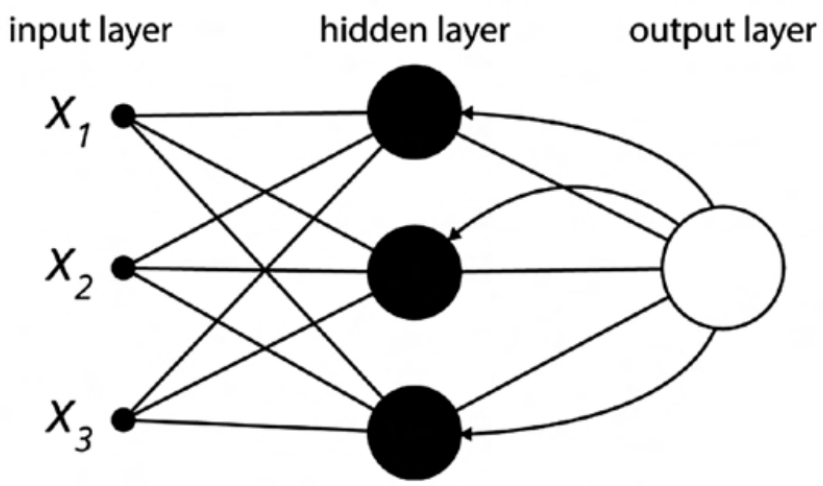


Figura 2.12: Estructura de una Red Neuronal Recurrente

- **Redes de Base Radial (RBF)**, que calculan la salida en función de la distancia a un punto denominado centro. Esta salida es combinación lineal de las funciones de activación radiales utilizadas por cada neurona.

2.2.4. El perceptrón

[5]El Perceptrón, que podría ser históricamente el primer modelo de Neurona Artificial propuesto (por el psicólogo F. Rosenblatt, en 1958), es la estructura básica de prácticamente todas las Redes de Neuronas Artificiales, por lo similar que es el modelo de neurona biológico.
Rosenblatt defendió en 1961 el teorema del aprendizaje del Perceptrón, donde explica que un Perceptrón puede resolver todo lo que pueda simular o representar. Esto es cierto si tenemos en cuenta que los perceptrones se pueden agrupar formando capas, ya que no todos los tipos de modelos los podría resolver un solo perceptrón.

2.2.4.1. El perceptrón simple

Un Perceptrón Simple o Mononivel es una Red de Neuronas Artificiales alimentada hacia delante que cuenta con dos capas que funciona de la siguiente forma:

- En la capa de entrada se reciben los valores correspondientes que se envían directamente a la capa de salida tras sumar la multiplicación de cada valor por su peso correspondiente y el bias.
- Esta suma pasará por la función de activación (función paso o signo) que definirá la salida.

Si tomamos como vector de entrada $\vec{x} = (x_1, x_2, \ldots, x_p)'$, la función de activación signo y definiendo $x_0 = 1$ tendríamos:

$$y = \begin{cases} 1 & si \sum_{i=0}^{p}(w_i x_i) \geq 0 \\ -1 & si \sum_{i=0}^{p}(w_i x_i) \leq 0 \end{cases} \tag{2.2.10}$$

El objetivo del Perceptrón simple es la clasificación en dos conjuntos correctamente. Definimos además el error total de clasificación como el total de patrones de entrenamiento que la red ha clasificado incorrectamente.

2.2.4.2. El perceptrón múltiple

Definimos el Perceptrón Multinivel o Multicapas como la Red Neuronal Artificial alimentada hacia delante con 3 o más capas de neuronas.
Dependiendo del número de conexiones de la red esta puede ser total o parcialmente conectada. Para una red total cada neurona de una misma capa estará conectada con todas las neuronas de la siguiente capa.
Además, dependiendo de en qué sentido se envían y reciben las señales de cada neurona tendríamos dos tipos de Perceptrones Multicapa:

- Con conexiones hacia delante o *feedforward*, donde todas las neuronas reciben las señales de la capa anterior, y las envían a la capa que está inmediatamente la siguiente.
- Con conexiones hacia atrás o *backforward*, donde la salida de una neurona no tiene por qué conectar inmediatamente con la siguiente, sino que puede ir a capas anteriores o posteriores.

La gran diferencia de los perceptrones multicapa respecto al perceptrón simple es la existencia del algoritmo de retropropagación, método que nos ayuda a entrenar la red neuronal. Este algoritmo calcula el error cometido en cada capa hasta la capa de salida, de forma que se puede ver dónde es menos preciso y así la red se puede modificar en ese punto.

2.3. Redes de aprendizaje profundo en RStudio y Python

Las Redes Neuronales Profundas se identifican porque cuenta con varias capas ocultas, de las que no tenemos por qué saber sus inputs y outputs. La cantidad de capas y las funciones que se utilizan en cada una de ellas crean una gran distinción entre las redes, unas más útiles que otras para ciertos problemas que haya que resolver.
El tener varias capas ayuda a que el modelo entienda desde los aspectos más simples a los más completos que los datos a estudiar puedan proporcionar.

El funcionamiento de la Red Neuronal Profunda es el siguiente:
La capa de entrada de la red depende de las características de los datos que se van a introducir, definiendo así el número de nodos que tendrá. Por otro lado, la capa de salida tendrá tantos nodos como valores a predecir. Las capas que quedan entre estas dos, las capas ocultas, también cuentan con nodos y las conexiones entre nodos de distintas capas tienen un peso asociado.
Para conseguir los pesos asociados más adecuados, la red realizará iteraciones ajustando los pesos de forma que el error sea el mínimo posible.
Para la elección de nodos en cada capa se debe tener en cuenta lo siguiente: si elegimos poca cantidad de nodos, puede resultar en un modelo simple pero con un error muy alto; pero si elegimos una gran cantidad de nodos (que podemos pensar que tratará los datos más a fondo), estaremos sobre ajustando el modelo y puede que no funcione bien para datos que la red no haya trabajado antes.

2.3.1. Análisis de RNAs en RStudio: Trabajando con la librería h2o

La librería *h2o* es un producto creado por la compañía *H2O.ai* que combina los algoritmos del machine learning y el aprendizaje estadístico. Por su forma de trabajar con los datos, es capaz de trabajar con millones de registros para un conjunto de datos.
Se puede trabajar con esta librería integramente desde R: podemos iniciar el cluster, cargar los datos, entrenar los modelos, predecir nuevas observaciones, etc. Pero aunque se esté ejecutando todo desde R, los datos y todo con lo que trabajamos se encuntra en el cluster de H2O, no en la memoria. Para transferir los datos de R al cluster, y viceversa, se utilizarán las funciones `as.data.frame()` y `as.h2o()`.
Cuando trabajamos con miles o millones de datos puede ocurrir que tengamos que esperar mucho tiempo para el entrenamiento de la red. La librería *h2o* tiene criterios de parada para que no se continue mejorando el modelo si ya se ha obtenido una solución aceptable.

Ahora sí, cuando hablamos concretamente de trabajar con Redes Neuronales Artificiales, *h2o* trabaja con las de tipo *Multicapa* y *feedforward* en concreto.
Los modelos basados en Redes Neuronales se caracterizan por ser potencialmente capaces de aprender cualquier función. La velocidad con la que hacen esto depende principalmente del número de observaciones, el número de neuronas y el de iteraciones de aprendizaje

(***epochs***). Veamos ahora cuáles son los parámetros propios de la función que crea estas redes en la librería:

- ***Para la estructura de la red***
  - `hidden`: Indica el número de capas ocultas y el número de neuronas por capa.
  - `training_frame`: conjunto de datos de entrenamiento en formato h2o.
  - `validation_frame`: conjunto de datos test en formato h2o.
- ***Preprocesado***
  - `standarize`: Para ver si se estandarizan los valores para que tengan media cero y varianza uno. Por defecto lo hace.
  - `missing_values_handling`: Como las RNAs no aceptan registros con valores ausentes H2O ofrece la opción de excluirlos o de imputarlos con la media.
  - `shuffle_training_data`: Mezcla aleatoriamente los registros antes del entrenamiento.
- ***Aprendizaje***
  - `activation`: Función de activación de las neuronas en las capas intermedias. La función de activación de la capa de salida se selecciona automáticamente según sea problema de clasificación o regresión.
  - `loss`: Función de coste.
  - `epochs`: Número de iteraciones de aprendizaje durante el entrenamiento de la red. Es importante identificar bien este valor ya que si se pone un valor más bajo del necesario la red puede no aprender lo suficente como para resolver el problema, y se si pone un valor demasiado alto podemos encontrarnos con un sobreajuste.
- ***Regularización***
  - `input_dropout_ratio`: Porcentaje de *dropout* en la capa de entrada. Controla que ningún predictor influya demasiado en la red.
  - `hidden_dropout_ratios`: Porcentaje de *dropout* en las capas intermedias.
  - `l1, l2`: Penalizaciones L1 (Lasso) y L2 (Ridge).

2.3.2. Análisis de RNAs en Python: Trabajando con las librerías Tensorflow, Scikit-Learn y Keras

El lenguaje de programación de Python se está convirtiendo en uno de los lenguajes más populares hoy en día. Es por eso que se utiliza no sólo en el ámbito académico, sino en el industrial también. *TensorFlow* es una bilbioteca de código abierto que se especializa en el Machine Learning. Dado que las redes neuronales (y otros algoritmos) funcionan principalmente con fórmulas complejas, es más conveniente ejecutarlas en Unidades de Procesamiento Gráfico (GPU) en lugar de Unidades de Procesamiento estándar (CPU). TensorFlow permite trabajar con ambas.
La otra biblioteca que utilizamos es *Scikit-Learn*, que sirve para el aprendizaje automático en varios algoritmos de clasificación, regresión y análisis de grupos. Es característica porque su lenguaje y su interfaz es fácil de entender, y así conseguimos que esta biblioteca sea más utilizada.
Para el tratamiento de imágenes utilizamos la biblioteca *Keras*, una API que fue creada para un rápido proceso de experimentación con redes de Aprendizaje Profundo en Python. Destaca por estar creada "para los humanos y no para las máquinas", haciendo referencia a su facil utilización por los usuarios.
Veamos, para la resolución de nuestros ejemplos, cuáles son las funciones más destacables para el Aprendizaje Profundo:

- `train_test_split`: De la librería *sklearn.model_selection*, nos sirve para hacer la partición en conjuntos de entrenamiento y test donde sólo tenemos que indicar los conjuntos de datos de las variables predictoras y a predecir, la proporción test/entrenamiento y una semilla.
- `MLPClassifier`: Función de la librería *sklearn.neural_network* que nos sirve para crear el modelo de Red Neuronal Artificial cuando el problema a solucionar es de clasificación.
- `MLPRegressor`: Función de la librería *sklearn.neural_network* que nos sirve para crear el modelo de Red Neuronal Artificial cuando el problema a solucionar es de regresión.
- `accuracy_score`, `roc_curve` y `roc_auc_score`: De la librería *sklearn.metrics*, que sirven para obtener el coeficiente de acierto en un problema de clasificación, la representacion de la curva ROC y el valor del Área Bajo la Curva (AUC).
- `r2_score`: De la misma librería que las funciones anteriores, sirve en este caso para los problemas de regresión para obtener el valor del coeficiente de determinación (R^2).
- `classification_report`: Función de la biblioteca *Keras* que nos muestra una tabla con los coeficientes de acierto y precisión de las clases a predecir, una vez se ha creado y adaptado el modelo.

Para la creación de una red neuronal en Keras, la estructura es la siguinte:

- `Sequential()`: Un modelo secuencial es apropiado cuando queremos crear una red definiendo las capas una a una. Es lo primero que debemos poner para crear la red.
- `modelo.add(Dense())`: Debemos crear tantas líneas de código de esta forma como capas (sin contar la de entrada) tenga nuestra red. En la primera capa oculta debemos indicar el tamaño de la capa de entrada, y en todas, el número de neuronas de la capa y la función de activación correspondiente.
- `modelo.compile()`: Sirve para entrenar la red, y en ella debemos incluir la función de pérdida y el solver, entre otros.
- `modelo.fit()`: Aquí debemos dar los conjuntos de datos de entrenamiento y los test para la validación. Obtendríamos ya el modelo completamente entrenado.

Capítulo 3

Ejemplo de clasificación

3.1. Planteamiento del problema

Este conjunto de datos contiene las transacciones realizadas con tarjeta de crédito en Septiembre de 2013 por personas europeas poseedoras de esta tarjeta de crédito. La base de datos con la que contamos ya ha pasado por un Análisis de Componentes Principales para así no tener todos los datos relativos a estas transacciones por problemas de confidencialidad.

3.2. Comentarios sobre la base de datos

Este conjunto de datos cuenta con 31 variables, todas ellas numéricas. Indican lo siguiente:

- *Time*: Esta variable hace referencia a los segundos que han pasado desde la primera transacción hasta la correspondiente en el registro.
- $V_1, ..., V_{28}$: Variables transformadas tras el ACP.
- *Amount*: Cantidad de dinero en la transacción.
- *Class*: Variable de tipo factor que indicará un 0 si la transacción no es fraudulenta, y un 1 en caso contrario.

3.3. Resolución en RStudio

Primero vamos a realizar un estudio de las variables en RStudio antes de empezar con las estructuras de las RNAs.
Tras una primera lectura vemos que las transacciones separadas en fraudulentas y no fraudulentas tienen las siguientes proporciones:

0 (Tr. no fraudulenta)	*1 (Tr. fraudulenta)*
284315	492

A continuación veamos las relaciones entre las variables del conjunto de datos a través de la matriz de correlaciones y una representación gráfica que nos facilite su estudio:

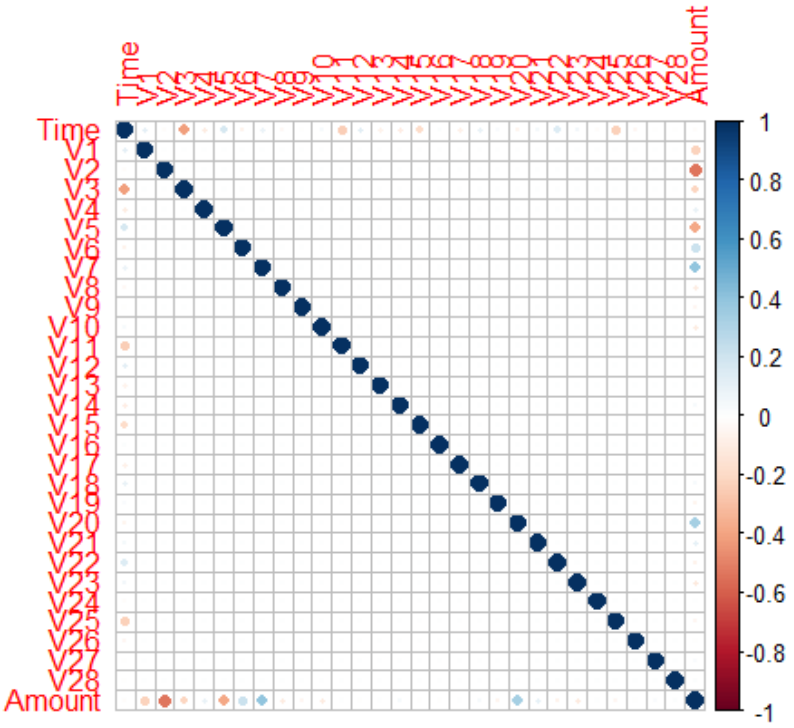


Figura 3.1: Representación gráfica de la matriz de correlaciones

Vemos cómo no habría correlación entre variables que sea significativa, pero por si acaso vamos a estudiar algunos casos donde la casilla se marca más:

- Entre las variables *Amount* y *V2*
  El valor de la correlación entre estas dos variables es de -0.5314089, es decir, que tienen una relación no muy fuerte y negativa entre ellas: cuando una variable crece, la otra decrecerá pero no de forma significativa. Si lo vemos gráficamente tendríamos:

```
plot(datos$V2, datos$Amount, xlab="Variable V2",
     ylab = "Cantidad de dinero transferido", pch=23,col="sienna2",
     main= "Representacion de las variables Amount y V2")
```

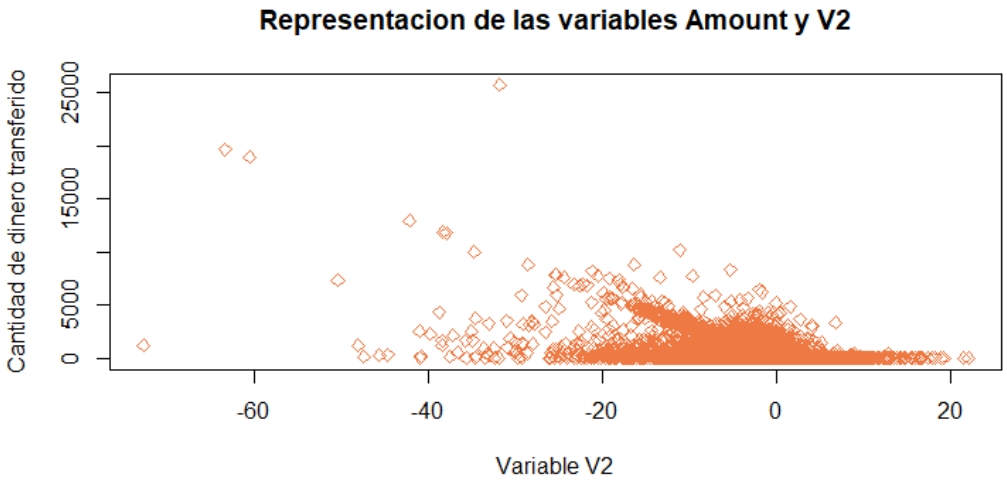


Figura 3.2: Representación gráfica de las variables Amount y V2

Podríamos intuir que cuando baja el valor de la variable V2 aumentaría el dinero que se ha transferido. No queda claramente representada esta correlación, ya que en realidad no es un valor que nos debiera preocupar.

- Entre las variables *Time* y *V3*
  El valor de la correlación entre estas dos variables es -0.4196182, también indicando una relación de mediana importancia y negativa, es decir, cuando creciera el valor de una variable, bajará el de la otra. Veámoslo gráficamente:

```
plot(datos$V3, datos$Time, xlab="Tiempo de ejecución de la transacción",
     ylab = "Variable V3", pch=20,col="sienna2",
     main= "Representación de las variables Time y V3")
```

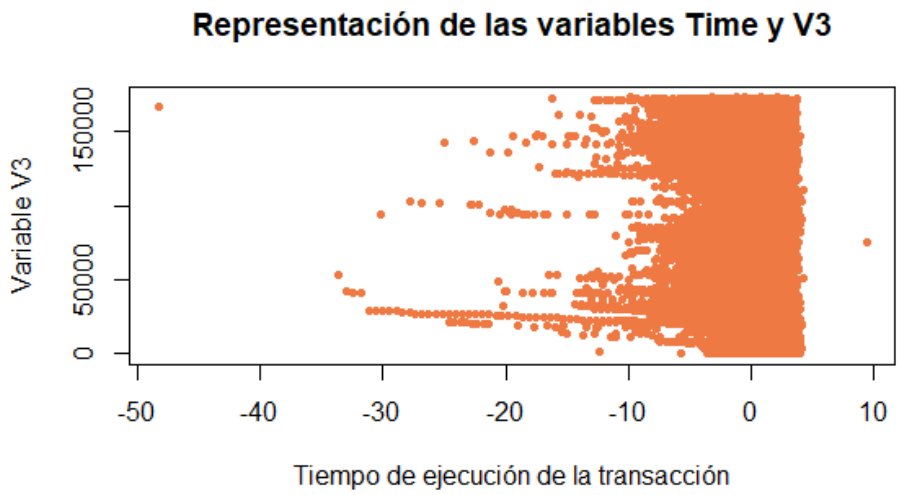


Figura 3.3: Representación gráfica de las variables Time y V3

Aquí no podemos apreciar esta influencia que una variable ejercería sobre la otra, y podemos atribuirlo a que el valor de la correlación entre estas dos variables en realidad no es tan alto como para que pudiera tener un efecto, aunque en la matriz de correlaciones pareciera que alguno podía tener.

El resto de valores de correlación están por debajo de 0.4, luego no merece la pena estudiarlos.
Una vez visto que no tenemos problemas con la interacción de las variables, procedo a trabajar con el conjunto de datos para estudiar distintas estructuras de Redes Neuronales Artificiales:
La partición en conjunto de datos test y entrenamiento la haremos con una proporción 70/30 y trabajaremos con la librería *h2o*.

```
n = nrow(datos)
nent = ceiling(0.7*n)    #n casos de entrenamiento
ntest = n-nent           #n casos de test
indin = 1:n
indient = sort(sample(indin,nent))
inditest = setdiff(indin,indient)
datos_ent = datos[indient,]
datos_test = datos[inditest,]
Y=decodeClassLabels(datos[,31])
ytest=Y[inditest,]
```

Voy a estudiar varias estructuras que modifican el número de capas ocultas y los nodos correspondientes, ya que las capas de entrada y de salida tienen la siguiente estructura fija (en este conjunto de datos):

- La capa de entrada cuenta con tantas neuronas como variables de entrada requiere, en nuestro caso una por cada una de las 28 variables especiales de la transacción, la variable Time y la variable Amount.
- La capa de salida, al tratarse de un ejemplo de clasificación en 2 tipos diferentes (Transacción Fraudulenta o No Fraudulenta), contará con 2 neuronas en la capa de salida.

Ahora, marcando unos parámetros concretos para todas las estructuras que cree a continuación, obtendremos diferentes modelos de los que estudiaremos *la especificidad*, *la sensibilidad*, *la matriz de confusión*, el área bajo la curva *(AUC)*, *la importancia de las variables*, etc.

1. **Modelo con dos capas ocultas y 25 nodos en cada una**
   Contamos con un problema de clasificación de dos clases, que sigue distribución multinomial y con estructura 30-25-25-2. La función de activación es la mencionada anteriormente *(ReLU, Rectifier Linear Unit)*, y la función *Softmax* para la capa de salida.

```
modeloc1 = h2o.deeplearning(
  x = 1:30,  #variables predictoras
  y = 31,    #variable respuesta
  training_frame = train.hex,   #datos de entrenamiento
  validation_frame = test.hex,  #datos de validacion/test
  distribution="multinomial",
  activation = 'RectifierWithDropout',
  hidden = c(25,25), #tamaño de las capas ocultas
  hidden_dropout_ratio = c(0.5, 0.5),
  input_dropout_ratio = 0.2,
  epochs = 50, #iteraciones
  l1 = 1e-5,
  l2 = 1e-5,
  rho = 0.99,
  epsilon = 1e-8)
```

   Obtenemos en este caso la siguiente información:

   - *Raíz del error cuadrático medio:* El valor del RMSE es 0.0277491, bastante apropiado ya que se acerca a 0.
   - *Área bajo la curva (AUC):* El área obtenido es 0.964825, bastante bueno también, aunque podemos estar acostumbrados a AUC más altos.
   - *Matriz de confusión:*

	0	**1**
0	85268	15
1	39	120

Cuadro 3.1: Tabla de confusión para el primer modelo propuesto en RStudio

La interpretación de esta tabla de confusión sobre los datos test es la siguiente: De las 85283 transacciones **no** fraudulentas, el modelo de RNA ha clasificado 85268 correctamente, mientras que ha clasificado de forma errónea como fraudulenta 15. Por otro lado, de las 159 transacciones que **sí** eran fraudulentas, el modelo ha clasificado correctamente 120 de ellas. Esto implicaría un porcentaje de fallo sobre las no fraudulentas de un 0.0176 %, mientras que obtendríamos un porcentaje del 24.5283 % sobre las que sí son fraudulentas.
De estos dos porcentajes obtenemos los valores de la sensibilidad y la especificidad:

- *Sensibilidad:*

$$1 - 0{,}000176 = 0{,}999824 \tag{3.3.1}$$

Esta nos está indicando la porporción de transacciones fraudulentas que el modelo ha clasificado correctamente como tales.

- *Especificidad:*

$$1 - 0{,}245283 = 0{,}754717 \tag{3.3.2}$$

Esta nos está indicando la porporción de transacciones no fraudulentas que el modelo ha clasificado correctamente como tales.

Una vez hemos obtenido y comentado los resultados más relevantes del modelo, vamos a realizar la predicción sobre el conjunto test para comprobar si este modelo es adecuado o no.
Una representación de la curva ROC y su correspondiente AUC es la siguiente:

```
  #Curva ROC y AUC
prediobj = prediction(pred[,3], ytest[,2])
plot(performance(prediobj, "tpr","fpr"),main="CURVA COR TEST",
     xlab="Tasa de falsos positivos",
     ylab="Tasa de verdaderos positivos", col="red")
abline(a=0,b=1,col="sienna",lty=2)
auc = as.numeric(performance(prediobj,"auc")@y.values)
cat("Area bajo la curva COR Test= ",auc,"\n")
```

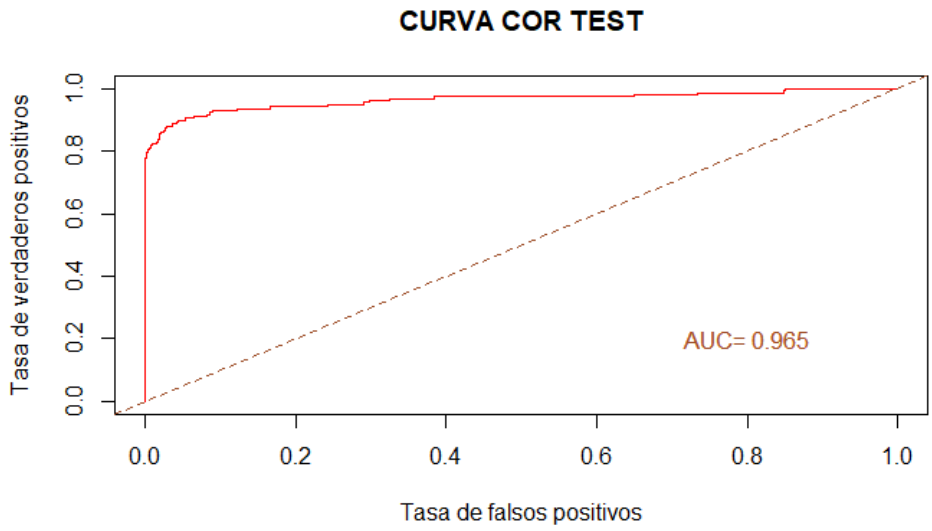


Figura 3.4: Curva ROC y AUC correspondiente al primer modelo de clasificación

Además contamos con otra representación de las curvas ROC para el conjunto de datos test y entrenamiento, que, como podemos apreciar y es lógico, es superior en el conjunto de datos de entrenamiento:

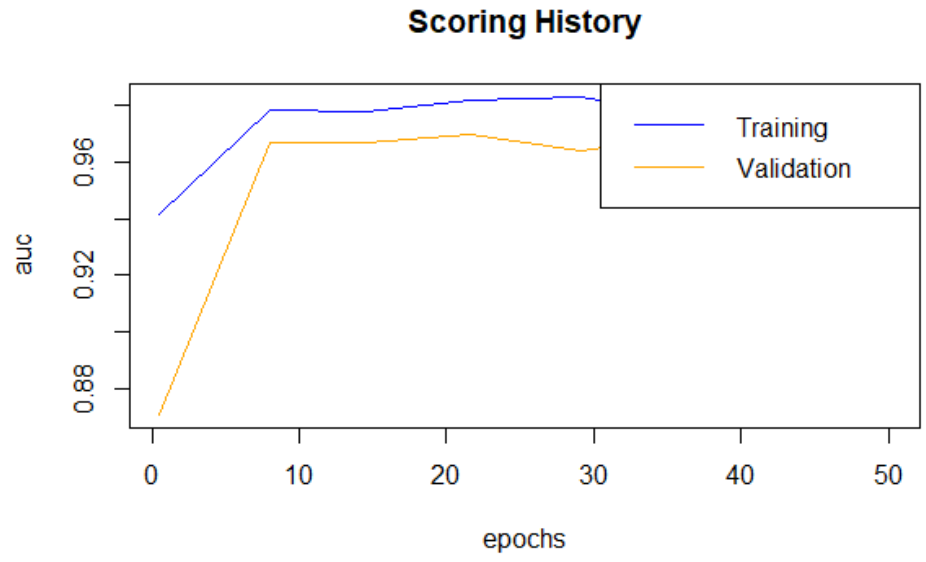


Figura 3.5: Curvas ROC para los conjuntos entrenamiento y test

Por último, hemos obtenido este gráfico que nos explica para este modelo cuales han sido las variables con mayor importancia. En el caso de este modelo, las variables más importantes han sido:

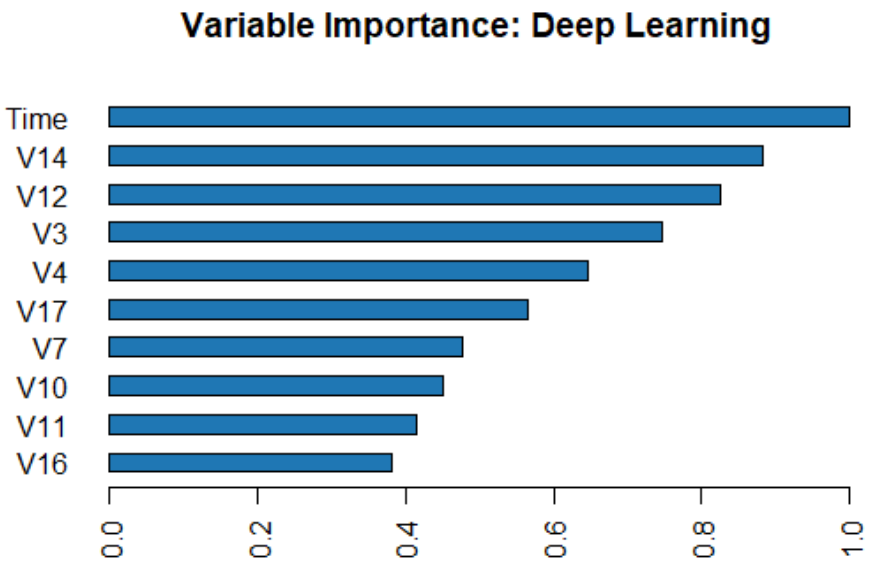


Figura 3.6: Importancia de las variables del modelo

2. **Modelo con dos capas ocultas y 10 nodos en cada una**

   La estructura de esta red es la misma que la anterior, sólo cambiando el número de nodos en las capas ocultas.

```
modeloc2 = h2o.deeplearning(
  x = 1:30,  #variables predictoras
  y = 31,    #variable respuesta
  training_frame = train.hex,   #datos de entrenamiento
  validation_frame = test.hex,  #datos de validacion/test
  distribution="multinomial",
  activation = 'RectifierWithDropout',
  hidden = c(10,10), #tamaño de las capas ocultas
  hidden_dropout_ratio = c(0.5, 0.5),
  input_dropout_ratio = 0.2,
  epochs = 50, #iteraciones
  l1 = 1e-5,
  l2 = 1e-5,
  rho = 0.99,
  epsilon = 1e-8)
```

Obtenemos en este caso la siguiente información:

- *Raíz del error cuadrático medio:* Obtenemos un valor de 0.02563881, bastante bueno para lo que buscamos
- *Área bajo la curva (AUC):* El AUC que obtenemos con este modelo es 0.9466821, también bueno, aunque se podría mejorar más.
- *Matriz de confusión:*

	0	**1**
0	85268	15
1	38	121

Cuadro 3.2: Tabla de confusión para el segundo modelo propuesto en RStudio

La interpretación de esta tabla de confusión sobre los datos test es la siguiente: De las 85283 transacciones **no** fraudulentas, el modelo de RNA ha clasificado 85268 correctamente, mientras que ha clasificado de forma errónea como fraudulenta 15. Por otro lado, de las 159 transacciones que **sí** eran fraudulentas, el modelo ha clasificado correctamente 121 de ellas. Esto implicaría un porcentaje de fallo sobre las no fraudulentas de un 0.0176 %, mientras que obtendríamos un porcentaje del 23.8994 % sobre las que sí son fraudulentas.

- *Sensibilidad:*

$$1 - 0{,}000176 = 0,999824 \tag{3.3.3}$$

La sensibilidad del modelo nos dice que de cada 100 transacciones que son fraudulentas, 99 habrán sido clasificadas correctamente como tal.

- *Especificidad:*

$$1 - 0{,}238994 = 0{,}761006 \tag{3.3.4}$$

  La especificidad del modelo nos dice que de cada 100 transacciones que son fraudulentas, 76 habrán sido clasificadas correctamente como tal.

Una vez hemos obtenido y comentado los resultados más relevantes del modelo, vamos a realizar la predicción sobre el conjunto test para comprobar si este modelo es adecuado o no.
Una representación de la curva ROC y su correspondiente AUC es la siguiente:

```
  #Curva ROC y AUC
prediobj2 = prediction(pred2[,3], ytest[,2])
plot(performance(prediobj2, "tpr","fpr"),main="CURVA COR TEST",
     xlab="Tasa de falsos positivos",
     ylab="Tasa de verdaderos positivos", col="red")
abline(a=0,b=1,col="sienna",lty=2)
auc2 = as.numeric(performance(prediobj2,"auc")@y.values)
cat("Area bajo la curva COR Test= ",auc2,"\n")
```

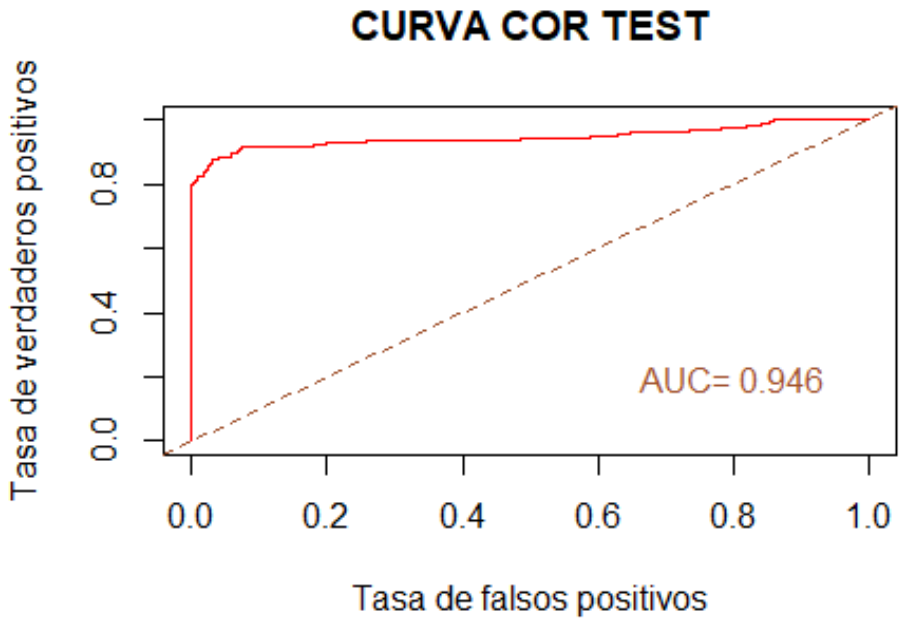


Figura 3.7: Curva ROC y AUC correspondiente al segundo modelo de clasificación

Además contamos con otra representación de las curvas ROC para el conjunto de datos test y entrenamiento, que, como podemos apreciar y es lógico, es superior en el conjunto de datos de entrenamiento:

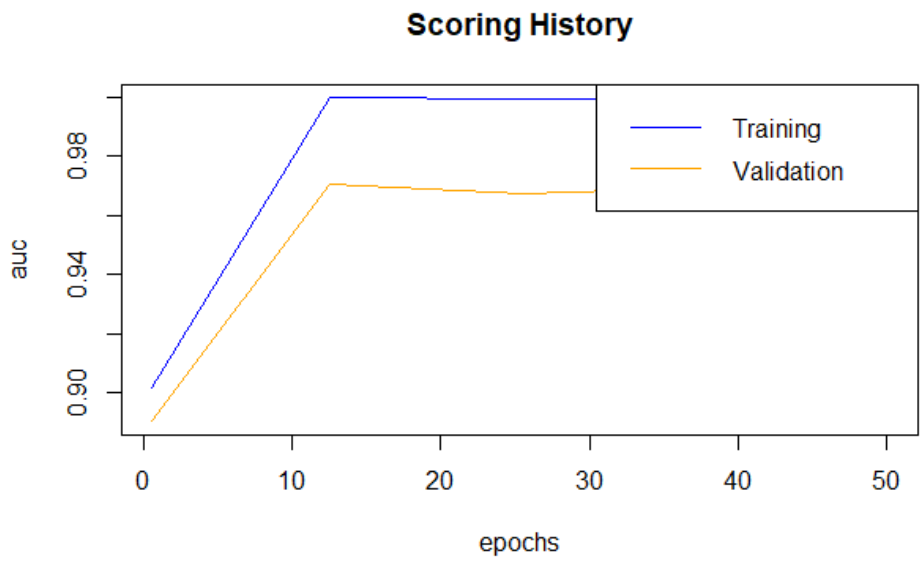


Figura 3.8: Curvas ROC para los conjuntos entrenamiento y test

Por último, hemos obtenido este gráfico que nos explica para este modelo cuales han sido las variables con mayor importancia. En el caso de este modelo, las variables más importantes han sido:

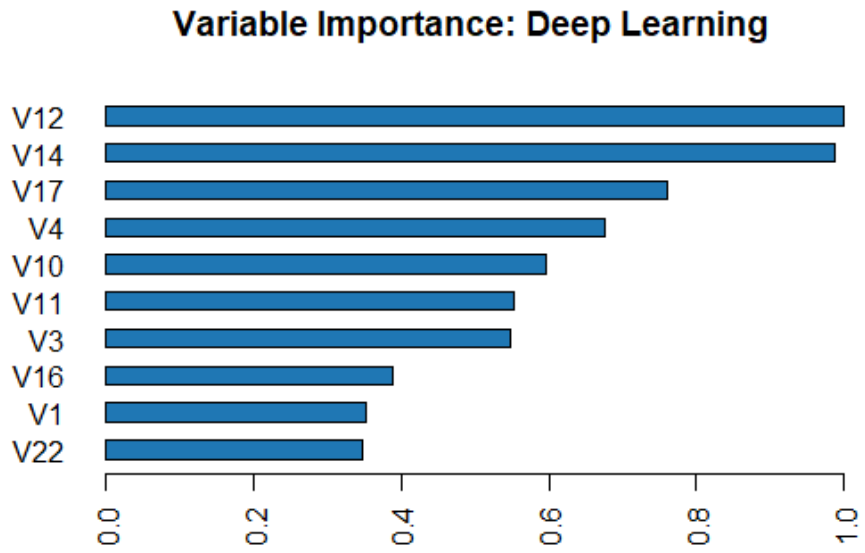


Figura 3.9: Importancia de las variables del modelo

3. **Modelo con una capa oculta y 25 nodos**

Ahora sólo contamos con una capa oculta, en este caso con 25 nodos en ella.

```
modeloc3 = h2o.deeplearning(
  x = 1:30,  #variables predictoras
  y = 31,    #variable respuesta
  training_frame = train.hex,   #datos de entrenamiento
  validation_frame = test.hex,  #datos de validacion/test
  distribution="multinomial",
  activation = 'RectifierWithDropout',
  hidden = 25, #tamaño de las capas ocultas
  hidden_dropout_ratio = 0.5,
  input_dropout_ratio = 0.2,
  epochs = 50, #iteraciones
  l1 = 1e-5,
  l2 = 1e-5,
  rho = 0.99,
  epsilon = 1e-8)
```

Veamos cuál es la información que hemos obtenido:

- *Raíz del error cuadrático medio:* El obtenido es 0.02642807, bastante acertado.
- *Área bajo la curva (AUC):* Similar a los anteriores, el AUC es 0.9695872.
- *Matriz de confusión:*

	0	**1**
0	85266	17
1	39	120

Cuadro 3.3: Tabla de confusión para el tercer modelo propuesto en RStudio

La interpretación de esta tabla de confusión es la siguiente:
De las 85283 transacciones **no** fraudulentas, el modelo de RNA ha clasificado 85266 correctamente, mientras que ha clasificado de forma errónea como fraudulenta 17. Por otro lado, de las 159 transacciones que **sí** eran fraudulentas, el modelo ha clasificado correctamente 120 de ellas. Esto implicaría un porcentaje de fallo sobre las no fraudulentas de un 0.0199 %, mientras que obtendríamos un porcentaje del 24.5283 % sobre las que sí son fraudulentas.

- *Sensibilidad:*

$$1 - 0{,}000199 = 0{,}999801 \tag{3.3.5}$$

La sensibilidad del modelo nos dice que de cada 100 transacciones que son fraudulentas, 99 habrán sido clasificadas correctamente como tal.

- *Especificidad:*

$$1 - 0{,}245283 = 0{,}754717 \tag{3.3.6}$$

La especificidad del modelo nos dice que de cada 100 transacciones que son fraudulentas, 75 habrán sido clasificadas correctamente como tal.

Una vez hemos obtenido y comentado los resultados más relevantes del modelo, vamos a realizar la predicción sobre el conjunto test para comprobar si este modelo es adecuado o no.
Una representación de la curva ROC y su correspondiente AUC es la siguiente:

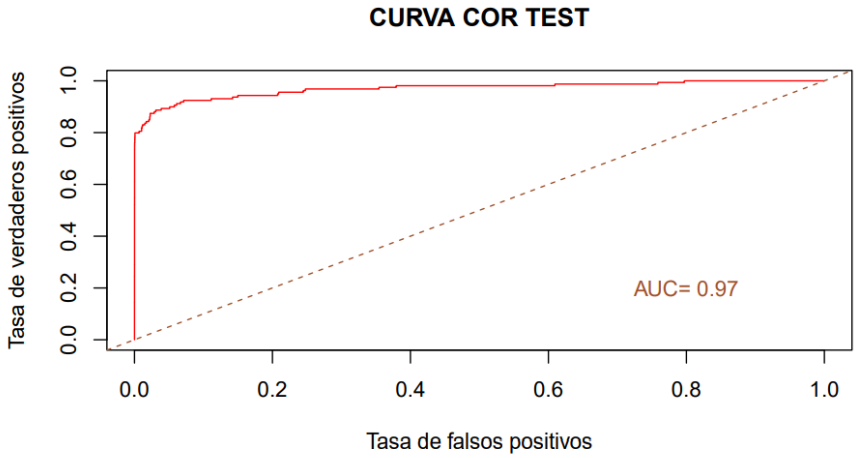


Figura 3.10: Curva ROC y AUC correspondiente al tercer modelo de clasificación

Ahora veamos la comparación de las curvas ROC para el conjunto de datos entrenamiento y test:

```
  #Curva ROC y AUC
prediobj3 = prediction(pred3[,3], ytest[,2])
plot(performance(prediobj3, "tpr","fpr"),main="CURVA COR TEST",
     xlab="Tasa de falsos positivos",
     ylab="Tasa de verdaderos positivos", col="red")
abline(a=0,b=1,col="sienna",lty=2)
auc3 = as.numeric(performance(prediobj3,"auc")@y.values)
cat("Area bajo la curva COR Test= ",auc3,"\n")
```

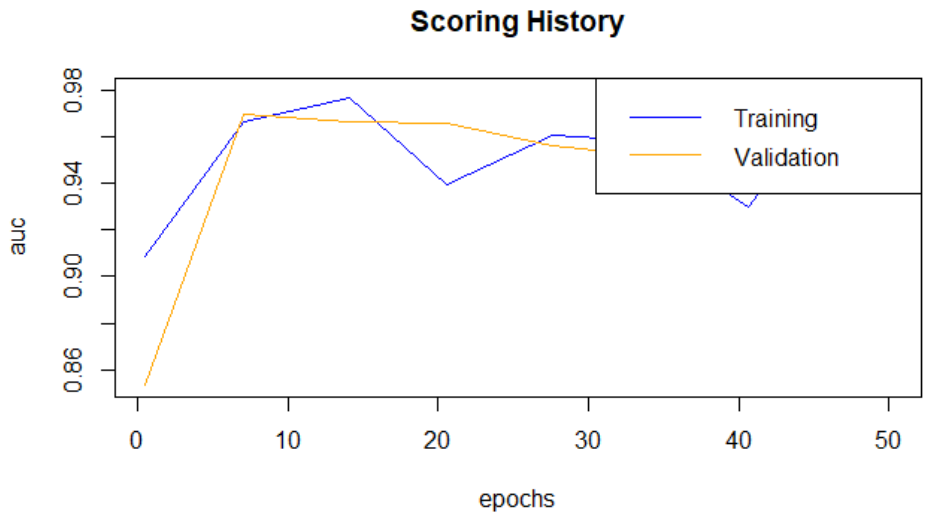


Figura 3.11: Curvas ROC para los conjuntos entrenamiento y test

Por último, hemos obtenido este gráfico que nos explica para este modelo cuales han sido las variables con mayor importancia. En el caso de este modelo, las variables más importantes han sido:

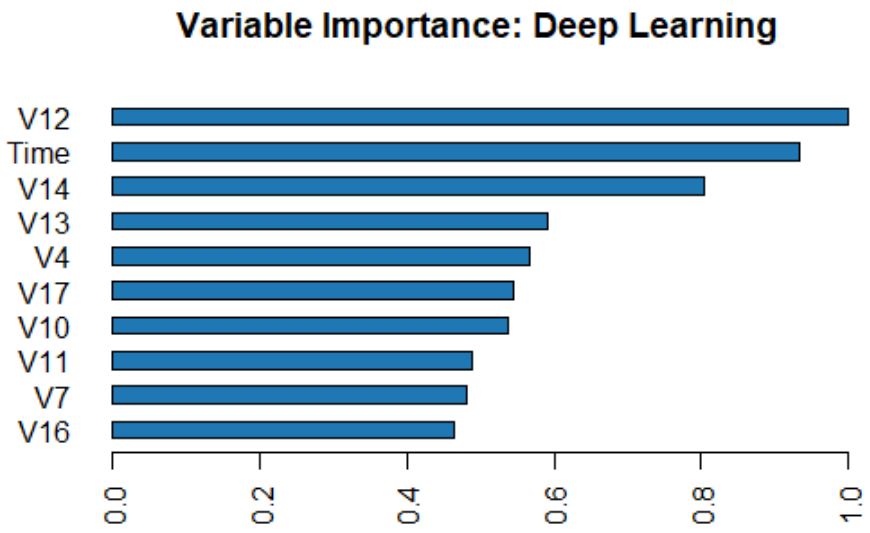


Figura 3.12: Importancia de las variables del modelo

4. **Modelo con una capa oculta y 5 nodos**

Obtenemos ahora la misma estructura de la red, pero con sólo 5 nodos en la capa oculta.

```
modeloc4 = h2o.deeplearning(
  x = 1:30,  #variables predictoras
  y = 31,    #variable respuesta
  training_frame = train.hex,   #datos de entrenamiento
  validation_frame = test.hex,  #datos de validacion/test
  distribution="multinomial",
  activation = 'RectifierWithDropout',
  hidden = 5, #tamaño de las capas ocultas
  hidden_dropout_ratio = 0.5,
  input_dropout_ratio = 0.2,
  epochs = 50, #iteraciones
  l1 = 1e-5,
  l2 = 1e-5,
  rho = 0.99,
  epsilon = 1e-8)
```

Veamos cuál es la información que hemos obtenido:

- *Raíz del error cuadrático medio:* Obtenemos prácticamente el mismo RMSE que en los otros modelos, 0.02574337.
- *Área bajo la curva (AUC):* 0.9643099, muy similar también.
- *Matriz de confusión:*

	0	**1**
0	85267	16
1	39	120

Cuadro 3.4: Tabla de confusión para el cuarto modelo propuesto en RStudio

La interpretación de esta tabla de confusión es la siguiente:
De las 85283 transacciones **no** fraudulentas, el modelo de RNA ha clasificado 85267 correctamente, mientras que ha clasificado de forma errónea como fraudulenta 16. Por otro lado, de las 159 transacciones que **sí** eran fraudulentas, el modelo ha clasificado correctamente 120 de ellas. Esto implicaría un porcentaje de fallo sobre las no fraudulentas de un 0.0188 %, mientras que obtendríamos un porcentaje del 24.5283 % sobre las que sí son fraudulentas.

- *Sensibilidad:*

$$1 - 0{,}000188 = 0{,}999812 \tag{3.3.7}$$

La sensibilidad del modelo nos dice que de cada 100 transacciones que son fraudulentas, 9 habrán sido clasificadas correctamente como tal.

- *Especificidad:*

$$1 - 0{,}245283 = 0{,}754717 \tag{3.3.8}$$

La especificidad del modelo nos dice que de cada 100 transacciones que son fraudulentas, 75 habrán sido clasificadas correctamente como tal.

Vamos a realizar la predicción sobre el conjunto test para comprobar si este modelo es adecuado o no.
Una representación de la curva ROC y su correspondiente AUC es la siguiente:

```
  #Curva ROC y AUC
prediobj4 = prediction(pred4[,3], ytest[,2])
plot(performance(prediobj4, "tpr","fpr"),main="CURVA COR TEST",
     xlab="Tasa de falsos positivos",
     ylab="Tasa de verdaderos positivos", col="red")
abline(a=0,b=1,col="sienna",lty=2)
auc4 = as.numeric(performance(prediobj4,"auc")@y.values)
cat("Area bajo la curva COR Test= ",auc4,"\n")
```

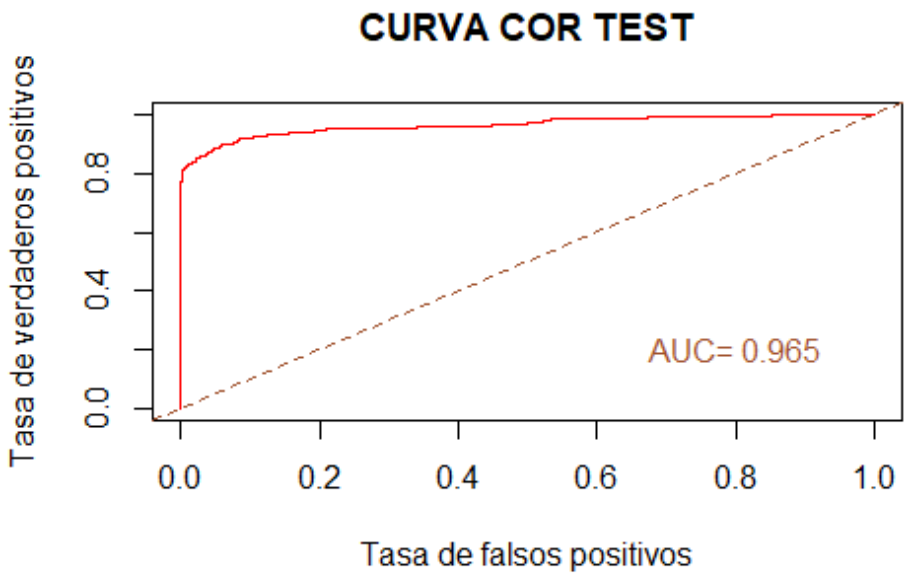


Figura 3.13: Curva ROC y AUC correspondiente al cuarto modelo de clasificación

Ahora veamos la comparación de las curvas ROC para el conjunto de datos entrenamiento y test:

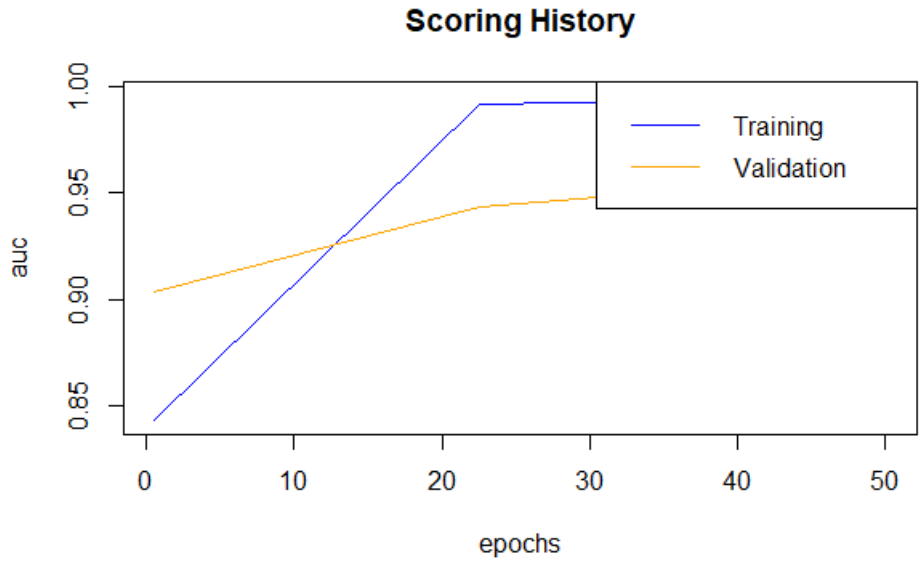


Figura 3.14: Curvas ROC para los conjuntos entrenamiento y test

Por último, hemos obtenido este gráfico que nos explica para este modelo cuales han sido las variables con mayor importancia. En el caso de este modelo, las variables más importantes han sido:

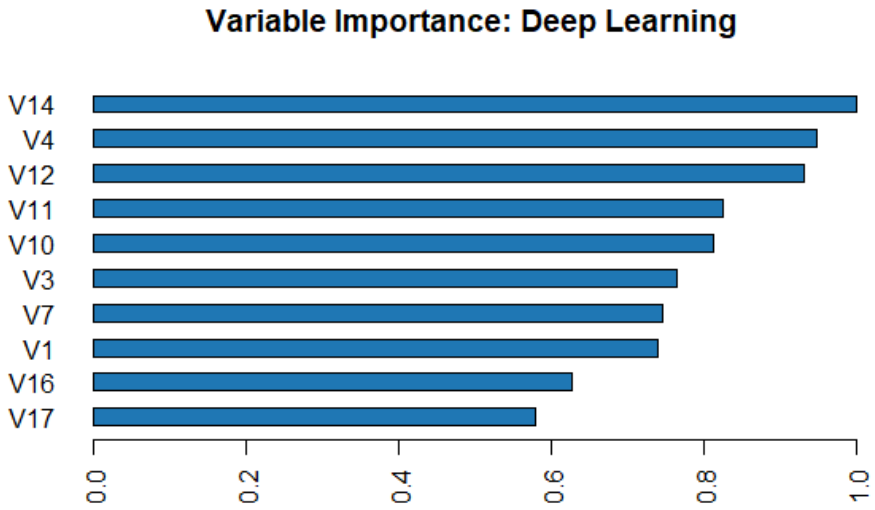


Figura 3.15: Importancia de las variables del modelo

3.3.1. Comentarios sobre los resultados obtenidos

He de decir primero que los cuatro modelos propuestos ofrecen muy buenos resultados: RMSE muy cercano a 0, AUC muy alto y matrices de confusión similares.
Es importante destacar que en este problema, el objetivo es identificar el mayor número de transacciones fraudulentas posibles. Por esto mismo, necesitamos priorizar el valor de la sensibilidad sobre el de la especificidad, ya que la sensibilidad indica el porcentaje de fraudulentas identificadas correctamente de las transacciones que realmente lo son. Entonces, para elegir el mejor modelo vamos a buscar la mejor sensibilidad:
Los cuatro modelos cuentan con una sensibilidad magnífica (más de 0.999), así que vamos a elegir el mejor entonces en función del resto de valores que he obtenido del modelo. Viendo como la especificidad y el AUC del tercer modelo **(Modelo con una capa oculta y 25 nodos)** son las más altas, elijo ese modelo como el óptimo para resolver este problema.

3.4. Resolución en Python

Vamos a seguir un proceso similar al hecho con RStudio. Primero vamos a estudiar las variables del conjunto de datos:
La proporción de transacciones fraudulentas y no fraudulentas se puede apreciar en la siguiente gráfica:

Figura 3.16: Proporcion de transacciones fraudulentas y no fraudulentas

Podemos apreciar como es mucho menor el número de transacciones fraudulentas.
Veamos ahora si existe una correlación significativa entre las variables del conjunto de datos a través de la información que nos ofrece la matriz de correlaciones. Como contamos con muchas variables para desarrollar la matriz entera, vamos a representar en una tabla las que tienen un mayor coeficiente:

```
def tidy_corr_matrix(corr_mat):
    '''Función para convertir una matriz de correlación de pandas en formato tidy.'''
    corr_mat = corr_mat.stack().reset_index()
    corr_mat.columns = ['variable_1','variable_2','r']
    corr_mat = corr_mat.loc[corr_mat['variable_1'] != corr_mat['variable_2'], :]
    corr_mat['abs_r'] = np.abs(corr_mat['r'])
    corr_mat = corr_mat.sort_values('abs_r', ascending=False)

    return(corr_mat)

tidy_corr_matrix(corr_matrix).head(10)
```

Variable 1	Variable 2	Coeficiente	V.Abs
Amount	***V2***	-0.531409	0.531409
Amount	***V7***	0.397311	0.397311
Amount	***V5***	-0.386356	0.386356
Amount	***V20***	0.339403	0.339403

No obtenemos ningún par de variables con un coeficiente de correlación extremadamente alto. Vamos a representar gráficamente los dos primeros para ver si, aun tomando

valores medianos, hay alguna relación visible entre ellas:

- *Entre las variables Amount y V2*
  El coeficiente de correlación es -0.531409, mayor a 0.5, que podría indicar un grado medio de relación inversa: cuando una variable crezca la otra decrecerá.

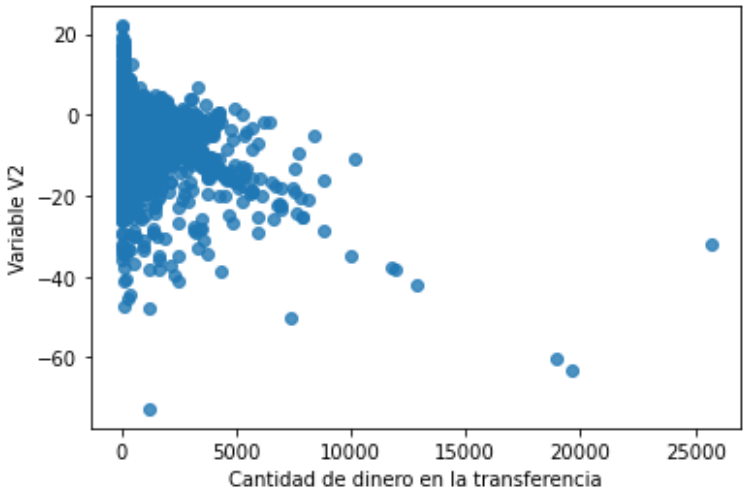


Figura 3.17: Representación gráfica de las variables Amount y V2

Para las cantidades de dinero más bajas podemos ver cómo se concentran en torno a 0 y -20,y cuando crece la cantidad de dinero transferida parece que va disminuyendo el valor de la variable V2, pero no deberíamos decir que ocurre esto claramente.

- *Entre las variables Amount y V7*
  El coeficiente de correlación en este caso es menor y positivo, 0.397311, lo que quiere decir que la relación será menor y que cuando una crezca, la otra también lo hará.

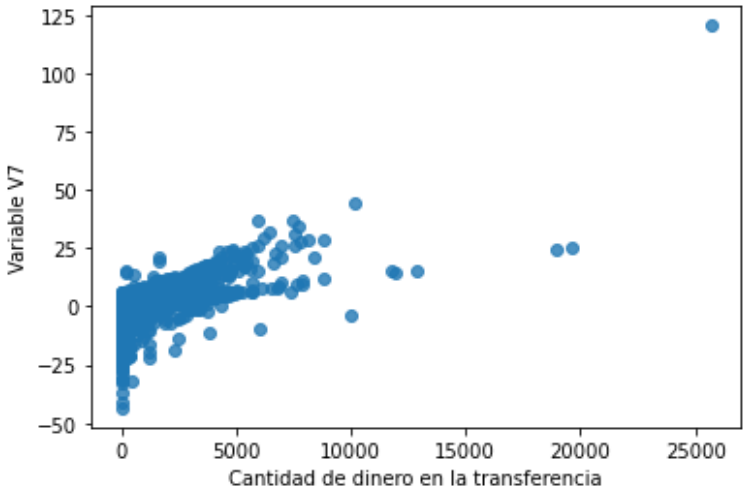


Figura 3.18: Representación gráfica de las variables Amount y V7

En este caso, podríamos decir que se asemeja a lo que debería ocurrir si hubiera correlación fuerte, que al aumnetar la cantidad de dinero en la transferencia, aumenta el valor de la variable V7, pero no ocurre así exactamente.

Una vez visto que no hay correlaciones que nos causen problemas en el estudio del conjunto de datos, vamos a realizar la partición Entrenamiento(70)/Test(30) para comenzar a crear las estructuras de Redes de Neuronas Artificiales.

```
X = datos.loc[:, datos.columns != 'Class']
y = datos["Class"]
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
```

Vamos a crear estos modelos con la misma estructura y parámetros que como hemos hecho en RStudio para hacer la comparación más acertada.

1. **Modelo con dos capas ocultas y 25 nodos en cada una**
   Contamos en este caso con una RNA que tiene dos capas ocultas y 25 nodos en cada capa.

```
modeloc1 = MLPClassifier(hidden_layer_sizes=(25,25),
                learning_rate_init=0.01,
                solver = 'adam',
                max_iter = 1000,
                random_state = 7978)

modeloc1.fit(X=X_train, y=y_train)
```

Una vez este modelo está creado y lo hemos ajustado a nuestras matrices de datos con las variables predictoras y la variable a predecir, obtenemos los siguientes resultados:

- *Matriz de confusión:*

	0	**1**
0	85293	14
1	48	88

Cuadro 3.5: Tabla de confusión para el primer modelo propuesto en Python

Esta tabla nos quiere decir lo siguiente: de 85307 registros que toman la transacción como no fraudulenta, el modelo ha sido capaz de clasificar correctamente 85293 de ellos, equivocándose en 14. De forma similar, de los 136

registros que teníamos como fraudulentos, la red ha clasificado bien 88 de ellos, equivocándose en 48.

- *Sensibilidad:* Tendremos que calcularla por definición:

$$\frac{85293}{85293+48} = 0{,}999438 \tag{3.4.1}$$

Obtenemos una sensibilidad muy alta, casi perfecta. Esto nos conviene mucho ya que lo interesante de este ejercicio es saber identificar correctamente las transacciones que son fraudulentas.

- *Especificidad:* También la calculamos por definición:

$$\frac{88}{88+14} = 0{,}862745 \tag{3.4.2}$$

En este caso la especificidad es más baja, sigue siendo un buen valor pero no tan fiable como la sensibilidad. Una explicación de lo que significa este número es la siguiente: por cada 100 pruebas que se hagan sobre una transacción que sabemos que no es fraudulenta, el modelo acertará en clasificarlas correctamente en 86 de ellas.

- *Porcentaje de acierto total (Accuracy score):* En este modelo hemos obtenido un porcentaje del 99.927437 %, un valor idóneo.
- *Curva ROC y AUC:* Para este primer modelo obtenemos un valor del Área Bajo la Curva de 0.95, valor bastante alto, nos indica que se podría tratar de un buen modelo. La representación de la curva ROC es la siguiente:

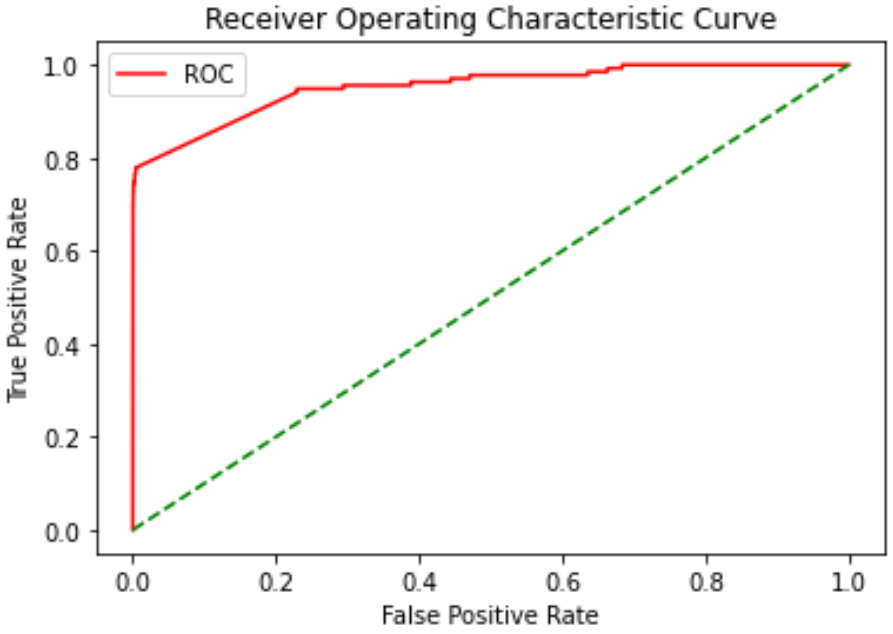


Figura 3.19: Representación gráfica de la curva ROC en el primer modelo

2. **Modelo con dos capas ocultas y 10 nodos en cada una**
Contamos en este caso con una RNA que tiene dos capas ocultas y 10 nodos en

cada capa.

```
modeloc2 = MLPClassifier(hidden_layer_sizes=(10,10),
                learning_rate_init=0.01,
                solver = 'adam',
                max_iter = 1000,
                random_state = 7978)

modeloc2.fit(X=X_train, y=y_train)
```

Una vez este modelo está creado y lo hemos ajustado a nuestras matrices de datos con las variables predictoras y la variable a predecir, obtenemos los siguientes resultados:

- *Matriz de confusión:*

	0	**1**
0	85280	27
1	24	112

Cuadro 3.6: Tabla de confusión para el segundo modelo propuesto en Python

Esta tabla nos quiere decir lo siguiente: de 85307 registros que toman la transacción como no fraudulenta, el modelo ha sido capaz de clasificar correctamente 85280 de ellos. Por otro lado, de los 136 registros que teníamos como fraudulentos, la red ha clasificado bien 112 de ellos, equivocándose en 24.

- *Sensibilidad:* Tendremos que calcularla por definición:

$$\frac{85280}{85280+27} = 0{,}999683 \tag{3.4.3}$$

Obtenemos una sensibilidad muy alta, casi perfecta. Esto nos conviene mucho ya que lo interesante de este ejercicio es saber identificar correctamente las transacciones que son fraudulentas.

- *Especificidad:* También la calculamos por definición:

$$\frac{112}{112+27} = 0{,}805755 \tag{3.4.4}$$

En este caso la especificidad también es más baja. Una explicación de lo que significa este número es la siguiente: por cada 100 pruebas que se hagan sobre una transacción que sabemos que no es fraudulenta, el modelo acertará en clasificarlas correctamente en 80 de ellas.

- *Porcentaje de acierto total (Accuracy score):* En este modelo hemos obtenido un porcentaje del 99.94031 %, un valor idóneo.

- *Curva ROC y AUC:* Obtenemos también el AUC con valor de 0.95, valor bastante alto, nos indica que se podría tratar de un buen modelo.

```
  #Curva ROC y AUC
prob2 = modeloc2.predict_proba(X_test)
probs2 = prob2[:,1]
fpr2, tpr2, thresholds2 = roc_curve(y_test, probs2)
plot_roc_curve(fpr2, tpr2)
```

La representación de la curva ROC es la siguiente:

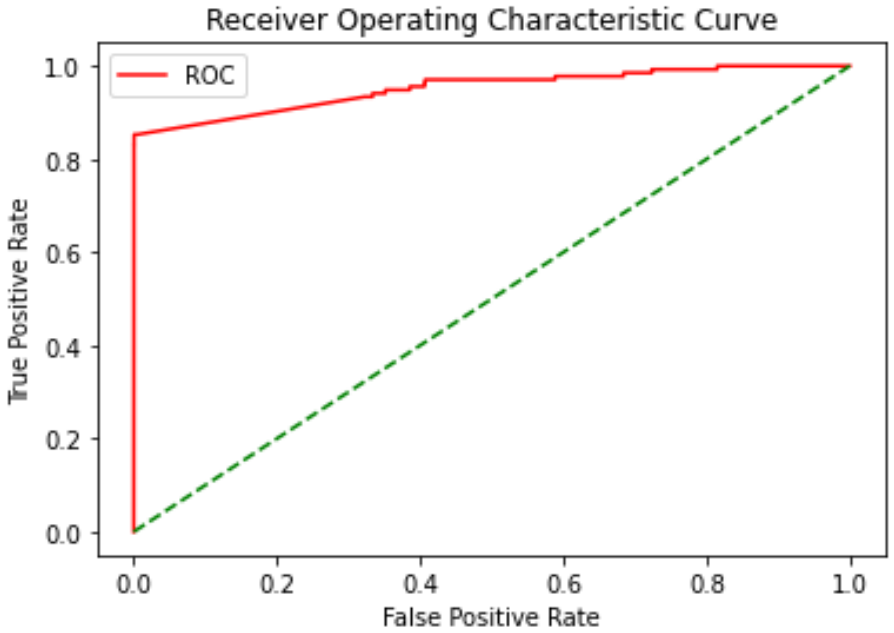


Figura 3.20: Representación gráfica de la curva ROC en el segundo modelo

3. **Modelo con una capa oculta y 25 nodos**
   Contamos en este caso con una RNA que tiene una capa oculta y 25 nodos en ella.

```
modeloc3 = MLPClassifier(hidden_layer_sizes=(25),
                learning_rate_init=0.01,
                solver = 'adam',
                max_iter = 1000,
                random_state = 7978)

modeloc3.fit(X=X_train, y=y_train)
```

Una vez este modelo está creado y lo hemos ajustado a nuestras matrices de datos con las variables predictoras y la variable a predecir, obtenemos los siguientes resultados:

- *Matriz de confusión:*

	0	**1**
0	85279	28
1	20	116

Cuadro 3.7: Tabla de confusión para el tercer modelo propuesto en Python

Esta tabla nos quiere decir lo siguiente: de 85307 registros que toman la transacción como no fraudulenta, el modelo ha sido capaz de clasificar correctamente 85279 de ellosequivocándose en 28. Por otro lado, de los 136 registros que teníamos como fraudulentos, la red ha clasificado bien 116 de ellos, equivocándose en 20.

- *Sensibilidad:* Tendremos que calcularla por definición:

$$\frac{85279}{85279 + 28} = 0{,}999672 \tag{3.4.5}$$

Obtenemos una sensibilidad muy alta, casi perfecta. Esto nos conviene mucho ya que lo interesante de este ejercicio es saber identificar correctamente las transacciones que son fraudulentas.

- *Especificidad:* También la calculamos por definición:

$$\frac{116}{116 + 20} = 0{,}852941 \tag{3.4.6}$$

La especificidad es otra vez más baja. Una explicación de lo que significa este número es la siguiente: por cada 100 pruebas que se hagan sobre una transacción que sabemos que no es fraudulenta, el modelo acertará en clasificarlas correctamente en 85 de ellas.

- *Porcentaje de acierto total (Accuracy score):* En este modelo hemos obtenido un porcentaje del 99.94382 %, un valor idóneo.

- *Curva ROC y AUC:* Para este modelo obtenemos también un valor del Área Bajo la Curva de 0.95, valor bastante alto, nos indica que se podría tratar de un buen modelo. La representación de la curva ROC es la siguiente:

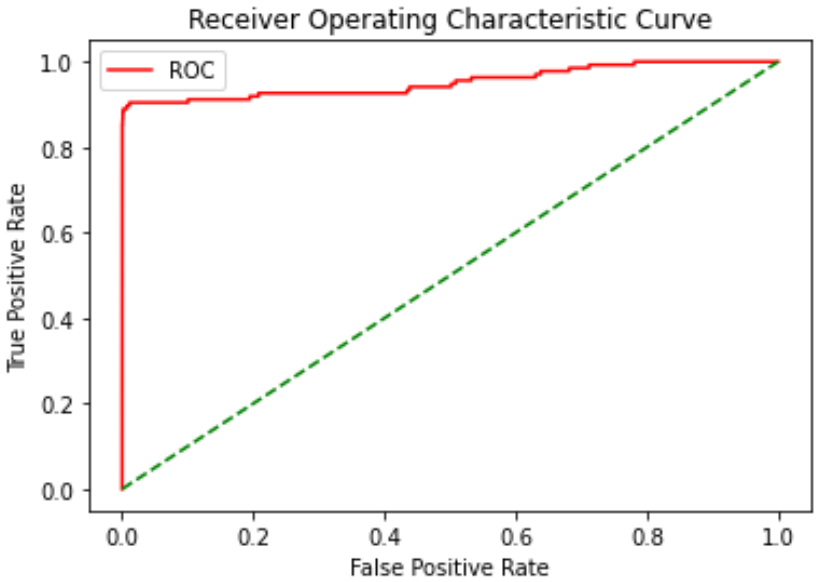


Figura 3.21: Representación gráfica de la curva ROC en el tercer modelo

4. **Modelo con una capa oculta y 5 nodos**
   Contamos en este caso con una RNA que tiene una capa oculta y 5 nodos en ella.

```
modeloc4 = MLPClassifier(hidden_layer_sizes=(5),
                         learning_rate_init=0.01,
                         solver = 'adam',
                         max_iter = 1000,
                         random_state = 7978)

modeloc4.fit(X=X_train, y=y_train)
```

Una vez este modelo está creado y lo hemos ajustado a nuestras matrices de datos con las variables predictoras y la variable a predecir, obtenemos los siguientes resultados:

- *Matriz de confusión:*

	0	**1**
0	85271	36
1	22	114

Cuadro 3.8: Tabla de confusión para el cuarto modelo propuesto en Python

Esta tabla nos quiere decir lo siguiente: de 85307 registros que toman la transacción como no fraudulenta, el modelo ha sido capaz de clasificar correctamente 85271 de ellosequivocándose en 36. Por otro lado, de los 136 registros que teníamos como fraudulentos, la red ha clasificado bien 114 de ellos, equivocándose en 22.

- *Sensibilidad:* Tendremos que calcularla por definición:

$$\frac{85271}{85271+36} = 0{,}999578 \tag{3.4.7}$$

  Obtenemos una sensibilidad muy alta, casi perfecta. Esto nos conviene mucho ya que lo interesante de este ejercicio es saber identificar correctamente las transacciones que son fraudulentas.

- *Especificidad:* También la calculamos por definición:

$$\frac{114}{114+22} = 0{,}838235 \tag{3.4.8}$$

  La especificidad es otra vez más baja. Una explicación de lo que significa este número es la siguiente: por cada 100 pruebas que se hagan sobre una transacción que sabemos que no es fraudulenta, el modelo acertará en clasificarlas correctamente en 83 de ellas.

- *Porcentaje de acierto total (Accuracy score):* En este modelo hemos obtenido un porcentaje del 99.9321 %, un valor idóneo.

- *Curva ROC y AUC:* En este caso conseguimos un Área Bajo la Curva de 0.96, valor levemente mayor a los anteriores y obviamente nos indica que se podría tratar de un buen modelo.

```
  #Curva ROC y AUC
prob4 = modeloc4.predict_proba(X_test)
probs4 = prob4[:,1]
fpr4, tpr4, thresholds4 = roc_curve(y_test, probs4)
plot_roc_curve(fpr4, tpr4)
```

La representación de la curva ROC es la siguiente:

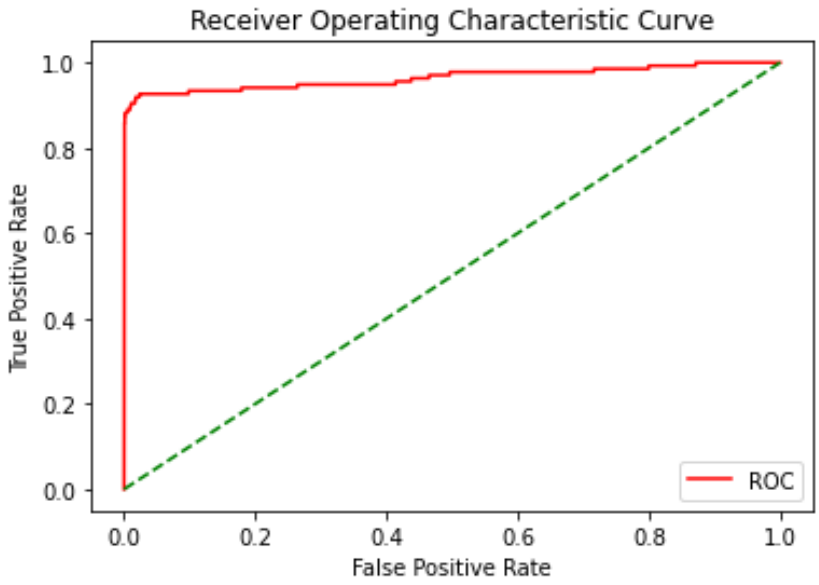


Figura 3.22: Representación gráfica de la curva ROC en el cuarto modelo

3.4.1. Comentarios sobre los resultados obtenidos

Los cuatro modelos presentan un AUC similar, bastante bueno en todos, luego este parámetro no debería servirnos para elegir el mejor. Como nos interesa clasificar correctamente sobre todo las transacciones fraudulentas, debemos buscar la sensibilidad más alta. Como en este aspecto todos los modelos prácticamente coinciden también, veamos cuál tiene la especificidad más alta: el primer modelo **(Modelo con dos capas ocultas y 25 nodos en cada una)** cuenta con la especificidad más alta, un 0.8627.

3.5. Comparación de eficiencia de ambos programas

En resumen, el mejor modelo según los cálculos de RStudio es el tercer modelo: **Modelo con dos capas ocultas y 25 nodos en cada una**, mientras que para Python el mejor modelo es el primero: **Modelo con dos capas ocultas y 25 nodos en cada una**. El tercer modelo (en RStudio) cuenta con un AUC de 0.9695872, una sensibilidad de 0,999801 y una especificidad de 0,754717. El primer modelo (en Python) tiene un AUC de 0.95, una sensibilidad de 0,999438 y una especificidad de 0,862745. Son datos bastante similares, pero por haber mayor diferencia en la especificidad que en el AUC, y porque aunque son ambos bajos los tiempos de ejecución de Python son más bajos, me decantaría por el modelo obtenido en Python.

Capítulo 4

Ejemplo de regresión

4.1. Planteamiento del problema

Contamos ahora con un conjunto de datos que trata variantes de tipos de vino español. Cualidades como la popularidad de la bodega, año o calidad pueden definir el precio de venta de estos vinos. Lo que queremos conseguir es crear una Red Neuronal Artificial que nos ayude a predecir correctamente el precio justo para cada tipo de vino.

4.2. Comentarios sobre la base de datos

Contamos con 11 variables, que son las siguientes:

- *winery*: Nombre de la bodega.
- *wine*: Nombre del vino.
- *year*: Año en el que las uvas fueron recogidas.
- *rating*: En una escala del 1 al 5, puntuación media dada al vino por varias personas.
- *num_reviews*: Número de personas que evaluaron este vino.
- *country*: País de origen, en este caso sólo España.
- *region*: Denominación de origen del vino.
- *price*: Precio (en euros).
- *type*: Tipo de vino.
- *body*: Puntuación, del 1 al 5, en función de la calidad y peso del vino al testarlo.
- *acidity*: Puntuación, del 1 al 5, de la acidez del vino.

4.3. Resolución en RStudio

Vamos a comenzar primero con un estudio general del conjunto de datos. Aquí es importante destacar que muchas de las variables tendremos que codificarlas con factor, ya que hablan de bodegas, regiones o tipos de vino, etc.
Para ver si existen relación entre las variables vamos a observar qué nos ofrece la matriz de correlaciones:

	year	**rating**	**num_reviews**	**price**	**body**	**acidity**
year	1	-0.2995	0.0421	-0.3844	-0.1007	0.1554
rating	-0.2995	1	-0.0031	0.5519	0.1611	-0.0851
num_reviews	0.0421	-0.0031	1	-0.0459	0.0796	0.0507
price	-0.3844	0.5519	-0.0459	1	0.1509	-0.0306
body	-0.1007	0.1611	0.0796	0.1509	1	-0.0016
acidity	0.1554	-0.0851	0.0507	-0.0306	-0.0016	1

Cuadro 4.1: Matriz de correlaciones del ejemplo de regresión en RStudio

No se aprecia ningún valor alto de correlación entre dos variables, pero sería interesante estudiar si hay gráficamente relación entre las variables *price* y *rating* (con un coeficiente de correlación de un 0.5519) y el par de variables *price* y *year*, con un coeficiente de correlación de -0.3844. Vamos a comprobar con la gráfica de correlaciones primero que estos dos valores son los más altos:

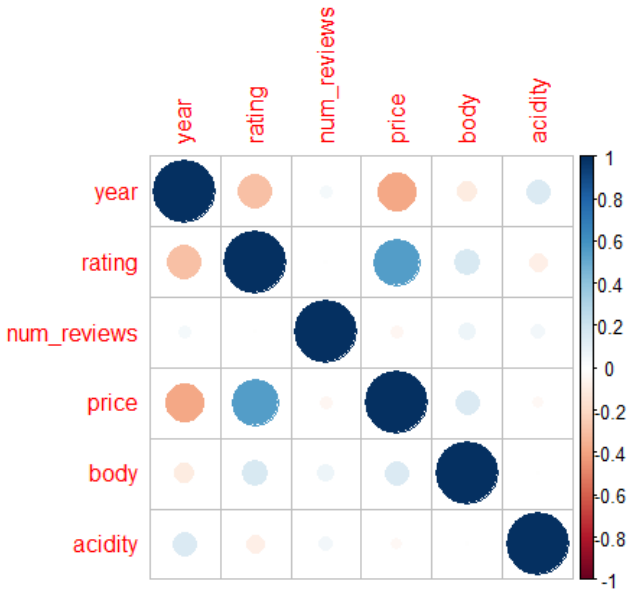


Figura 4.1: Representación gráfica de la matriz de correlaciones

Como habíamos visto por los valores de los coeficientes, esos dos pares de variables merecen la pena de estudiar por si existe algún tipo de relación que nos afecte al estudio general.

- *Entre las variables price y rating*
  El coeficiente de correlación es 0.5519, y esto quiere decir que entre ambas variables habría una relación positiva y no de mucha intensidad: cuando una variable crezca la otra también lo hará.
  Veamos una representación gráfica de ambas variables enfrentadas:

```
plot(datos$price, datos$rating,  xlab="Precio de la botella de vino",
     ylab = "Calificación del vino", col="sienna2",
     main="Representación de las variables price y rating")
```

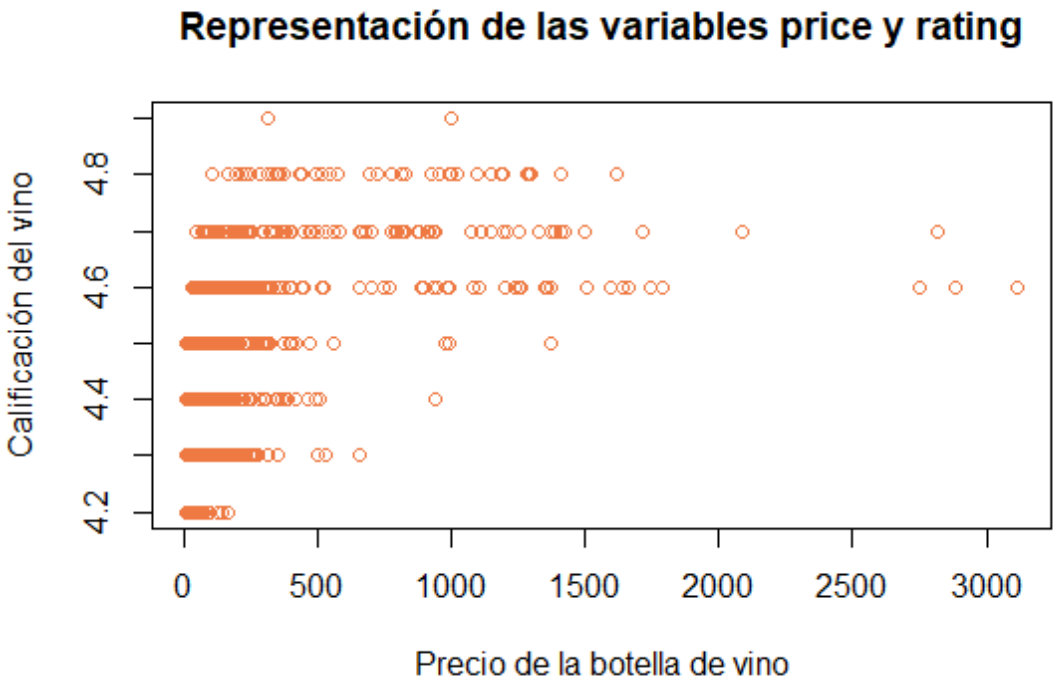


Figura 4.2: Representación gráfica de las variables price y rating

Teniendo en cuenta que la calificación es una escala del 0 al 5, y que todos los datos parecen encontrarse a partir de 4.2, que no es una mala calificación, podríamos asociar el precio más bajo de los vinos con estas calificaciones "bajasz los precios superiores a los 1500 euros a los que superan la calificación de 4.6. De todas formas, como también podemos ver que hay vinos de bajo precio con mjuy buena calificación, no considero que esta relación directa sea tan relevante entre estas dos variables.

- *Entre las variables price y year*
  El coeficiente de correlación es en este caso -0.3844, que indicaría una relación inversa y no muy relevante entre estas dos variables. Este coeficiente es inferior a 0.4, luego no debemos esperar mucha relación entre las variables. Si fuera más alto, lo que debería ocurrir es que cuando una variable crezca, la otra bajará su valor.
  Veamos una representación gráfica de ambas variables enfrentadas:

```
plot(datos$price, datos$year,  xlab="Precio de la botella de vino",
     ylab = "Año del vino", col="sienna2",
     main="Representación de las variables price y year")
```

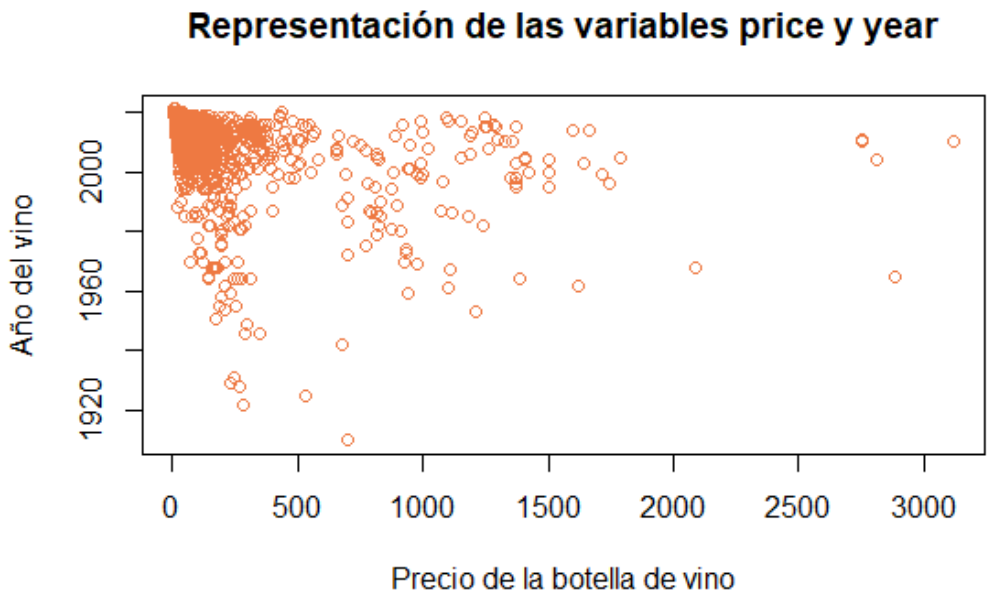


Figura 4.3: Representación gráfica de las variables price y year

Aquí, como sospechábamos, no habría una relación clara que podamos ver en la gráfica. Cabe destacar aun así los precios bajos de las botellas de vino cuando son de años recientes (en su gran mayoría).

Una vez que hemos visto que no hay relaciones entre variables que afecten a nuestro estudio, pasamos a proponer estructuras diferentes de modelos de Redes de Neuronas Artificiales.
Igual que en el ejemplo de clasificación, sobre los datos haremos una partición de entrenamiento(70)/test(30) y trabajaremos con la libraría *h2o*. En el caso de este ejemplo se proponen más modelos diferentes debido a la dificultad de encontrar un modelo adecuado, o que nos proporcione resultados medianamente aceptables.
Marcando los parámetros que tendrán en común todas las etsructuras, vamos a comparar los modelos obtenidos según los diferentes tipos de error que obtengamos y, sobre todo, el coeficiente que obtengamos del R^2.

1. **Modelo con dos capas ocultas de 8 nodos en cada una**
   Vamos a trabajar con una RNA para la predicción de la variable *price* contando con 968 neuronas en la primera capa, ocho en las segunda y tercera (como hemos querido establecer) y 1 neurona en la capa de salida, que nos dará el valor del precio de la botella de vino.

```
modelo_r1 = h2o.deeplearning(
  x = c(1,2,3,4,5,6,8,9,10), #variables predictoras
  y = 7, #variable respuesta
  training_frame = train.hex, #datos de entrenamiento
  validation_frame = test.hex, #datos de validacion/test
  activation = 'RectifierWithDropout',
  hidden = c(8,8), #tamaño de las capas ocultas
  hidden_dropout_ratio = c(0.5, 0.5),
  input_dropout_ratio = 0.2,
  epochs = 50, #iteraciones
  l1 = 1e-5,
  l2 = 1e-5,
  rho = 0.99,
  epsilon = 1e-8)

summary(modelo_r1)
```

   Veamos qué información hemos obtenido del modelo:

   - *Raíz del Error Cuadrático Medio (RMSE):* El obtenido en este modelo es 103.3896, un valor considerablemente alto teniendo en cuenta que el precio medio de la botella de vino es de 67.3977 euros.
   - *Mean Residual Deviance:* toma un valor de 10689.4, un valor extremadamente alto, ya que buscamos que sea lo más cercano a 0 posible.
   - R^2*:* El valor del coeficiente de determinación que hemos obtenido es 0.3457417, es decir, este modelo explica un 34.57417 % de la variabilidad del modelo. Este valor del R^2 no nos da la confianza suficiente en el modelo.

2. **Modelo con una capa oculta y 15 nodos**
   Ahora vamos a trabajar con una RNA para la predicción de la variable *price* contando con 968 neuronas en la primera capa, 15 en la capa oculta (como hemos querido establecer) y 1 neurona en la capa de salida, que nos dará el valor del precio de la botella de vino.

```
modelo_r6 = h2o.deeplearning(
  x = c(1,2,3,4,5,6,8,9,10), #variables predictoras
  y = 7, #variable respuesta
  training_frame = train.hex, #datos de entrenamiento
  validation_frame = test.hex, #datos de validacion/test
  activation = 'RectifierWithDropout',
  hidden = 15, #tamaño de las capas ocultas
  hidden_dropout_ratio =  0.5,
  input_dropout_ratio = 0.2,
  epochs = 50, #iteraciones
  l1 = 1e-5,
  l2 = 1e-5,
  rho = 0.99,
  epsilon = 1e-8)


summary(modelo_r6)
```

Veamos qué información hemos obtenido del modelo:

- *Raíz del Error Cuadrático Medio (RMSE):* El obtenido en este modelo es 77.71608, un valor considerablemente alto (pero menor que en el modelo anterior) teniendo en cuenta que el precio medio de la botella de vino es de 67.3977 euros.
- *Mean Residual Deviance:* toma un valor de 6039.789, un valor que sigue siendo bastante alto, ya que buscamos que sea lo más cercano a 0 posible.
- R^2*:* El valor del coeficiente de determinación que hemos obtenido es 0.6573941, es decir, este modelo explica un 65.73941 % de la variabilidad del modelo. Este valor del R^2 no nos da la confianza suficiente en el modelo, aunque es mejor que el anterior.

3. **Modelo con una capa oculta y 60 nodos**
   Ahora vamos a trabajar con una RNA con 968 neuronas en la primera capa, 60 en la capa oculta (como hemos querido establecer) y 1 neurona en la capa de salida, que nos dará el valor del precio de la botella de vino.

```
modelo_r7 = h2o.deeplearning(
  x = c(1,2,3,4,5,6,8,9,10), #variables predictoras
  y = 7, #variable respuesta
  training_frame = train.hex, #datos de entrenamiento
  validation_frame = test.hex, #datos de validacion/test
  activation = 'RectifierWithDropout',
  hidden = 60, #tamaño de las capas ocultas
  hidden_dropout_ratio =  0.5,
  input_dropout_ratio = 0.2,
  epochs = 50, #iteraciones
  l1 = 1e-5,
  l2 = 1e-5,
  rho = 0.99,
  epsilon = 1e-8)


summary(modelo_r7)
```

Veamos qué información hemos obtenido del modelo:

- *Raíz del Error Cuadrático Medio (RMSE):* El obtenido en este modelo es 63.52925, un valor que sigue siendo considerablemente alto (teniendo en cuenta que el precio medio de la botella de vino es de 67.3977 euros).
- *Mean Residual Deviance:* toma un valor de 4035.965, un valor alto aún, ya que buscamos que sea lo más cercano a 0 posible.
- R^2*:* El valor del coeficiente de determinación que hemos obtenido es 0.7239635, es decir, este modelo explica un 72.39635 % de la variabilidad del modelo. Este valor de R^2 es mejor que los anteriores, no es perfecto pero nos puede llegar a servir.

4. **Modelo con tres capas ocultas y 30 nodos en cada una**
   Por último vamos a trabajar con una RNA con 968 neuronas en la primera capa, 3 capas ocultas con 30 nodos en cada una y 1 neurona en la capa de salida, que nos dará el valor del precio de la botella de vino.

```
modelo_r8 = h2o.deeplearning(
  x = c(1,2,3,4,5,6,8,9,10), #variables predictoras
  y = 7, #variable respuesta
  training_frame = train.hex, #datos de entrenamiento
  validation_frame = test.hex, #datos de validacion/test
  activation = 'RectifierWithDropout',
  hidden = c(30,30,30), #tamaño de las capas ocultas
  hidden_dropout_ratio =  c(0.5,0.5,0.5),
  input_dropout_ratio = 0.2,
  epochs = 50, #iteraciones
  l1 = 1e-5,
  l2 = 1e-5,
  rho = 0.99,
  epsilon = 1e-8)
```

Veamos qué información hemos obtenido del modelo:

- *Raíz del Error Cuadrático Medio (RMSE):* El obtenido en este modelo es 66.12811, un valor que sigue siendo considerablemente alto (teniendo en cuenta que el precio medio de la botella de vino es de 67.3977 euros).
- *Mean Residual Deviance:* toma un valor de 4372.927, valor alto, ya que buscamos que sea lo más cercano a 0 posible.
- R^2*:* El valor del coeficiente de determinación que hemos obtenido es 0.7131944, es decir, este modelo explica un 71.31944 % de la variabilidad del modelo. Este valor de R^2 no es ideal, debería ser más alto aún.

5. **Modelo con dos capas ocultas y 646 nodos en cada una**
   Ahora vamos a trabajar con una RNA con 968 neuronas en la primera capa, 646 en cada capa oculta (como hemos querido establecer) y 1 neurona en la capa de salida, que nos dará el valor del precio de la botella de vino. Para elegir ese número

concreto de neuronas en la capa oculta hemos seguido la regla de tomar 2/3 aproximadamente del total de neuronas que hay en las capas de entrada y salida.[2]

```
modelo_r9 = h2o.deeplearning(
  x = c(1,2,3,4,5,6,8,9,10), #variables predictoras
  y = 7, #variable respuesta
  training_frame = train.hex, #datos de entrenamiento
  validation_frame = test.hex, #datos de validacion/test
  activation = 'RectifierWithDropout',
  hidden = c(646,646), #tamaño de las capas ocultas
  hidden_dropout_ratio = c(0.5, 0.5),
  input_dropout_ratio = 0.2,
  epochs = 50, #iteraciones
  l1 = 1e-5,
  l2 = 1e-5,
  rho = 0.99,
  epsilon = 1e-8)


summary(modelo_r9)
```

Veamos qué información hemos obtenido del modelo:

- *Raíz del Error Cuadrático Medio (RMSE):* El obtenido en este modelo es 47.48021, un valor que sigue siendo considerablemente alto (teniendo en cuenta que el precio medio de la botella de vino es de 67.3977 euros), pero bastante menor en comparación a los anteriores.
- *Mean Residual Deviance:* toma un valor de 2254.371, un valor que es bastante alto, ya que buscamos que sea lo más cercano a 0 posible.
- R^2*:* El valor del coeficiente de determinación que hemos obtenido es 0.8060004, es decir, este modelo explica un 80.60004 % de la variabilidad del modelo. Este valor de R^2 es mejor que los anteriores, y aunque no es mayor que 0.95, que sería lo óptimo, es bastante más acertado que los anteriores.

6. **Modelo con una capa oculta y 646 nodos**
   Ahora vamos a trabajar con una RNA con 968 neuronas en la primera capa, 646 en la capa oculta (como hemos querido establecer) y 1 neurona en la capa de salida, que nos dará el valor del precio de la botella de vino. Para elegir ese número concreto de neuronas en la capa oculta hemos seguido la regla de tomar 2/3 aproximadamente del total de neuronas que hay en las capas de entrada y salida.[2]

```
modelo_r10 = h2o.deeplearning(
  x = c(1,2,3,4,5,6,8,9,10), #variables predictoras
  y = 7, #variable respuesta
  training_frame = train.hex, #datos de entrenamiento
  validation_frame = test.hex, #datos de validacion/test
  activation = 'RectifierWithDropout',
  hidden = 646, #tamaño de las capas ocultas
  hidden_dropout_ratio = 0.5,
  input_dropout_ratio = 0.2,
  epochs = 50, #iteraciones
  l1 = 1e-5,
  l2 = 1e-5,
  rho = 0.99,
  epsilon = 1e-8)
```

Veamos qué información hemos obtenido del modelo:

- *Raíz del Error Cuadrático Medio (RMSE):* El obtenido en este modelo es 57.49214, un valor que sigue siendo considerablemente alto (teniendo en cuenta que el precio medio de la botella de vino es de 67.3977 euros).
- *Mean Residual Deviance:* toma un valor de 3305.347.
- R^2*:* El valor del coeficiente de determinación que hemos obtenido es 0.785514, es decir, este modelo explica un 78.5514 % de la variabilidad del modelo.

4.3.1. Comentarios sobre los resultados obtenidos

Aquí elegir el modelo para resolver este problema es más fácil, ya que el penúltimo destaca por ser el mejor de todos en todos los aspectos **(Modelo con dos capas ocultas y 646 nodos en cada una)**. Cuenta con un R^2 de 0.8060004, que no es ideal pero se acerca mucho a esa perfección que buscamos.

4.4. Resolución en Python

Vamos a resolver esta vez el mismo problema pero en Python. Buscamos ahora una Red Neuronal Artificial que nos ayude a predecir el precio del vino español según las diferentes variables que se han descrito inicialmente. Como contamos con muchas de ellas cualitativas, vamos a estudiarlas por separado antes de crear los modelos.
Primero veamos la matriz de correlaciones entre las variables cuantitativas del conjunto de datos:

	rating	**num_reviews**	**price**	**body**	**acidity**
rating	1	0.015229	0.544809	0.163033	-0.094553
num_reviews	0.015229	1	-0.030083	0.067106	0.040138
price	0.544809	-0.030083	1	0.153624	-0.032870
body	0.163033	0.067106	0.153624	1	-0.017950
acidity	-0.094553	0.040138	-0.032870	-0.017950	1

Cuadro 4.2: Matriz de correlaciones del ejemplo de regresión en Python

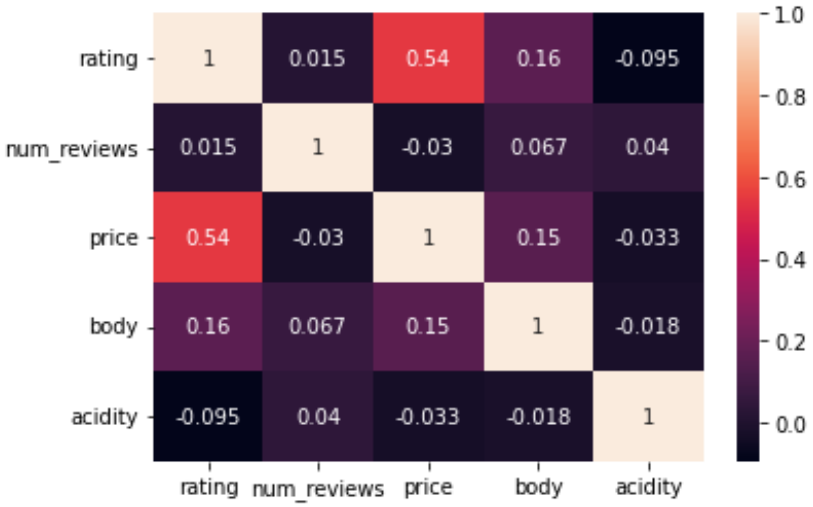


Figura 4.4: Representación gráfica de la matriz de correlaciones

Solo obtendríamos un par de variables cuya correlación es interesante de estudiar:

- *Entre las variables rating y price*
  El coeficiente de correlación asociado es 0.551943, que indicaría una relación directa y no muy fuerte: cuando una variable crezca la otra también lo hará, pero no claramente. veámoslo gráficamente:

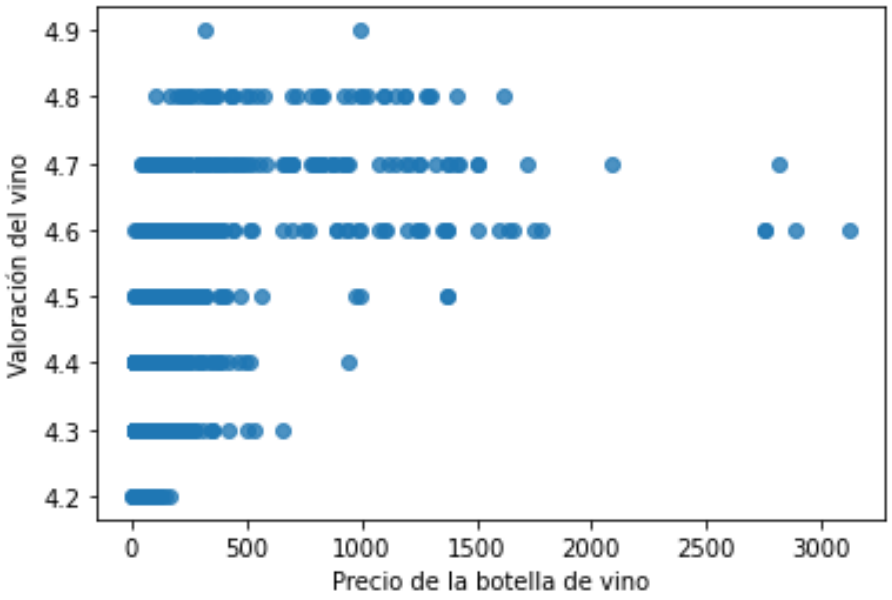


Figura 4.5: Representación gráfica de las variables price y rating

Lo lógico sería que al aumentar la calificación que una botella de vino reciba, el precio de ésta suba, pero en esta gráfica no podemos confirmar que esto ocurre así, ya que hay botellas de bajo precio que obtienen las mayores calificaciones.

Ahora veamos qué ocurre con las variables cualitativas. En la siguente tabla podremos ver cómo se distribuyen dichas variables:

	winery	**wine**	**year**	**region**	**type**
count	7500	7500	7498	7500	6955
unique	480	847	71	76	21
top	*Contino*	*Reserva*	*2011*	*Rioja*	*Rioja Red*
freq	457	467	1190	2440	2357

Cuadro 4.3: Breve estudio de las variables cualitativas del ejemplo de regresión

De este recuento podemos comentar lo siguiente: hay 480 tipos de bodegas diferentes que ofrecen hasta 847 tipos diferentes de vino, siendo el más repetido es el Reserva, que se reptite entre los registros del conjunto de datos hasta 467 veces. El año al que pertenecen la mayoría de los vinos (1190 de 7498 veces) es el año 2011. La gran mayoria de vinos provienen de la región de La Rioja, entre las 76 regiones diferentes que hay.
Una vez tenemos comentadas las variables y vemos que no hay problema para trabajar con ellas, pasamos a estudiar las diferentes estructuras de Redes de Neuronas Artificiales que se proponen:

1. **Modelo con dos capas ocultas de 8 nodos en cada una**
   Obtendríamos aquí una red neuronal con dos capas ocultas que tienen 8 neuronas cada una, con funciones de activación *ReLU* en las capas ocultas y *función lineal* en la capa de salida.

```
modelor1 = MLPRegressor(hidden_layer_sizes=(8,8),
                learning_rate_init=0.01,
                solver = 'adam',
                max_iter = 1000,
                random_state = 7978)

modelor1.fit(X=X_train, y=y_train)
```

   Para medir la efectividad de estos modelos miraremos sobre todo el valor que obtenemos como R^2:

   - R^2*:* El valor del coeficiente de determinación es muy bajo, 0.201510123, que no nos da ninguna confianza sobre el modelo que acabamos de crear.
   - *RMSE:* Obtenemos una Raíz del Error Cuadrático Medio bastante alto teniendo en cuenta que la media de los datos se encuentra en torno a 67 euros: 173.925 euros es el RMSE obtenido.

2. **Modelo con una capa oculta y 15 nodos**
   Obtendríamos aquí una red neuronal con una capa oculta que cuenta con 15 neuronas, con función de activación *ReLU* en la capa oculta y *función lineal* en la capa de salida.

```
modelor6 = MLPRegressor(hidden_layer_sizes=(15),
                learning_rate_init=0.01,
                solver = 'adam',
                max_iter = 1000,
                random_state = 7978)

modelor6.fit(X=X_train, y=y_train)
```

   Para medir la efectividad de estos modelos miraremos el valor que obtenemos de R^2 y RMSE:

   - R^2*:* El valor del coeficiente de determinación lo obtenemos tras hacer las predicciones sobre el conjunto test en el modelo que acabamos de crear, y lo que obtenemos no es un valor bueno: el valor de R^2 es 0.212967, ligeramente mejor que el anterior pero sigue sin ser de confianza.

```
#Capacidad de predicción
predictions6 = modelor6.predict(X_test)

rmse6 = mean_squared_error(
        y_true = y_test,
        y_pred = predictions6,
        squared = False)
rmse6
```

- *RMSE:* Obtenemos un RMSE similar al anterior, 172.672826, y sigue siendo bastante alto para definir nuestro modelo como bueno.

3. **Modelo con una capa oculta y 60 nodos**
   Aquí aumentamos el número de nodos en la capa oculta, pasamos a tener 60, con función de activación *ReLU* en lesta capa y *función lineal* en la capa de salida.

```
modelor7 = MLPRegressor(hidden_layer_sizes=(60),
                learning_rate_init=0.01,
                solver = 'adam',
                max_iter = 1000,
                random_state = 7978)

modelor7.fit(X=X_train, y=y_train)
```

Para medir la efectividad de estos modelos miraremos el valor que obtenemos como R^2:

- R^2*:* El valor del coeficiente de determinación sigue siendo muy bajo, 0.2526115, que no nos da ninguna confianza sobre el modelo que acabamos de crear, pero es el mejor hasta ahora.

```
#Capacidad de predicción
predictions7 = modelor7.predict(X_test)

rmse7 = mean_squared_error(
        y_true = y_test,
        y_pred = predictions7,
        squared = False)
rmse7
```

- *RMSE:* 168.26769 es el valor de la Raíz del Error Cuadrático Medio obtenido mediante este modelo, que sigue siendo demasiado alto, ya que lo ideal es que se acerque a 0.

4. **Modelo con tres capas ocultas y 30 nodos en cada una**
   Vamos a probar ahora un modelo que cuente con 3 capas ocultas y 30 nodos en cada una, con funciones de activación *ReLU* en la capa oculta y *función lineal* en la capa de salida.

```
modelor8 = MLPRegressor(hidden_layer_sizes=(30,30,30),
                learning_rate_init=0.01,
                solver = 'adam',
                max_iter = 1000,
                random_state = 7978)

modelor8.fit(X=X_train, y=y_train)
```

Para medir la efectividad de estos modelos miraremos el valor que obtenemos como R^2:

- R^2*:* Obtenemos el mejor coeficiente de determinación hasta ahora: 0.2740399. Sigue siendo un valor extremadamente bajo y así no nos interesa trabajar con este modelo.

```
#Capacidad de predicción
predictions8 = modelor8.predict(X_test)

rmse8 = mean_squared_error(
        y_true = y_test,
        y_pred = predictions8,
        squared = False)
rmse8
```

- *RMSE:* Obtenemos un RMSE de 165.8379, valor que es mejor qur los anteriores pero sigue muy lejano a 0.

5. **Modelo con dos capas ocultas y 646 nodos en cada una**
   [2]El siguiente número de nodos en la capa oculta lo vamos a probar gracias a la regla que nos dice que el número ideal de nodos en la capa oculta es 2/3 de la suma del número total de nodos en la capa de entrada y la de salida, pero esta vez probamos en dos capas ocultas. Este modelo lo hemos probado antes en RStudio y nos ha dado buen resultado.

```
modelor9 = MLPRegressor(hidden_layer_sizes=(646,646),
                learning_rate_init=0.01,
                solver = 'adam',
                max_iter = 1000,
                random_state = 7978)

modelor9.fit(X=X_train, y=y_train)
```

Veamos que ocurre aquí con el coeficiente de determinación:

- R^2*:* Hemos obtenido un coeficiente demasiado bajo, 0.2208621, luego parece que esta norma en este caso no sirve.
- *RMSE:* El valor que obtenemos aquí es 171.80457.

6. **Modelo con una capa oculta y 646 nodos**
   [2]El siguiente número de nodos en la capa oculta lo vamos a probar gracias a la regla que nos dice que el número ideal de nodos en la capa oculta es 2/3 de la suma del número total de nodos en la capa de entrada y la de salida. Este modelo lo hemos probado antes en RStudio y nos ha dado buen resultado.

```
modelor10 = MLPRegressor(hidden_layer_sizes=(646),
                learning_rate_init=0.01,
                solver = 'adam',
                max_iter = 1000,
                random_state = 7978)

modelor10.fit(X=X_train, y=y_train)
```

Veamos que ocurre aquí con el coeficiente de determinación:

- R^2*:* Hemos obtenido un coeficiente demasiado bajo, 0.2123897, luego parece que esta norma en este caso no sirve.
- *RMSE:* En este modelo el valor del Error Cuadrático Medio es 172.73615, demasiado alto para que sea fiable.

4.4.1. Comentarios sobre los resultados obtenidos

No hemos obtenido ningún modelo que resuelva bien nuestro problema, ya que ninguno de los coeficientes de determinación supera el 30 %. Si tuviera que elegir alguno me quedaría con el que tiene mayor coeficiente **(Modelo con tres capas ocultas y 30 nodos en cada una)**, pero aún así no es un modelo que me guste.
Este fallo en la determinación de un modelo adecuado a este problema que se presenta se puede deber a la necesidad de contar con más registros, ya que contamos en este conjunto de datos con muchas variables cualitativas que toman valores demasiado diferentes como para establecer una norma correctamente.

4.5. Comparación de eficiencia de ambos programas

Hemos obtenido para RStudio que el mejor modelo es el quinto: **Modelo con dos capas ocultas y 646 nodos en cada una**. Según los modelos de Python, destaca el cuarto: **Modelo con tres capas ocultas y 30 nodos en cada una**.
El modelo de RStudio cuenta con un RMSE de 47.48021 y un R^2 de 0.8060004, y el modelo de Python tiene un RMSE de 165.8379 y un valor de R^2 de 0.2740399. Claramente el modelo que resuelve nuestro problema en RStudio es el mejor de todos los propuestos en ambos programas.

Capítulo 5

Ejemplo de clasificación de imágenes

5.1. Planteamiento del problema

El conjunto de datos Fashion MNIST cuenta con 70.000 imágenes en escala de grises (60.000 de ellas para el conjunto de entrenamiento y 10.000 para el conjunto test). Cada imágen es de tamaño 28x28 y representa una prenda de vestir numerada del 0 al 9:

- 0 - Camiseta/top
- 1 - Pantalones
- 2 - Pullover
- 3 - Vestido
- 4 - Abrigo
- 5 - Sandalia
- 6 - Camisa
- 7 - Zapatilla de deporte
- 8 - Bolso
- 9 - Botín

El objetivo es utilizar las librerías necesarias para que nuestros modelos sepan identificar correctamente el tipo de prenda que es cada imagen.

5.2. Comentarios sobre la base de datos

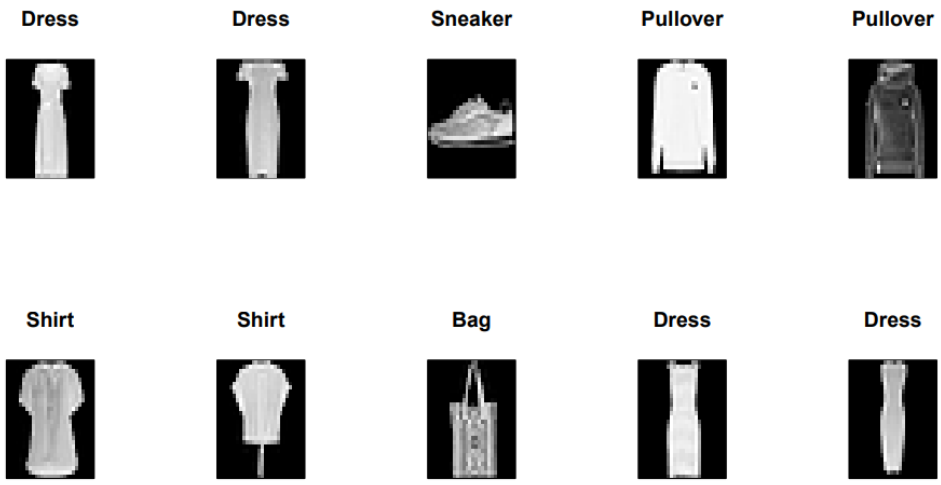
Como las imágenes sonde 28x28 píxeles, el conjunto de datos tiene la siguiente forma:

- 784 variables que indican el grado de oscuridad en una escala de grises (de 0 a 255).
- Una variable que indica el tipo de prenda que es.

5.3. Resolución en RStudio

El conjunto de datos Fashion MNIST se encuentra dentro de una librería propia de R. Obtenemos directamente la separación en conjunto de entrenamiento y conjunto test, donde se cunmple que en cada uno de estos conjuntos hay para cada tipo de prenda el mismo número de imágenes asociadas. Así, el conjunto de imágenes de entrenamiento cuenta con 60.000 prendas (6.000 para cada una), y el conjunto test cuenta con 10.000 prendas (1.000 para cada una).
Cada píxel de cada imagen viene representado por una variable que toma un valor del 0 al 255, para hacerlo más fácil de manejar, vamos a ajustar los valores a una escala de 0 a 1:

```
#Dibujamos 10 prendas
par(mfrow=c(2,5))
for (i in 51:60) {
  img <- train_images[i, , ]
  #para que no salga "tumbado"
  img <- t(apply(img, 2, rev))
  image(1:28, 1:28, img, col = gray((0:255)/255),
        xlab="",ylab="",
        xaxt = 'n', yaxt = 'n',
        main = paste(class_names[train_labels[i] + 1]))
}
```



Una vez tenemos los conjuntos bien preparados podemos comenzar a trabajar con la librería H2O para crear diferentes modelos. Para ello debemos de tener en cuenta que la capa de entrada contará con 784 neuronas (tantas como píxeles), y la de salida, con 10 neuronas (tantas como clases de prendas):

1. **Modelo con dos capas ocultas de 100 nodos cada una** La estructura que seguirá esta red es (784-100-10), con función de activación en la capa de salida *Softmax* y obtenemos los siguientes resultados:

   - *RMSE*: La Raíz del error Cuadrático Medio obtenida es 0.3235146, cercano a 0 pero aun así no lo suficiente.
   - *Tabla de confusión*:

	0	***1***	***2***	***3***	***4***	***5***	***6***	***7***	***8***	***9***
0	849	0	4	23	6	0	106	0	12	0
1	5	960	2	26	4	0	2	0	1	0
2	19	0	748	12	129	0	90	0	2	0
3	31	5	2	879	44	0	33	0	6	0
4	1	1	69	20	847	0	58	0	4	0
5	0	0	0	1	0	960	0	28	2	9
6	141	0	76	26	78	0	663	0	16	0
7	0	0	0	0	0	21	0	953	0	26
8	0	0	1	6	5	3	13	5	967	0
9	0	0	0	0	0	16	1	50	1	932

Cuadro 5.1: Tabla de confusión del primer modelo propuesto

   Por lo general se han clasificado bien las prendas, pero como es obvio hay algunos fallos (después veremos el porcentaje de acierto en la clasificación).
   Por ejemplo, la prenda mejor clasificada ha sido la 8 (bolso), y ha sido confundida sobre todo con camisas (13 errores), vestidos (6 errores) y y abrigos y zapatillas de deportes (5 veces cada uno).
   Por otro lado, el modelo se ha equivocado bastante más en la prenda çamisa", por su similitud a camisetas, pullovers o abrigos.

   - *Capacidad de acierto*: Expresado en porcentaje, la capacidad de acierto de este modelo es de 87.58 %

2. **Modelo con una capa oculta de 100 nodos** La estructura que seguirá esta red es (784-100-10), con función de activación en la capa de salida *Softmax* y obtenemos los siguientes resultados:

- *RMSE*: La Raíz del error Cuadrático Medio de este modelo es 0.2941131, cercano a 0, que es lo que buscamos, aunque no es un valor ideal.

- *Tabla de confusión*:

	0	***1***	***2***	***3***	***4***	***5***	***6***	***7***	***8***	***9***
0	859	1	11	31	1	0	86	0	11	0
1	4	964	1	24	3	0	4	0	0	0
2	23	0	765	8	138	1	61	0	4	0
3	21	6	10	875	46	1	34	0	7	0
4	0	0	71	20	854	0	51	0	4	0
5	0	0	0	0	0	940	0	32	2	26
6	161	1	97	29	97	0	598	0	17	0
7	0	0	0	0	0	17	0	953	1	29
8	2	0	3	3	6	4	11	4	967	0
9	0	0	0	0	0	10	1	41	0	948

Cuadro 5.2: Tabla de confusión del segundo modelo propuesto

La clasificación está bien, aunque tiene sus fallos como es de esperar. Por ejemplo, la prenda mejor clasificada es la 8, como el en modelo anterior, pero es interesante fijarse en que, aunque se clasifiquen bien el mismo número de bolsos, los errores no se cometen por igual: este modelo ha confundido un bolso con uns camisa hasta 11 veces, y con un abrigo, 4. Por otro lado, donde más se ha equivocado este modelo en clasificar en la prenda número 6: la camisa, acertando sólo 598 veces de 1000.

- *Capacidad de acierto*: Expresado en porcentaje, la capacidad de acierto de este modelo es de 87.23 %

3. **Modelo con una capa oculta de 530 nodos** La estructura que seguirá esta red es (784-100-10), con función de activación en la capa de salida *Softmax* y obtenemos los siguientes resultados:

- *RMSE*: La Raíz del error Cuadrático Medio de este modelo es 0.3058115, cercano a 0, que es lo que buscamos, aunque no es un valor ideal.

- *Tabla de confusión*:

	0	***1***	***2***	***3***	***4***	***5***	***6***	***7***	***8***	***9***
0	853	1	12	16	5	1	121	0	9	0
1	1	974	1	18	2	0	3	0	1	0
2	12	1	825	11	93	1	55	0	2	0
3	18	3	12	899	36	1	28	0	3	0
4	0	0	91	25	822	0	60	0	2	0
5	0	0	0	0	0	964	0	24	1	11
6	119	1	96	27	62	0	683	0	12	0
7	0	0	0	0	0	15	0	938	1	46
8	2	0	4	3	5	5	8	3	970	0
9	0	0	0	0	0	16	1	25	0	958

Cuadro 5.3: Tabla de confusión del tercer modelo propuesto

Aquí la prenda mejor clasificada es otra: los pantalones (1). Se ha clasificado correctamente 974 veces, sólo confundiéndose 18 veces con un vestido (3), 3 veces con una camisa (6), 2 veces con un abrigo, y una vez con una camiseta (0), un pullover (2) y un bolso (8).
La prenda peor clasificada vuelve a ser la camisa, acertando 683 veces de 1000 camisas que había en total. Con la prenda que más se ha confundido la camisa es con la camiseta (hasta 119 veces).

- *Capacidad de acierto*: Expresado en porcentaje, la capacidad de acierto de este modelo es de 88.68 %

4. **Modelo con dos capas ocultas de 530 nodos cada una** La estructura que seguirá esta red es (784-530-10), con función de activación en la capa de salida *Softmax* y obtenemos los siguientes resultados:

   - *RMSE*: La Raíz del Error Cuadrático Medio de este modelo es 0.2941131, cercano a 0, que es lo que buscamos, aunque no es un valor ideal.
   - *Tabla de confusión*: Por lo general clasifica bien las prendas, pero como es lógico y las prendas pueden tomar muchas formas, algunas han sido clasificadas mal. Por ejemplo, la prenda 6, que es una camisa, se ha confundido 124 veces con la prenda 0 (camiseta) y 76 veces con la prenda 2 (pullover). Tienen sentido estos errores ya que al final todas estas prendas son similares.
     Por otro lado la prenda que mejor se ha clasificado ha sido la categorizada como 8 (bolsos), lo cual tiene mucho sentido. Ha clasificado correctamente 979 de 1000 bolsos que había, pero ha confundido 6 de ellos con un vestido, 5 de ellos con un abrigo, 3 con sandalias, 3 con zapatillas de deporte, 2 con camisetas y 1 con pullovers y camisas.

	0	***1***	***2***	***3***	***4***	***5***	***6***	***7***	***8***	***9***
0	850	0	8	18	6	0	107	1	10	0
1	1	970	1	20	3	0	4	0	1	0
2	17	0	796	9	110	1	67	0	0	0
3	26	3	8	897	33	1	29	0	3	0
4	1	0	56	25	882	0	33	0	3	0
5	0	0	0	0	0	946	0	35	1	19
6	124	1	76	18	74	0	697	0	10	0
7	0	0	0	0	0	6	0	969	0	25
8	2	0	1	6	5	3	1	3	979	0
9	0	0	0	0	0	7	1	39	0	953

Cuadro 5.4: Tabla de confusión del cuarto modelo propuesto

- *Capacidad de acierto*: Expresado en porcentaje, la capacidad de acierto de este modelo es de 89.39 %

5.3.1. Comentarios sobre los resultados obtenidos

Los cuatro modelos han resultado similares en cuanto a la capacidad de acierto en las prendas, todas se encuentran entre un 87 y un 90 por ciento. Sí que es escierto que en este caso la correcta clasificación ha sido mejor en el cuarto modelo. Además este modelo cuenta con la menor Raíz del Error Cuadrático Medio, luego el **Modelo con dos capas ocultas de 530 nodos cada una** es el que mejor trabaja de todos, aunque no con una gran diferencia.

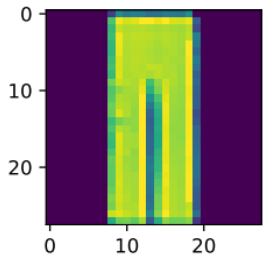
5.4. Resolución en Python

Esta vez no vamos a trabajar con las bibliotecas de los anteriores ejemplos. Como para el tratamiento de imágenes lo óptimo es utilizar una red convolucional, vamos a trabajar con *keras*, tanto para descargar los datos como para trabajar en los modelos.
Como ocurre en RStudio, lo primero que debemos hacer con los datos es redimensionarlos y ponerlos en escala de grises de 0 a 1 (en lugar de 0 a 255).

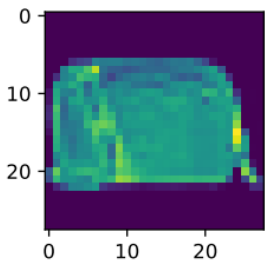
```
# Redimensionamos los datos
X_train = X_train.reshape((X_train.shape[0], 28 * 28 * 1))
X_test = X_test.reshape((X_test.shape[0], 28 * 28 * 1))
    #Normalizamos los datos
X_train = X_train.astype('float32') / 255.0
X_test = X_test.astype('float32') / 255.0
```

Vamos a ver alguna de estas imágenes que contiene el conjunto de entrenamiento:

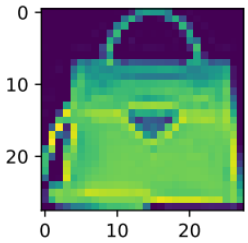
```
plt.imshow(X_train[151].reshape(28,28), cmap='viridis', interpolation='none')
```




```
plt.imshow(X_train[152].reshape(28,28), cmap='viridis', interpolation='none')
```




```
plt.imshow(X_train[156].reshape(28,28), cmap='viridis', interpolation='none')
```



Una vez sabemos la estructura de los conjuntos de datos (iguales a los de RStudio) y hemos visto gráficamente como pueden ser, pasamos a bscar algún modelo que lo explique correctamente.

1. **Modelo con dos capas ocultas de 100 nodos cada una**
   Para construir este modelo que tiene la estructura (784-100-100-10) debemos ir escribiendo una a una las capas ocultas y la de salida, y dejar indicado el tamaño de la capa de entrada:

```
modelci1 = Sequential()
modelci1.add(Dense(100, input_shape=(28 * 28 * 1,), activation='relu')) #primera capa oculta
modelci1.add(Dense(100, activation='relu')) #segunda capa oculta
modelci1.add(Dense(10, activation='softmax')) #capa de salida
```

   Una vez tenemos la estructura de la red tenemos que entrenarla. Utilizaremos el solver *Adam*, que es el que está programado por defecto y hemos utilizado anteriormente.

```
  #Entrenamos la red
adam = Adam()
modelci1.compile(loss='categorical_crossentropy', optimizer=adam, metrics=['accuracy'])
modelci_1 = modelci1.fit(X_train, y_train, validation_data=(X_test, y_test), epochs=50)
```

   Una vez tenemos ajustado el modelo con los conjuntos de entrenamiento y test, vamos a realizar las predicciones y ver la *capacidad de acierto* en cada prenda:

```
print(classification_report(y_test.argmax(axis=1),
                            predictions.argmax(axis=1),
                            target_names=['T-shirt/top', 'Trouser', 'Pullover', 'Dress',
                            'Coat', 'Sandal', 'Shirt', 'Sneaker', 'Bag', 'Ankle boot']))
```

	Precisión	***f1-score***
Camiseta/top	0.8	0.83
Pantalón	0.98	0.98
Pullover	0.77	0.81
Vestido	0.91	0.90
Abrigo	0.84	0.82
Sandalia	0.97	0.97
Camisa	0.76	0.70
Zapatillas de deporte	0.94	0.96
Bolso	0.97	0.97
Botín	0.97	0.96
Accuracy		0.89

Cuadro 5.5: Informe de clasificación del primer modelo propuesto

Aquí podemos apreciar como la prenda que mejor se clasifica es el pantalón, seguido del bolso y el botín, con casi un 100 % de acierto. Sin embargo, la prenda que peor se clasifica es la camisa.

```
score = modelci1.evaluate(X_test,y_test,verbose=0)
print('Test Accuracy : {:.4f}'.format(score[1]))
```

Por último, cabe destacar que el valor del *coeficiente del test de acierto/precisión* es de 0,8904.

2. **Modelo con una capa oculta de 100 nodos**
   Para construir este modelo que tiene la estructura (784-100-10) debemos ir escribiendo una a una las capas ocultas y la de salida, y dejar indicado el tamaño de la capa de entrada:

```
modelci2 = Sequential()
modelci2.add(Dense(100, input_shape=(28 * 28 * 1,), activation='relu')) #primera capa oculta
modelci2.add(Dense(10, activation='softmax')) #capa de salida
```

Una vez tenemos la estructura de la red tenemos que entrenarla. Utilizaremos el solver *Adam*, que es el que está programado por defecto y hemos utilizado anteriormente.

```
  #Entrenamos la red
adam = Adam()
modelci2.compile(loss='categorical_crossentropy', optimizer=adam, metrics=['accuracy'])
modelci_2 = modelci2.fit(X_train, y_train, validation_data=(X_test, y_test), epochs=50)
```

Una vez tenemos ajustado el modelo con los conjuntos de entrenamiento y test, vamos a realizar las predicciones y ver la *capacidad de acierto* en cada prenda:

```
print(classification_report(y_test.argmax(axis=1),
                            predictions2.argmax(axis=1),
                            target_names=['T-shirt/top', 'Trouser', 'Pullover', 'Dress',
                            'Coat', 'Sandal', 'Shirt', 'Sneaker', 'Bag', 'Ankle boot']))
```

	Precisión	*f1-score*
Camiseta/top	0.83	0.83
Pantalón	0.98	0.98
Pullover	0.78	0.79
Vestido	0.92	0.88
Abrigo	0.74	0.81
Sandalia	0.97	0.96
Camisa	0.77	0.70
Zapatillas de deporte	0.91	0.94
Bolso	0.98	0.97
Botín	0.96	0.95
Accuracy		0.88

Cuadro 5.6: Informe de clasificación del segundo modelo propuesto

Aquí las prendas que mejor se clasifican son el pantalón y el bolso, seguido de la sandalia y el botín, con casi un 100 % de acierto. Sin embargo, las prendas que peor se clasifican son la camisa y el abrigo.

```
score2 = modelci2.evaluate(X_test,y_test,verbose=0)
print('Test Accuracy : {:.4f}'.format(score2[1]))
```

Por último, cabe destacar que el valor del *coeficiente del test de acierto/precisión* es de 0,8823.

3. **Modelo con una capa oculta de 530 nodos**
   Para construir este modelo que tiene la estructura (784-530-10) debemos ir escribiendo una a una las capas ocultas y la de salida, y dejar indicado el tamaño de la capa de entrada. El tamaño de la capa oculta se ha decidido tomando como regla la de los 2/3 del total de neuronas en las capas de entrada y salida.

```
modelci3 = Sequential()
modelci3.add(Dense(530, input_shape=(28 * 28 * 1,), activation='relu')) #primera capa oculta
modelci3.add(Dense(10, activation='softmax')) #capa de salida
```

Una vez tenemos la estructura de la red tenemos que entrenarla. Utilizaremos el solver *Adam*, que es el que está programado por defecto y hemos utilizado anteriormente.

```
  #Entrenamos la red
adam = Adam()
modelci3.compile(loss='categorical_crossentropy', optimizer=adam, metrics=['accuracy'])
modelci_3 = modelci3.fit(X_train, y_train, validation_data=(X_test, y_test), epochs=50)
```

Una vez tenemos ajustado el modelo con los conjuntos de entrenamiento y test, vamos a realizar las predicciones y ver la *capacidad de acierto* en cada prenda:

```
print(classification_report(y_test.argmax(axis=1),
                            predictions3.argmax(axis=1),
                            target_names=['T-shirt/top', 'Trouser', 'Pullover', 'Dress',
                            'Coat', 'Sandal', 'Shirt', 'Sneaker', 'Bag', 'Ankle boot']))
```

	Precisión	***f1-score***
Camiseta/top	0.88	0.80
Pantalón	0.97	0.98
Pullover	0.81	0.80
Vestido	0.92	0.90
Abrigo	0.82	0.81
Sandalia	0.97	0.97
Camisa	0.63	0.70
Zapatillas de deporte	0.97	0.96
Bolso	0.98	0.97
Botín	0.96	0.96
Accuracy		0.88

Cuadro 5.7: Informe de clasificación del tercer modelo propuesto

Aquí la prenda que mejor se clasifica es el bolso, seguido de las zapatillas de deporte, la sandalia y el pantalón, con casi un 100 % de acierto. Sin embargo, la prenda que peor se clasifica vuelve a ser la camisa.

```
score3 = modelci3.evaluate(X_test,y_test,verbose=0)
print('Test Accuracy : {:.4f}'.format(score3[1]))
```

Por último, cabe destacar que el valor del *coeficiente del test de acierto/precisión* es de 0,8848.

4. **Modelo con dos capas ocultas de 530 nodos cada una**
   Para construir este modelo que tiene la estructura (784-530-530-10) debemos ir

escribiendo una a una las capas ocultas y la de salida, y dejar indicado el tamaño de la capa de entrada. El tamaño de las capas ocultas se han decidido tomando como regla la de los 2/3 del total de neuronas en las capas de entrada y salida.

```
modelci4 = Sequential()
modelci4.add(Dense(530, input_shape=(28 * 28 * 1,), activation='relu')) #primera capa oculta
modelci4.add(Dense(530, activation='relu')) #segunda capa oculta
modelci4.add(Dense(10, activation='softmax')) #capa de salida
```

Una vez tenemos la estructura de la red tenemos que entrenarla. Utilizaremos el solver *Adam*, que es el que está programado por defecto y hemos utilizado anteriormente.

```
  #Entrenamos la red
adam = Adam()
modelci4.compile(loss='categorical_crossentropy', optimizer=adam, metrics=['accuracy'])
modelci_4 = modelci4.fit(X_train, y_train, validation_data=(X_test, y_test), epochs=50)
```

Una vez tenemos ajustado el modelo con los conjuntos de entrenamiento y test, vamos a realizar las predicciones y ver la *capacidad de acierto* en cada prenda:

```
print(classification_report(y_test.argmax(axis=1),
                            predictions4.argmax(axis=1),
                            target_names=['T-shirt/top', 'Trouser', 'Pullover', 'Dress',
                            'Coat', 'Sandal', 'Shirt', 'Sneaker', 'Bag', 'Ankle boot']))
```

	Precisión	***f1-score***
Camiseta/top	0.84	0.84
Pantalón	1	1
Pullover	0.80	0.83
Vestido	0.89	0.91
Abrigo	0.85	0.83
Sandalia	0.98	0.97
Camisa	0.74	0.72
Zapatillas de deporte	0.94	0.95
Bolso	0.99	0.98
Botín	0.96	0.95
Accuracy		0.90

Cuadro 5.8: Informe de clasificación del cuarto modelo propuesto

Este modelo parece que clasifica las prendas mejor. Ha acertado todos los pantalones correctamente, y el 99 % de los bolsos. por otro lado, la prenda en la que ha

cometido más fallos ha sido en la camisa, igual que ocurre en los otros modelos. Eso sí, el porcentaje de fallo es mejor que otros modelos.

```
score3 = modelci3.evaluate(X_test,y_test,verbose=0)
print('Test Accuracy : {:.4f}'.format(score3[1]))
```

Por último, cabe destacar que el valor del *coeficiente del test de acierto/precisión* es de 0,8982.

5.4.1. Comentarios sobre los resultados obtenidos

Aunque los cuatro modelos tienen resultados similares, el que por lo general en todas las prendas tiene mejor resultado, y por ende mejor coeficiente del test de acierto es el cuarto modelo: **Red con dos capas ocultas de 530 nodos cada una**.

5.5. Comparación de eficiencia de ambos programas

Tanto en RStudio como en Python hemos obtenido como mejor modelo el cuarto: **Red con dos capas ocultas de 530 nodos cada una**. Obtienen resultados muy similares (la capacidad de acierto del modelo en Python es levemente mayor). Si debemos decantarnos por uno de los dos, el mejor sería el de Python ya que los tiempos de trabajo en el programa son mucho menores.

Bibliografía

[1] HEATON, J. (2015). *Artificial Intelligence for Humans, Volume 3: Deep Learning and Neural Networks.*

[2] BEYSOLOW II (2017). *Introduction to Deep Learning Using R, A Step-by-Step Guide to Learning and Implementing Deep Learning Models Using R*. San Francisco, California.

[3] GHATAK, A. (2019). *Deep Learning with R*. Singapore.

[4] MATICH, D. (2001). *Redes Neuronales: Conceptos Básicos y Aplicaciones.*

[5] GRAUPE, D. (2013). *Principles of Artificial Neural Networks (3rd ed.)*. University of Illinois, Chicago, USA.

[6] SILAPARASETTY, V. (2020). *Deep Learning Projects Using TensorFlow 2, Neural Network Development with Python and Keras*. Bangalore, India.

[7] FLÓREZ LÓPEZ, R. & FERNÁNDEZ FERNÁNDEZ, J. (2008). *Las Redes Neuronales Artificiales, Fundamentos teóricos y aplicaciones prácticas*. Universidad de León.

[8] CANDEL, A. & LEDELL, E. (2018) *Deep Learning with H2O*, Sexta Edición

[9] PEDREGOSA, F., VAROQUAUX, G., GRAMFORT, A., MICHEL, V. & THIRION, B. (2011) *Scikit-learn: Machine Learning in Python*

[10] BASOGAIN OLABE, X. (1998). *Redes Neuronales Artificiales y sus aplicaciones*. Escuela Superior de Ingeniería de Bilbao.

[11] CALVO, D. (2022). *Clasificación de redes neuronales artificiales* Disponible en https://www.diegocalvo.es/clasificacion-de-redes-neuronales-artificiales/

[12] (2022). *¿Qué es el aprendizaje profundo? - Cómo funciona* Disponible en https://www.netapp.com/es/artificial-intelligence/what-is-deep-learning/

[13] (2022). *What is Artificial Intelligence (AI)? | Glossary*. Disponible en https://www.hpe.com/es/es/what-is/artificial-intelligence.html

[14] (2022). *¿Qué es Aprendizaje profundo (deep learning)?* Disponible en https://www.computerweekly.com/es/definicion/Aprendizaje-profundo-deep-learning

[15] Microsoft. (2022, 27 abril) *Aprendizaje profundo frente a aprendizaje automático en Azure Machine Learning*. Disponible en https://docs.microsoft.com/es-es/azure/machine-learning/concept-deep-learning-vs-machine-learning

[16] DataSmarts Español. (2020, 2 julio). *Redes Neuronales Convolucionales en Profundidad* Disponible en https://datasmarts.net/es/redes-neuronales-convolucionales-en-profundidad/

Ejemplo de clasificación - Transacciones fraudulentas en RStudio

Raquel Beltrán Barba

DETECCIÓN DE TRANSACCIONES FRAUDULENTAS EN TARJETAS DE CRÉDITO

Primero cargamos las librerías que vamos a necesitar:

```
set.seed(07978) #semilla
library(readr)
library(h2o)
library(RSNNS)
library(corrplot)
library(ROCR)
library(kableExtra)
```

Lectura y modificación del conjunto de datos

```
datos = read_csv("creditcard.csv")
datos = as.data.frame(datos)
datos[,31] = as.factor(datos[,31])
str(datos)
```

```
## 'data.frame':    284807 obs. of  31 variables:
##  $ Time  : num  0 0 1 1 2 2 4 7 7 9 ...
##  $ V1    : num  -1.36 1.192 -1.358 -0.966 -1.158 ...
##  $ V2    : num  -0.0728 0.2662 -1.3402 -0.1852 0.8777 ...
##  $ V3    : num  2.536 0.166 1.773 1.793 1.549 ...
##  $ V4    : num  1.378 0.448 0.38 -0.863 0.403 ...
##  $ V5    : num  -0.3383 0.06 -0.5032 -0.0103 -0.4072 ...
##  $ V6    : num  0.4624 -0.0824 1.8005 1.2472 0.0959 ...
##  $ V7    : num  0.2396 -0.0788 0.7915 0.2376 0.5929 ...
##  $ V8    : num  0.0987 0.0851 0.2477 0.3774 -0.2705 ...
##  $ V9    : num  0.364 -0.255 -1.515 -1.387 0.818 ...
##  $ V10   : num  0.0908 -0.167 0.2076 -0.055 0.7531 ...
##  $ V11   : num  -0.552 1.613 0.625 -0.226 -0.823 ...
##  $ V12   : num  -0.6178 1.0652 0.0661 0.1782 0.5382 ...
##  $ V13   : num  -0.991 0.489 0.717 0.508 1.346 ...
##  $ V14   : num  -0.311 -0.144 -0.166 -0.288 -1.12 ...
##  $ V15   : num  1.468 0.636 2.346 -0.631 0.175 ...
##  $ V16   : num  -0.47 0.464 -2.89 -1.06 -0.451 ...
##  $ V17   : num  0.208 -0.115 1.11 -0.684 -0.237 ...
##  $ V18   : num  0.0258 -0.1834 -0.1214 1.9658 -0.0382 ...
##  $ V19   : num  0.404 -0.146 -2.262 -1.233 0.803 ...
```

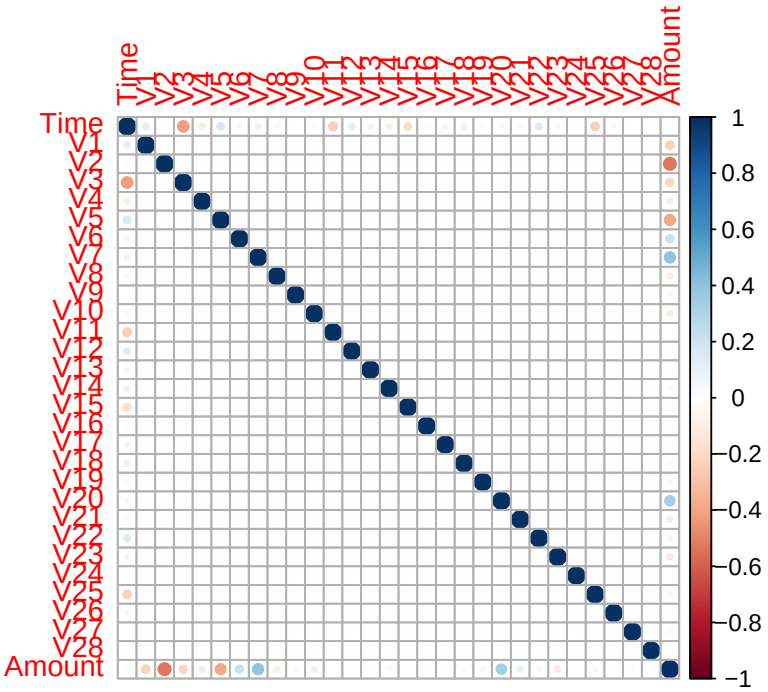
```
##  $ V20   : num  0.2514 -0.0691 0.525 -0.208 0.4085 ...
##  $ V21   : num  -0.01831 -0.22578 0.248 -0.1083 -0.00943 ...
##  $ V22   : num  0.27784 -0.63867 0.77168 0.00527 0.79828 ...
##  $ V23   : num  -0.11 0.101 0.909 -0.19 -0.137 ...
##  $ V24   : num  0.0669 -0.3398 -0.6893 -1.1756 0.1413 ...
##  $ V25   : num  0.129 0.167 -0.328 0.647 -0.206 ...
##  $ V26   : num  -0.189 0.126 -0.139 -0.222 0.502 ...
##  $ V27   : num  0.13356 -0.00898 -0.05535 0.06272 0.21942 ...
##  $ V28   : num  -0.0211 0.0147 -0.0598 0.0615 0.2152 ...
##  $ Amount: num  149.62 2.69 378.66 123.5 69.99 ...
##  $ Class : Factor w/ 2 levels "0","1": 1 1 1 1 1 1 1 1 1 1 ...
```

Estudio de las variables

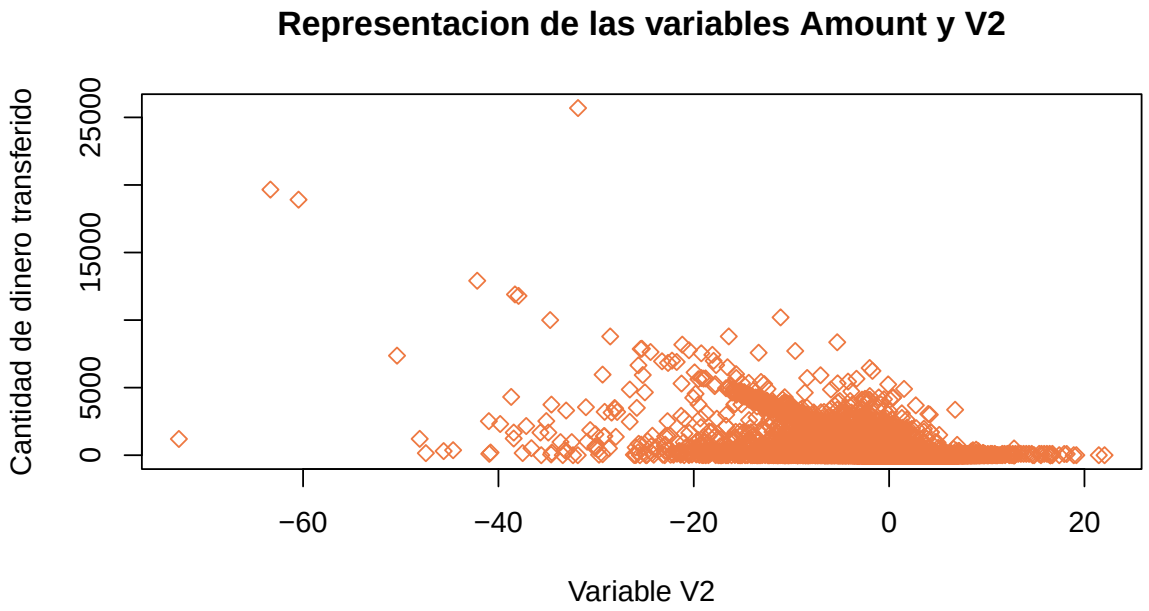
```
table(datos$Class)
```

```
##
##      0      1
## 284315    492
```

```
#Correlacion entre las variables
corr = cor(datos[,-31])
correlaciones = round(corr,4)
corrplot(correlaciones)
```



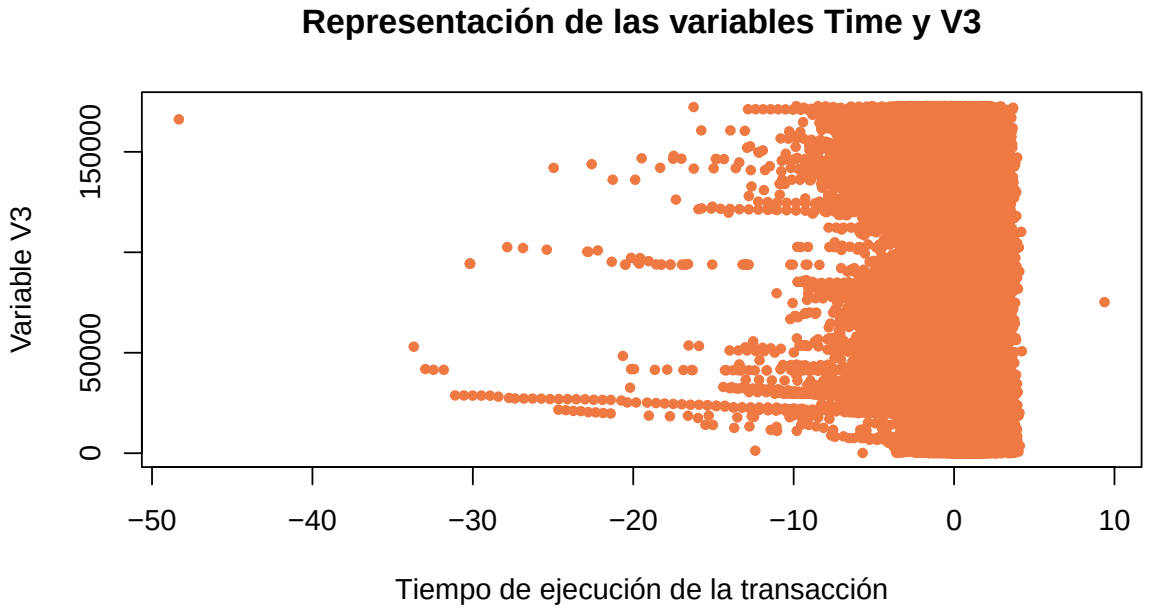

```
plot(datos$V2, datos$Amount, xlab="Variable V2",
     ylab = "Cantidad de dinero transferido", pch=23,col="sienna2",
     main= "Representacion de las variables Amount y V2")
```




```
cor(datos$V2,datos$Amount)
```

```
## [1] -0.5314089
```

```
plot(datos$V3, datos$Time, xlab="Tiempo de ejecución de la transacción",
     ylab = "Variable V3", pch=20,col="sienna2",
     main= "Representación de las variables Time y V3")
```



```r
cor(datos$V3,datos$Time)
```

```
## [1] -0.4196182
```

Partición Entrenamiento/Test

```r
n = nrow(datos)
nent = ceiling(0.7*n)    #n casos de entrenamiento
ntest = n-nent           #n casos de test
indin = 1:n
indient = sort(sample(indin,nent))
inditest = setdiff(indin,indient)
datos_ent = datos[indient,]
datos_test = datos[inditest,]
Y=decodeClassLabels(datos[,31])
ytest=Y[inditest,]
```

Comenzamos a trabajar con el modelo H2O

```r
  #Inicializacion del servicio h2o
localH2O = h2o.init(nthreads = -1)
```

```
##  Connection successful!
##
## R is connected to the H2O cluster:
##     H2O cluster uptime:         35 minutes 56 seconds
##     H2O cluster timezone:       Europe/Madrid
##     H2O data parsing timezone:  UTC
##     H2O cluster version:        3.36.0.4
##     H2O cluster version age:    2 months and 16 days
##     H2O cluster name:           H2O_started_from_R_raque_cki562
##     H2O cluster total nodes:    1
##     H2O cluster total memory:   3.11 GB
##     H2O cluster total cores:    8
##     H2O cluster allowed cores:  8
##     H2O cluster healthy:        TRUE
##     H2O Connection ip:          localhost
##     H2O Connection port:        54321
##     H2O Connection proxy:       NA
##     H2O Internal Security:      FALSE
##     R Version:                  R version 4.2.0 (2022-04-22 ucrt)
```

```r
  #Convertir los data.frames en objetos que puedan ser procesados
train.hex = as.h2o(datos_ent)
```

```
##   |                                                                      |
```

```
test.hex = as.h2o(datos_test)
```

```
##   |                                                                      |
```

A continuación creo y estudio los distintos modelos propuestos:

Modelo 1: Red con dos capas ocultas de 25 nodos cada una

```
modeloc1 = h2o.deeplearning(
  x = 1:30,  #variables predictoras
  y = 31,    #variable respuesta
  training_frame = train.hex,   #datos de entrenamiento
  validation_frame = test.hex,  #datos de validacion/test
  distribution="multinomial",
  activation = 'RectifierWithDropout',
  hidden = c(25,25), #tamaño de las capas ocultas
  hidden_dropout_ratio = c(0.5, 0.5),
  input_dropout_ratio = 0.2,
  epochs = 50, #iteraciones
  l1 = 1e-5,
  l2 = 1e-5,
  rho = 0.99,
  epsilon = 1e-8)
```

```
##   |                                                                      |
```

```
summary(modeloc1)
```

```
## Model Details:
## ==============
##
## H2OBinomialModel: deeplearning
## Model Key:  DeepLearning_model_R_1655376581048_553
## Status of Neuron Layers: predicting Class, 2-class classification, multinomial distribution, CrossEn
##   layer units             type dropout       l1       l2 mean_rate rate_rms
## 1     1    30            Input 20.00 %       NA       NA        NA       NA
## 2     2    25 RectifierDropout 50.00 % 0.000010 0.000010  0.677713 0.456421
## 3     3    25 RectifierDropout 50.00 % 0.000010 0.000010  0.762005 0.413119
## 4     4     2          Softmax      NA 0.000010 0.000010  0.225570 0.396629
##   momentum mean_weight weight_rms mean_bias bias_rms
## 1       NA          NA         NA        NA       NA
## 2 0.000000   -0.004609   0.031345  0.075825 0.260343
## 3 0.000000    0.023798   0.114147 -0.036745 0.588752
## 4 0.000000    0.028275   0.575320  0.002562 3.420491
##
## H2OBinomialMetrics: deeplearning
## ** Reported on training data. **
## ** Metrics reported on temporary training frame with 10037 samples **
##
## MSE:  0.0007235588
```

```
## RMSE:  0.02689905
## LogLoss:  0.007986724
## Mean Per-Class Error:  0.1250998
## AUC:  0.9807311
## AUCPR:  0.678522
## Gini:  0.9614622
##
## Confusion Matrix (vertical: actual; across: predicted) for F1-optimal threshold:
##            0  1    Error      Rate
## 0      10019  2 0.000200  =2/10021
## 1          4 12 0.250000     =4/16
## Totals 10023 14 0.000598  =6/10037
##
## Maximum Metrics: Maximum metrics at their respective thresholds
##                         metric threshold        value idx
## 1                       max f1  0.067138     0.800000   8
## 2                       max f2  0.000624     0.802469  11
## 3                 max f0point5  0.067138     0.833333   8
## 4                 max accuracy  0.067138     0.999402   8
## 5                max precision  0.999995     0.857143   1
## 6                   max recall  0.000012     1.000000 298
## 7              max specificity  1.000000     0.999900   0
## 8             max absolute_mcc  0.067138     0.801490   8
## 9   max min_per_class_accuracy  0.000017     0.917374 225
## 10 max mean_per_class_accuracy  0.000065     0.934456  59
## 11                     max tns  1.000000 10020.000000   0
## 12                     max fns  1.000000    11.000000   0
## 13                     max fps  0.000008 10021.000000 399
## 14                     max tps  0.000012    16.000000 298
## 15                     max tnr  1.000000     0.999900   0
## 16                     max fnr  1.000000     0.687500   0
## 17                     max fpr  0.000008     1.000000 399
## 18                     max tpr  0.000012     1.000000 298
##
## Gains/Lift Table: Extract with `h2o.gainsLift(<model>, <data>)` or `h2o.gainsLift(<model>, valid=<T/
## H2OBinomialMetrics: deeplearning
## ** Reported on validation data. **
## ** Metrics reported on full validation frame **
##
## MSE:  0.0007700126
## RMSE:  0.0277491
## LogLoss:  0.006719206
## Mean Per-Class Error:  0.1227295
## AUC:  0.964825
## AUCPR:  0.7210647
## Gini:  0.9296499
##
## Confusion Matrix (vertical: actual; across: predicted) for F1-optimal threshold:
##            0   1    Error       Rate
## 0      85268  15 0.000176  =15/85283
## 1         39 120 0.245283    =39/159
## Totals 85307 135 0.000632  =54/85442
##
## Maximum Metrics: Maximum metrics at their respective thresholds
```

```
##                         metric threshold        value idx
## 1                       max f1  0.055725     0.816327  84
## 2                       max f2  0.033383     0.780645  88
## 3                 max f0point5  0.055725     0.858369  84
## 4                 max accuracy  0.055725     0.999368  84
## 5                max precision  0.999991     0.924528   2
## 6                   max recall  0.000009     1.000000 390
## 7              max specificity  1.000000     0.999953   0
## 8             max absolute_mcc  0.055725     0.818754  84
## 9   max min_per_class_accuracy  0.000020     0.911950 295
## 10 max mean_per_class_accuracy  0.000020     0.927212 295
## 11                      max tns  1.000000 85279.000000   0
## 12                      max fns  1.000000   116.000000   0
## 13                      max fps  0.000008 85283.000000 399
## 14                      max tps  0.000009   159.000000 390
## 15                      max tnr  1.000000     0.999953   0
## 16                      max fnr  1.000000     0.729560   0
## 17                      max fpr  0.000008     1.000000 399
## 18                      max tpr  0.000009     1.000000 390
##
## Gains/Lift Table: Extract with `h2o.gainsLift(<model>, <data>)` or `h2o.gainsLift(<model>, valid=<T/
##
##
## Scoring History:
##              timestamp   duration training_speed   epochs iterations
## 1  2022-06-16 13:25:41  0.000 sec             NA  0.00000          0
## 2  2022-06-16 13:25:42  0.893 sec 179344 obs/sec  0.50196          1
## 3  2022-06-16 13:25:47  5.948 sec 184298 obs/sec  5.01871         10
## 4  2022-06-16 13:25:52 11.033 sec 183920 obs/sec  9.53528         19
## 5  2022-06-16 13:25:57 16.412 sec 206065 obs/sec 16.06129         32
## 6  2022-06-16 13:26:02 21.600 sec 219340 obs/sec 22.58263         45
## 7  2022-06-16 13:26:07 26.649 sec 228546 obs/sec 29.10750         58
## 8  2022-06-16 13:26:13 31.899 sec 236691 obs/sec 36.12843         72
## 9  2022-06-16 13:26:18 37.304 sec 241475 obs/sec 43.15099         86
## 10 2022-06-16 13:26:23 42.603 sec 243212 obs/sec 49.66977         99
## 11 2022-06-16 13:26:24 43.144 sec 243418 obs/sec 50.17086        100
##            samples training_rmse training_logloss training_r2 training_auc
## 1         0.000000            NA               NA          NA           NA
## 2    100074.000000       0.03993          0.02631    -0.00160      0.69475
## 3   1000555.000000       0.02771          0.00705     0.51762      0.97633
## 4   1901001.000000       0.02737          0.00764     0.52934      0.98905
## 5   3202059.000000       0.02588          0.00882     0.57925      0.97973
## 6   4502187.000000       0.02530          0.00914     0.59767      0.98427
## 7   5803017.000000       0.02551          0.00899     0.59105      0.97913
## 8   7202745.000000       0.02533          0.00961     0.59691      0.98568
## 9   8602797.000000       0.02603          0.00827     0.57429      0.97961
## 10  9902414.000000       0.02633          0.00848     0.56449      0.98516
## 11 10002314.000000       0.02690          0.00799     0.54538      0.98073
##    training_pr_auc training_lift training_classification_error validation_rmse
## 1               NA            NA                           NA              NA
## 2          0.23191      37.26609                      0.00120         0.04314
## 3          0.71929      86.95421                      0.00060         0.02926
## 4          0.67108      86.95421                      0.00060         0.02878
## 5          0.67274      86.95421                      0.00060         0.02611
```

```
## 6          0.65849      86.95421                      0.00060         0.02554
## 7          0.65210      86.95421                      0.00060         0.02584
## 8          0.66318      86.95421                      0.00060         0.02550
## 9          0.67788      86.95421                      0.00060         0.02626
## 10         0.67883      86.95421                      0.00060         0.02683
## 11         0.67852      86.95421                      0.00060         0.02775
##    validation_logloss validation_r2 validation_auc validation_pr_auc
## 1                  NA            NA             NA                NA
## 2             0.03012      -0.00186        0.77101           0.37197
## 3             0.00679       0.53899        0.96788           0.72696
## 4             0.00687       0.55399        0.97052           0.72940
## 5             0.00715       0.63310        0.95852           0.71536
## 6             0.00713       0.64873        0.95515           0.71371
## 7             0.00704       0.64066        0.97483           0.71412
## 8             0.00755       0.65003        0.96996           0.71748
## 9             0.00674       0.62866        0.96773           0.72716
## 10            0.00699       0.61241        0.97041           0.72277
## 11            0.00672       0.58545        0.96482           0.72106
##    validation_lift validation_classification_error
## 1               NA                             NA
## 2         54.05136                        0.00144
## 3         81.07704                        0.00063
## 4         81.07704                        0.00064
## 5         81.07704                        0.00063
## 6         81.07704                        0.00064
## 7         82.33405                        0.00066
## 8         81.70554                        0.00063
## 9         81.70554                        0.00063
## 10        81.70554                        0.00063
## 11        82.33405                        0.00063
##
## Variable Importances: (Extract with `h2o.varimp`)
## =================================================
##
## Variable Importances:
##   variable relative_importance scaled_importance percentage
## 1      V12            1.000000          1.000000   0.106533
## 2     Time            0.785539          0.785539   0.083686
## 3      V14            0.736501          0.736501   0.078462
## 4       V4            0.632106          0.632106   0.067340
## 5      V10            0.631333          0.631333   0.067258
##
## ---
##    variable relative_importance scaled_importance percentage
## 25      V18            0.114896          0.114896   0.012240
## 26      V28            0.110261          0.110261   0.011746
## 27      V22            0.101107          0.101107   0.010771
## 28       V8            0.100631          0.100631   0.010721
## 29       V6            0.090120          0.090120   0.009601
## 30      V13            0.083160          0.083160   0.008859
```

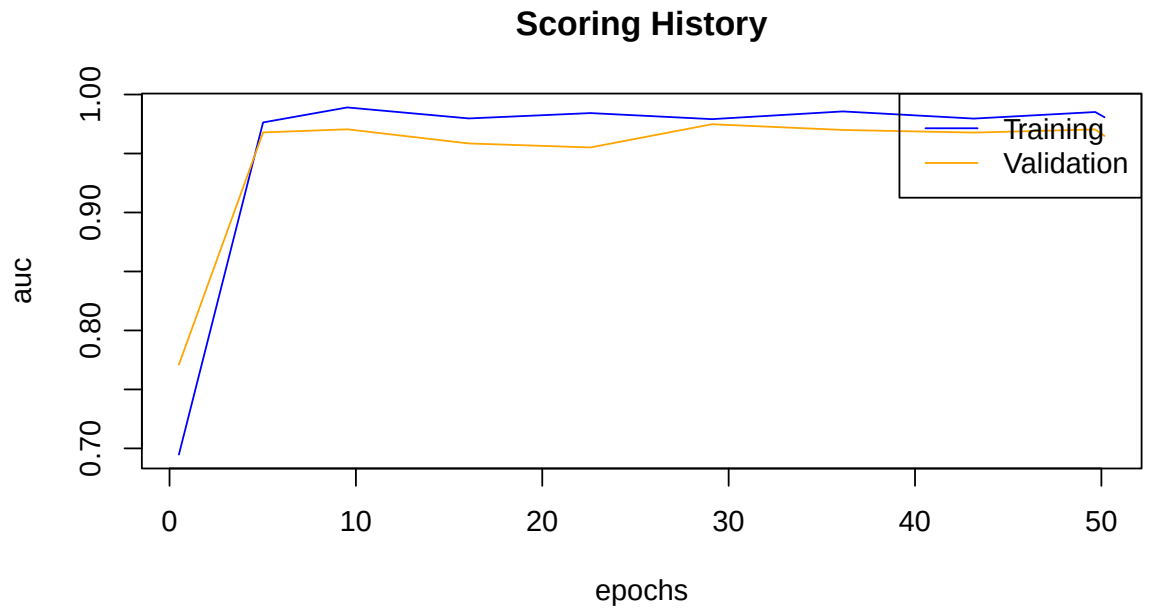
```
#Veamos la capacidad de prediccion del modelo
predict_test = h2o.predict(modeloc1, newdata = test.hex)
```

```
##   |                                                                      |
```

```
pred = as.data.frame(predict_test)

tabla = table(Obs = datos_test[,31], Pred = pred[,1])

aciertos=100*diag(prop.table(tabla,1))
acierto=100*sum(diag(tabla))/sum(tabla)
acierto
```

```
## [1] 99.9368
```

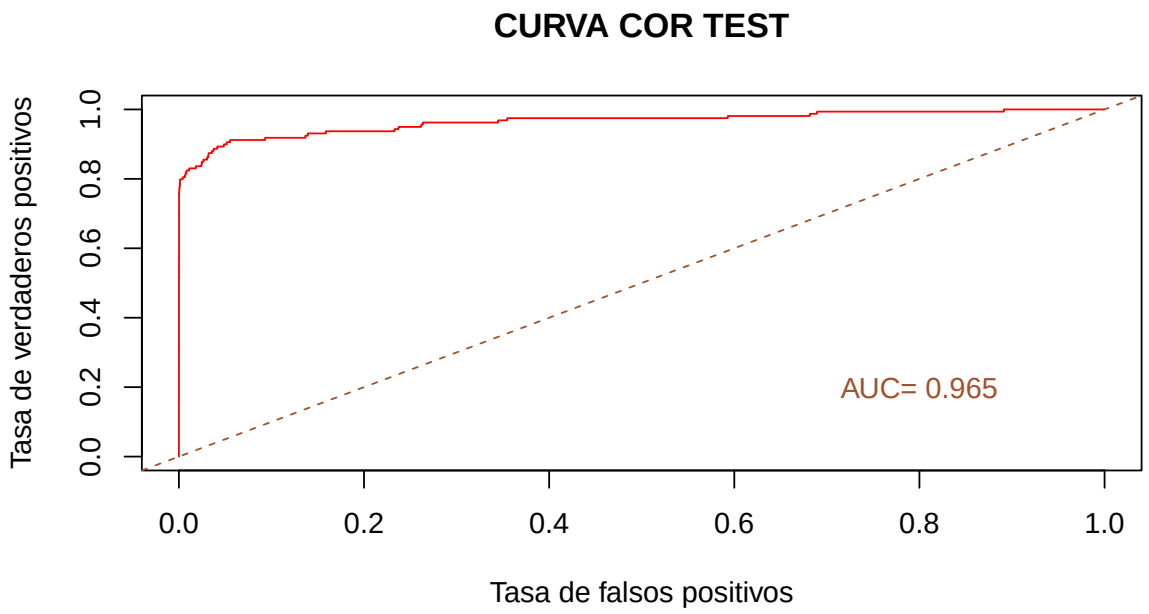
```
#Curva ROC
  #ROC para los conjuntos de entrenamiento y test
plot(modeloc1, metric="auc")
```



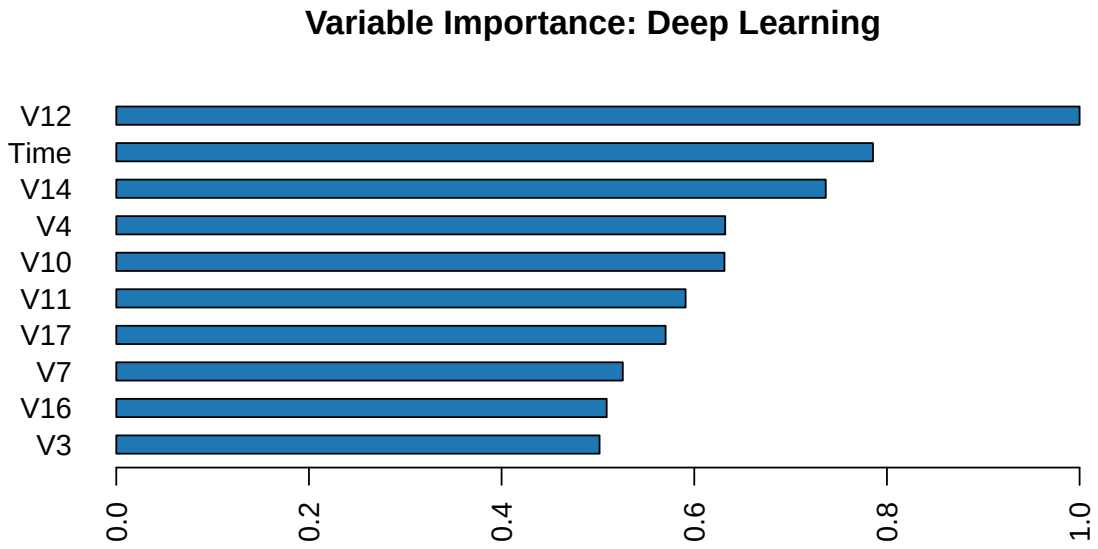

```
  #Curva ROC y AUC
prediobj = prediction(pred[,3], ytest[,2])
plot(performance(prediobj, "tpr","fpr"),main="CURVA COR TEST",
     xlab="Tasa de falsos positivos",
     ylab="Tasa de verdaderos positivos", col="red")
abline(a=0,b=1,col="sienna",lty=2)
auc = as.numeric(performance(prediobj,"auc")@y.values)
cat("Area bajo la curva COR Test= ",auc,"\n")
```

```
## Area bajo la curva COR Test=  0.9648469
```

```
text(0.8,0.2,paste("AUC=",round(auc,3)),col="sienna" )
```




```
#Importancia de las variables
h2o.varimp_plot(modeloc1)
```



Modelo 2: Red con dos capas ocultas de 10 nodos cada una

```
modeloc2 = h2o.deeplearning(
  x = 1:30,  #variables predictoras
  y = 31,    #variable respuesta
  training_frame = train.hex,   #datos de entrenamiento
  validation_frame = test.hex,  #datos de validacion/test
  distribution="multinomial",
  activation = 'RectifierWithDropout',
  hidden = c(10,10), #tamaño de las capas ocultas
  hidden_dropout_ratio = c(0.5, 0.5),
  input_dropout_ratio = 0.2,
  epochs = 50, #iteraciones
  l1 = 1e-5,
  l2 = 1e-5,
  rho = 0.99,
  epsilon = 1e-8)
```

```
##   |                                                                              |
```

```
summary(modeloc2)
```

```
## Model Details:
## ==============
##
## H2OBinomialModel: deeplearning
## Model Key:  DeepLearning_model_R_1655376581048_574
## Status of Neuron Layers: predicting Class, 2-class classification, multinomial distribution, CrossEn
##   layer units             type dropout       l1       l2 mean_rate rate_rms
## 1     1    30            Input 20.00 %       NA       NA        NA       NA
## 2     2    10 RectifierDropout 50.00 % 0.000010 0.000010  0.053851 0.175343
## 3     3    10 RectifierDropout 50.00 % 0.000010 0.000010  0.057400 0.165545
## 4     4     2          Softmax      NA 0.000010 0.000010  0.015647 0.032247
##   momentum mean_weight weight_rms mean_bias bias_rms
## 1       NA          NA         NA        NA       NA
## 2 0.000000   -0.008410   0.051935  0.513355 0.520243
## 3 0.000000    0.020206   0.267734  0.690256 0.440091
## 4 0.000000    0.023101   1.088132 -0.601283 1.981450
##
## H2OBinomialMetrics: deeplearning
## ** Reported on training data. **
## ** Metrics reported on temporary training frame with 9965 samples **
##
## MSE:  0.0004456277
## RMSE:  0.0211099
## LogLoss:  0.006295865
## Mean Per-Class Error:  0.1001005
## AUC:  0.9374586
## AUCPR:  0.5798407
## Gini:  0.8749171
##
## Confusion Matrix (vertical: actual; across: predicted) for F1-optimal threshold:
```

```
##           0  1    Error     Rate
## 0      9953  2 0.000201  =2/9955
## 1         2  8 0.200000    =2/10
## Totals 9955 10 0.000401  =4/9965
##
## Maximum Metrics: Maximum metrics at their respective thresholds
##                         metric threshold       value idx
## 1                       max f1  0.972065    0.800000   6
## 2                       max f2  0.026740    0.833333  10
## 3                 max f0point5  0.972065    0.800000   6
## 4                 max accuracy  0.972065    0.999599   6
## 5                max precision  0.972065    0.800000   6
## 6                   max recall  0.000138    1.000000 336
## 7              max specificity  1.000000    0.999799   0
## 8             max absolute_mcc  0.972065    0.799799   6
## 9   max min_per_class_accuracy  0.026740    0.900000  10
## 10 max mean_per_class_accuracy  0.026740    0.949749  10
## 11                      max tns  1.000000 9953.000000   0
## 12                      max fns  1.000000    8.000000   0
## 13                      max fps  0.000085 9955.000000 399
## 14                      max tps  0.000138   10.000000 336
## 15                      max tnr  1.000000    0.999799   0
## 16                      max fnr  1.000000    0.800000   0
## 17                      max fpr  0.000085    1.000000 399
## 18                      max tpr  0.000138    1.000000 336
##
## Gains/Lift Table: Extract with `h2o.gainsLift(<model>, <data>)` or `h2o.gainsLift(<model>, valid=<T/
## H2OBinomialMetrics: deeplearning
## ** Reported on validation data. **
## ** Metrics reported on full validation frame **
##
## MSE:  0.0006573484
## RMSE:  0.02563881
## LogLoss:  0.00557946
## Mean Per-Class Error:  0.1195848
## AUC:  0.9466821
## AUCPR:  0.713189
## Gini:  0.8933641
##
## Confusion Matrix (vertical: actual; across: predicted) for F1-optimal threshold:
##            0   1    Error       Rate
## 0      85268  15 0.000176  =15/85283
## 1         38 121 0.238994    =38/159
## Totals 85306 136 0.000620  =53/85442
##
## Maximum Metrics: Maximum metrics at their respective thresholds
##                         metric threshold        value idx
## 1                       max f1  0.252883     0.820339  69
## 2                       max f2  0.252883     0.783679  69
## 3                 max f0point5  0.252883     0.860597  69
## 4                 max accuracy  0.252883     0.999380  69
## 5                max precision  0.997954     0.920455  21
## 6                   max recall  0.000118     1.000000 385
## 7              max specificity  0.999999     0.999930   0
```

```
## 8             max absolute_mcc  0.252883     0.822543  69
## 9   max min_per_class_accuracy  0.000230     0.893082 285
## 10 max mean_per_class_accuracy  0.000310     0.915328 254
## 11                     max tns  0.999999 85277.000000   0
## 12                     max fns  0.999999   109.000000   0
## 13                     max fps  0.000076 85283.000000 399
## 14                     max tps  0.000118   159.000000 385
## 15                     max tnr  0.999999     0.999930   0
## 16                     max fnr  0.999999     0.685535   0
## 17                     max fpr  0.000076     1.000000 399
## 18                     max tpr  0.000118     1.000000 385
##
## Gains/Lift Table: Extract with `h2o.gainsLift(<model>, <data>)` or `h2o.gainsLift(<model>, valid=<T/
##
##
## Scoring History:
##             timestamp   duration training_speed   epochs iterations
## 1 2022-06-16 13:26:27  0.000 sec             NA  0.00000          0
## 2 2022-06-16 13:26:28  0.540 sec 352070 obs/sec  0.49977          1
## 3 2022-06-16 13:26:33  5.600 sec 402726 obs/sec 10.52848         21
## 4 2022-06-16 13:26:38 10.716 sec 421857 obs/sec 21.55577         43
## 5 2022-06-16 13:26:43 15.901 sec 426769 obs/sec 32.59559         65
## 6 2022-06-16 13:26:48 21.017 sec 430652 obs/sec 43.62801         87
## 7 2022-06-16 13:26:51 24.010 sec 434094 obs/sec 50.14515        100
## 8 2022-06-16 13:26:51 24.178 sec 433999 obs/sec 50.14515        100
##          samples training_rmse training_logloss training_r2 training_auc
## 1       0.000000            NA               NA          NA           NA
## 2   99636.000000       0.03168          0.01138    -0.00097      0.95044
## 3 2099011.000000       0.02111          0.00630     0.55549      0.93746
## 4 4297466.000000       0.02036          0.00691     0.58643      0.92320
## 5 6498419.000000       0.02010          0.00706     0.59706      0.97825
## 6 8697898.000000       0.02015          0.00739     0.59489      0.98192
## 7 9997188.000000       0.02006          0.00724     0.59859      0.98209
## 8 9997188.000000       0.02111          0.00630     0.55549      0.93746
##   training_pr_auc training_lift training_classification_error validation_rmse
## 1              NA            NA                           NA              NA
## 2         0.28081      69.75500                      0.00080         0.04314
## 3         0.57984      89.68500                      0.00040         0.02564
## 4         0.57981      89.68500                      0.00040         0.02592
## 5         0.58517      89.68500                      0.00040         0.02689
## 6         0.58525      89.68500                      0.00040         0.02622
## 7         0.58526      89.68500                      0.00040         0.02720
## 8         0.57984      89.68500                      0.00040         0.02564
##   validation_logloss validation_r2 validation_auc validation_pr_auc
## 1                 NA            NA             NA                NA
## 2            0.02230      -0.00184        0.86270           0.10441
## 3            0.00558       0.64610        0.94668           0.71319
## 4            0.00617       0.63831        0.94925           0.72222
## 5            0.00675       0.61065        0.96664           0.72284
## 6            0.00666       0.62988        0.96846           0.72470
## 7            0.00705       0.60155        0.96794           0.72552
## 8            0.00558       0.64610        0.94668           0.71319
##   validation_lift validation_classification_error
## 1              NA                             NA
```

```
## 2        44.62380                          0.00393
## 3        81.07704                          0.00062
## 4        81.07704                          0.00063
## 5        81.70554                          0.00063
## 6        81.70554                          0.00064
## 7        81.70554                          0.00063
## 8        81.07704                          0.00062
##
## Variable Importances: (Extract with `h2o.varimp`)
## =================================================
##
## Variable Importances:
##   variable relative_importance scaled_importance percentage
## 1     Time            1.000000          1.000000   0.135420
## 2      V12            0.575190          0.575190   0.077892
## 3      V11            0.534217          0.534217   0.072344
## 4       V3            0.424669          0.424669   0.057509
## 5      V14            0.420427          0.420427   0.056934
##
## ---
##    variable relative_importance scaled_importance percentage
## 25      V28            0.090843          0.090843   0.012302
## 26      V20            0.081206          0.081206   0.010997
## 27       V8            0.080306          0.080306   0.010875
## 28       V1            0.075630          0.075630   0.010242
## 29      V19            0.069034          0.069034   0.009349
## 30      V23            0.034290          0.034290   0.004644
```

```
#Veamos la capacidad de prediccion del modelo
predict_test2 = h2o.predict(modeloc2, newdata = test.hex)
```

```
##   |                                                                              |
```

```
pred2 = as.data.frame(predict_test2)
head(pred2)
```

```
##   predict        p0           p1
## 1       0 0.9998953 0.0001046992
## 2       0 0.9998932 0.0001067669
## 3       0 0.9998977 0.0001022923
## 4       0 0.9998963 0.0001037319
## 5       0 0.9998911 0.0001089203
## 6       0 0.9998685 0.0001314905
```

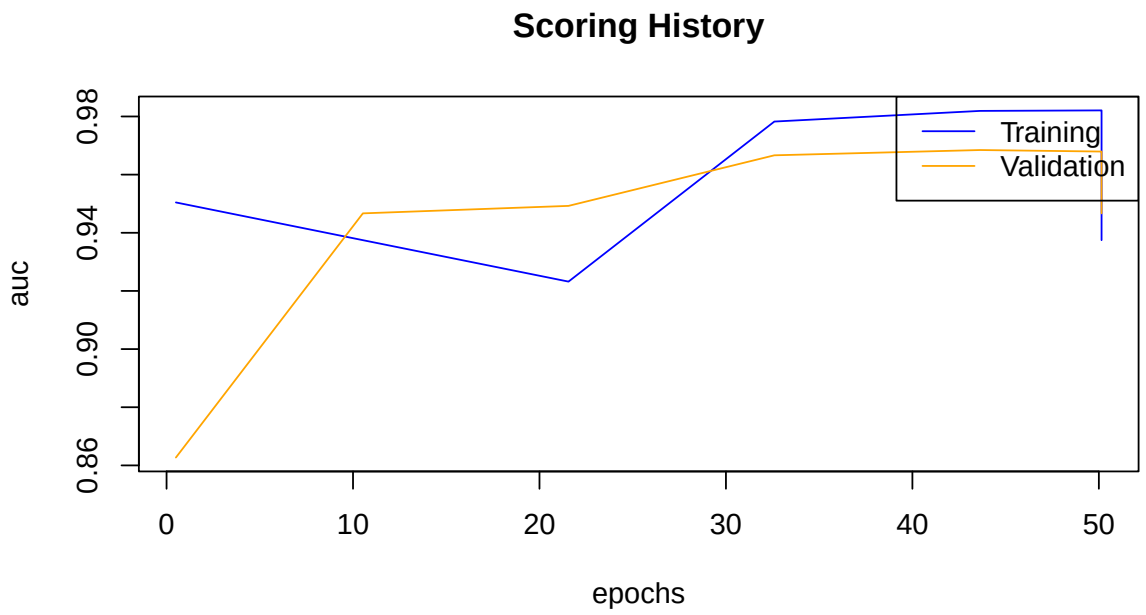
```
tabla2 = table(Obs = datos_test[,31], Pred = pred2[,1])
tabla2
```

```
##    Pred
## Obs     0     1
##   0 85268    15
##   1    38   121
```

```
aciertos2 = 100*diag(prop.table(tabla2,1))
acierto2 = 100*sum(diag(tabla2))/sum(tabla2)
acierto2
```

```
## [1] 99.93797
```

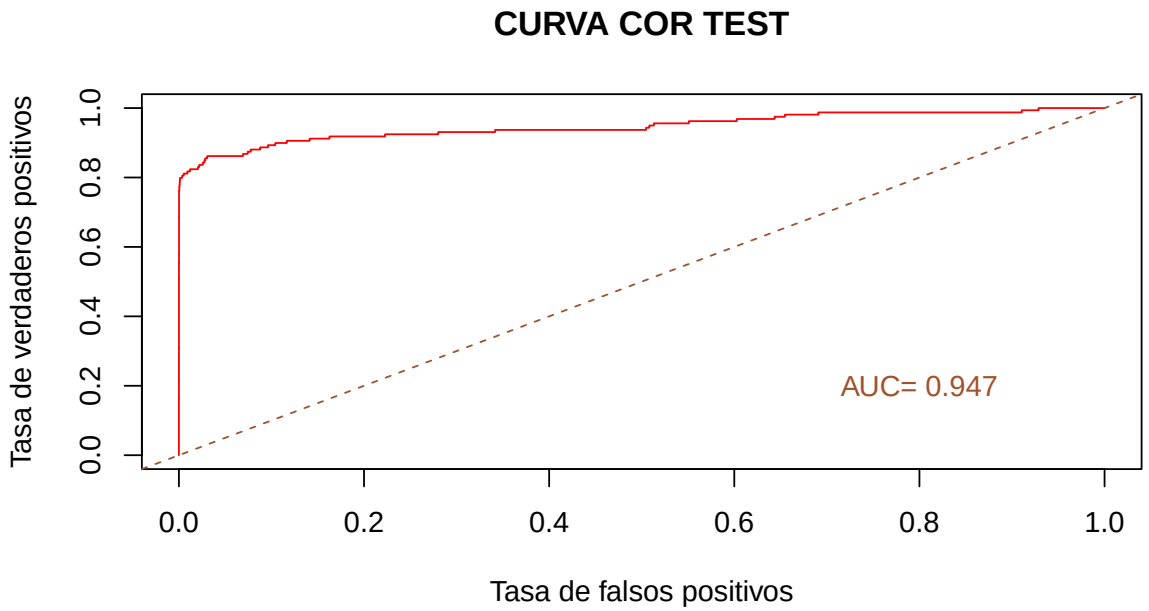
```
#Curva ROC
  #ROC para los conjuntos de entrenamiento y test
plot(modeloc2, metric="auc")
```



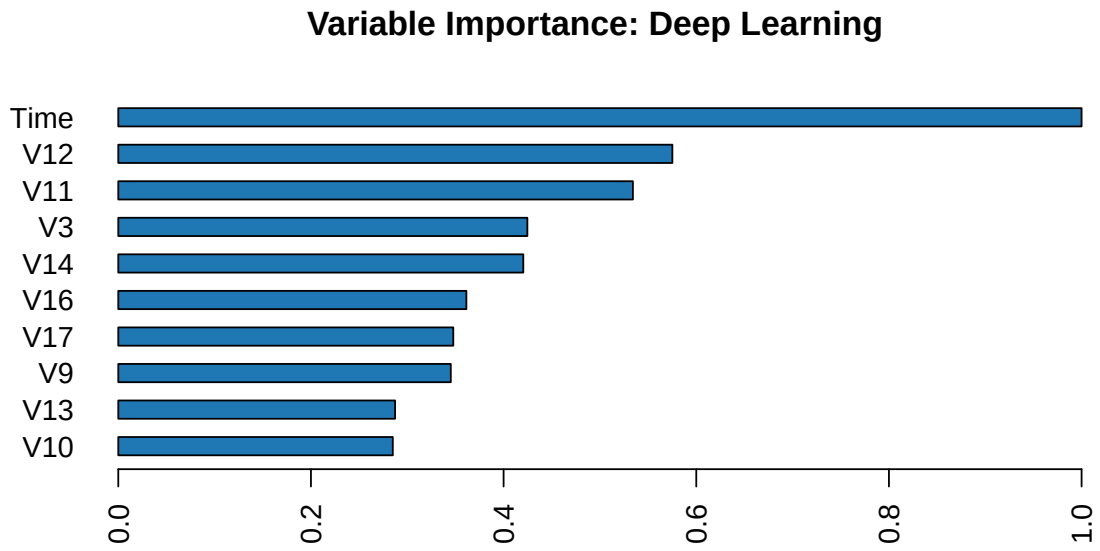

```
  #Curva ROC y AUC
prediobj2 = prediction(pred2[,3], ytest[,2])
plot(performance(prediobj2, "tpr","fpr"),main="CURVA COR TEST",
     xlab="Tasa de falsos positivos",
     ylab="Tasa de verdaderos positivos", col="red")
abline(a=0,b=1,col="sienna",lty=2)
auc2 = as.numeric(performance(prediobj2,"auc")@y.values)
cat("Area bajo la curva COR Test= ",auc2,"\n")
```

```
## Area bajo la curva COR Test=  0.9466591
```

```
text(0.8,0.2,paste("AUC=",round(auc2,3)),col="sienna")
```




```
#Importancia de las variables
h2o.varimp_plot(modeloc2)
```



Modelo 3: Red con una capa oculta de 25 nodos

```
modeloc3 = h2o.deeplearning(
  x = 1:30,  #variables predictoras
  y = 31,    #variable respuesta
  training_frame = train.hex,   #datos de entrenamiento
  validation_frame = test.hex,  #datos de validacion/test
  distribution="multinomial",
  activation = 'RectifierWithDropout',
  hidden = 25, #tamaño de las capas ocultas
  hidden_dropout_ratio = 0.5,
  input_dropout_ratio = 0.2,
  epochs = 50, #iteraciones
  l1 = 1e-5,
  l2 = 1e-5,
  rho = 0.99,
  epsilon = 1e-8)
```

```
##   |                                                                              |
```

```
summary(modeloc3)
```

```
## Model Details:
## ==============
##
## H2OBinomialModel: deeplearning
## Model Key:  DeepLearning_model_R_1655376581048_589
## Status of Neuron Layers: predicting Class, 2-class classification, multinomial distribution, CrossEn
##   layer units             type dropout       l1       l2 mean_rate rate_rms
## 1     1    30            Input 20.00 %       NA       NA        NA       NA
## 2     2    25 RectifierDropout 50.00 % 0.000010 0.000010  0.169344 0.378191
## 3     3     2          Softmax      NA 0.000010 0.000010  0.165322 0.376624
##   momentum mean_weight weight_rms mean_bias bias_rms
## 1       NA          NA         NA        NA       NA
## 2 0.000000   -0.004906   0.042788  0.538920 0.460601
## 3 0.000000   -0.063186   0.585461  0.001303 2.315043
##
## H2OBinomialMetrics: deeplearning
## ** Reported on training data. **
## ** Metrics reported on temporary training frame with 9950 samples **
##
## MSE:  0.0002464956
## RMSE:  0.01570018
## LogLoss:  0.004640048
## Mean Per-Class Error:  0.0294621
## AUC:  0.9916855
## AUCPR:  0.8577107
## Gini:  0.9833709
##
## Confusion Matrix (vertical: actual; across: predicted) for F1-optimal threshold:
##           0  1    Error     Rate
## 0      9932  1 0.000101  =1/9933
```

```
## 1         1 16 0.058824    =1/17
## Totals 9933 17 0.000201  =2/9950
##
## Maximum Metrics: Maximum metrics at their respective thresholds
##                         metric threshold       value idx
## 1                        max f1  0.499768    0.941176   7
## 2                        max f2  0.499768    0.941176   7
## 3                  max f0point5  0.499768    0.941176   7
## 4                  max accuracy  0.499768    0.999799   7
## 5                 max precision  0.499768    0.941176   7
## 6                    max recall  0.000008    1.000000 242
## 7               max specificity  1.000000    0.999899   0
## 8              max absolute_mcc  0.499768    0.941076   7
## 9   max min_per_class_accuracy  0.499768    0.941176   7
## 10 max mean_per_class_accuracy  0.499768    0.970538   7
## 11                       max tns  1.000000 9932.000000   0
## 12                       max fns  1.000000    8.000000   0
## 13                       max fps  0.000003 9933.000000 399
## 14                       max tps  0.000008   17.000000 242
## 15                       max tnr  1.000000    0.999899   0
## 16                       max fnr  1.000000    0.470588   0
## 17                       max fpr  0.000003    1.000000 399
## 18                       max tpr  0.000008    1.000000 242
##
## Gains/Lift Table: Extract with `h2o.gainsLift(<model>, <data>)` or `h2o.gainsLift(<model>, valid=<T/
## H2OBinomialMetrics: deeplearning
## ** Reported on validation data. **
## ** Metrics reported on full validation frame **
##
## MSE:  0.0006984428
## RMSE:  0.02642807
## LogLoss:  0.007176472
## Mean Per-Class Error:  0.1227412
## AUC:  0.9695872
## AUCPR:  0.7157677
## Gini:  0.9391743
##
## Confusion Matrix (vertical: actual; across: predicted) for F1-optimal threshold:
##            0   1    Error       Rate
## 0      85266  17 0.000199  =17/85283
## 1         39 120 0.245283    =39/159
## Totals 85305 137 0.000655  =56/85442
##
## Maximum Metrics: Maximum metrics at their respective thresholds
##                         metric threshold        value idx
## 1                        max f1  0.057142     0.810811  77
## 2                        max f2  0.046933     0.779639  80
## 3                  max f0point5  0.184638     0.856515  71
## 4                  max accuracy  0.184638     0.999345  71
## 5                 max precision  0.996823     0.922222  30
## 6                    max recall  0.000005     1.000000 373
## 7               max specificity  1.000000     0.999930   0
## 8              max absolute_mcc  0.057142     0.812740  77
## 9   max min_per_class_accuracy  0.000010     0.924528 300
```

```
## 10 max mean_per_class_accuracy  0.000014     0.927637 273
## 11                     max tns  1.000000 85277.000000   0
## 12                     max fns  1.000000   112.000000   0
## 13                     max fps  0.000002 85283.000000 399
## 14                     max tps  0.000005   159.000000 373
## 15                     max tnr  1.000000     0.999930   0
## 16                     max fnr  1.000000     0.704403   0
## 17                     max fpr  0.000002     1.000000 399
## 18                     max tpr  0.000005     1.000000 373
##
## Gains/Lift Table: Extract with `h2o.gainsLift(<model>, <data>)` or `h2o.gainsLift(<model>, valid=<T/
##
##
## Scoring History:
##              timestamp   duration training_speed   epochs iterations
## 1  2022-06-16 13:26:55  0.000 sec             NA  0.00000          0
## 2  2022-06-16 13:26:56  0.672 sec 234774 obs/sec  0.50048          1
## 3  2022-06-16 13:27:01  5.992 sec 232827 obs/sec  6.51423         13
## 4  2022-06-16 13:27:06 11.383 sec 231530 obs/sec 12.53784         25
## 5  2022-06-16 13:27:11 16.571 sec 233708 obs/sec 18.55226         37
## 6  2022-06-16 13:27:17 21.911 sec 238215 obs/sec 25.08153         50
## 7  2022-06-16 13:27:22 27.088 sec 242027 obs/sec 31.60386         63
## 8  2022-06-16 13:27:27 32.441 sec 243237 obs/sec 38.12698         76
## 9  2022-06-16 13:27:32 37.489 sec 246162 obs/sec 44.64291         89
## 10 2022-06-16 13:27:37 41.892 sec 247405 obs/sec 50.16112        100
## 11 2022-06-16 13:27:37 42.039 sec 247399 obs/sec 50.16112        100
##            samples training_rmse training_logloss training_r2 training_auc
## 1         0.000000            NA               NA          NA           NA
## 2     99779.000000       0.03766          0.00961     0.16852      0.94957
## 3   1298710.000000       0.01570          0.00464     0.85548      0.99169
## 4   2499607.000000       0.01473          0.00473     0.87280      0.99343
## 5   3698672.000000       0.01435          0.00469     0.87921      0.99290
## 6   5000380.000000       0.01433          0.00472     0.87967      0.98594
## 7   6300703.000000       0.01425          0.00468     0.88087      0.99380
## 8   7601186.000000       0.01422          0.00469     0.88144      0.99068
## 9   8900233.000000       0.01461          0.00472     0.87483      0.99236
## 10 10000371.000000       0.01424          0.00470     0.88108      0.98582
## 11 10000371.000000       0.01570          0.00464     0.85548      0.99169
##    training_pr_auc training_lift training_classification_error validation_rmse
## 1               NA            NA                           NA              NA
## 2          0.86529      93.64706                      0.00030         0.04158
## 3          0.85771      93.64706                      0.00020         0.02643
## 4          0.86296      93.64706                      0.00020         0.02588
## 5          0.86290      93.64706                      0.00020         0.02554
## 6          0.86250      93.64706                      0.00020         0.02542
## 7          0.86302      93.64706                      0.00020         0.02536
## 8          0.86271      93.64706                      0.00020         0.02532
## 9          0.86284      93.64706                      0.00020         0.02578
## 10         0.86250      93.64706                      0.00020         0.02538
## 11         0.85771      93.64706                      0.00020         0.02643
##    validation_logloss validation_r2 validation_auc validation_pr_auc
## 1                  NA            NA             NA                NA
## 2             0.01724       0.06933        0.84087           0.59169
## 3             0.00718       0.62398        0.96959           0.71577
```

```
## 4             0.00741       0.63939        0.96630           0.71391
## 5             0.00736       0.64889        0.95797           0.71446
## 6             0.00742       0.65199        0.96945           0.71853
## 7             0.00737       0.65369        0.96337           0.72094
## 8             0.00740       0.65491        0.97449           0.72325
## 9             0.00720       0.64222        0.95700           0.71847
## 10            0.00737       0.65307        0.95490           0.72333
## 11            0.00718       0.62398        0.96959           0.71577
##    validation_lift validation_classification_error
## 1               NA                              NA
## 2         71.64948                         0.00096
## 3         80.44853                         0.00066
## 4         81.70554                         0.00066
## 5         81.70554                         0.00064
## 6         82.96255                         0.00066
## 7         82.33405                         0.00064
## 8         82.96255                         0.00064
## 9         82.33405                         0.00064
## 10        82.96255                         0.00064
## 11        80.44853                         0.00066
##
## Variable Importances: (Extract with `h2o.varimp`)
## =================================================
##
## Variable Importances:
##   variable relative_importance scaled_importance percentage
## 1      V14            1.000000          1.000000   0.127356
## 2      V12            0.697325          0.697325   0.088808
## 3       V4            0.606611          0.606611   0.077255
## 4       V9            0.585193          0.585193   0.074528
## 5      V16            0.526602          0.526602   0.067066
##
## ---
##    variable relative_importance scaled_importance percentage
## 25      V25            0.084651          0.084651   0.010781
## 26       V1            0.064749          0.064749   0.008246
## 27      V23            0.063781          0.063781   0.008123
## 28       V8            0.055933          0.055933   0.007123
## 29   Amount            0.052236          0.052236   0.006653
## 30      V24            0.052159          0.052159   0.006643
```

```
#Veamos la capacidad de prediccion del modelo
predict_test3 = h2o.predict(modeloc3, newdata = test.hex)
```

```
##   |                                                                              |
```

```
pred3 = as.data.frame(predict_test3)
head(pred3)
```

```
##   predict        p0           p1
## 1       0 0.9999930 7.046596e-06
## 2       0 0.9999938 6.200740e-06
## 3       0 0.9999937 6.265694e-06
```

```
## 4       0 0.9999946 5.393018e-06
## 5       0 0.9999922 7.797358e-06
## 6       0 0.9999915 8.468504e-06
```

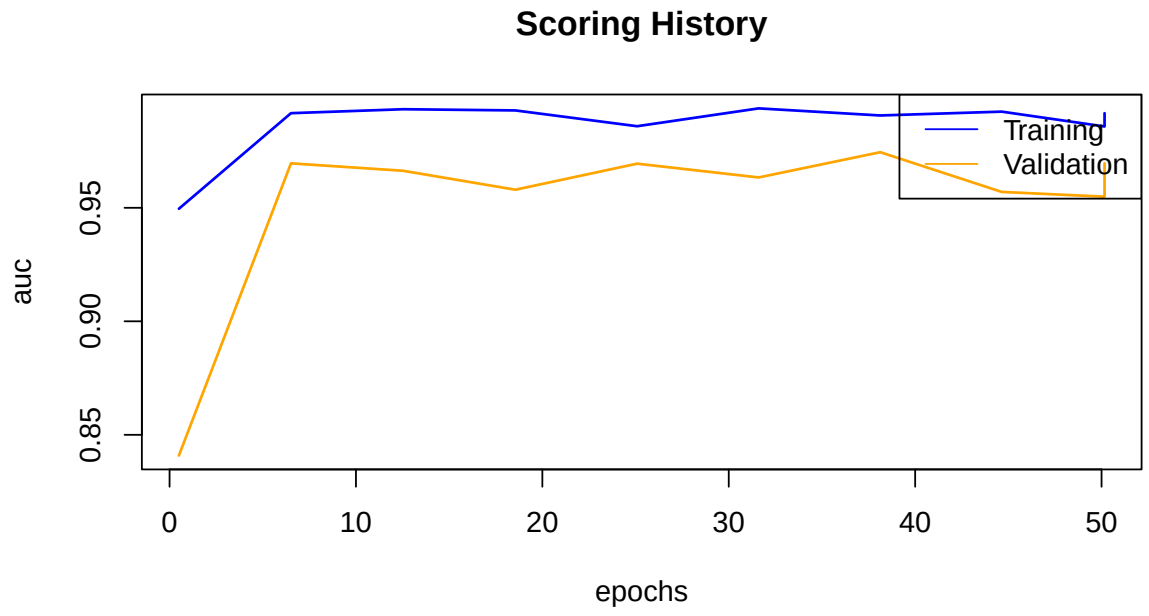
```
tabla3 = table(Obs = datos_test[,31], Pred = pred3[,1])
tabla3
```

```
##    Pred
## Obs     0     1
##   0 85266    17
##   1    39   120
```

```
aciertos3 = 100*diag(prop.table(tabla3,1))
acierto3 = 100*sum(diag(tabla3))/sum(tabla3)
acierto3
```

```
## [1] 99.93446
```

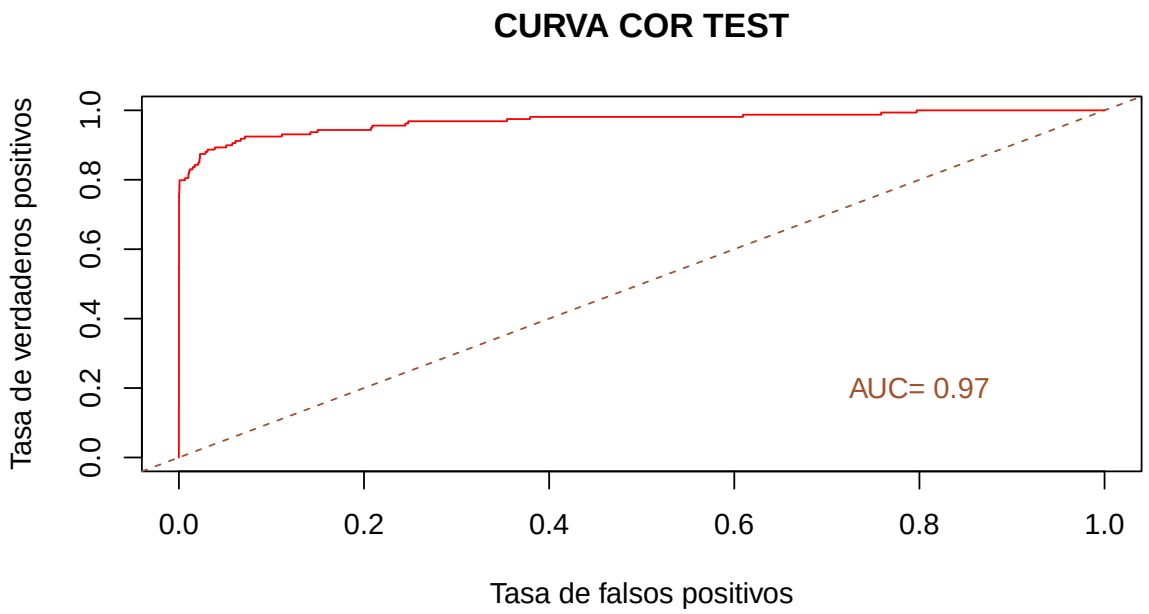
```
#Curva ROC
  #ROC para los conjuntos de entrenamiento y test
plot(modeloc3, metric="auc", lwd=1.5)
```



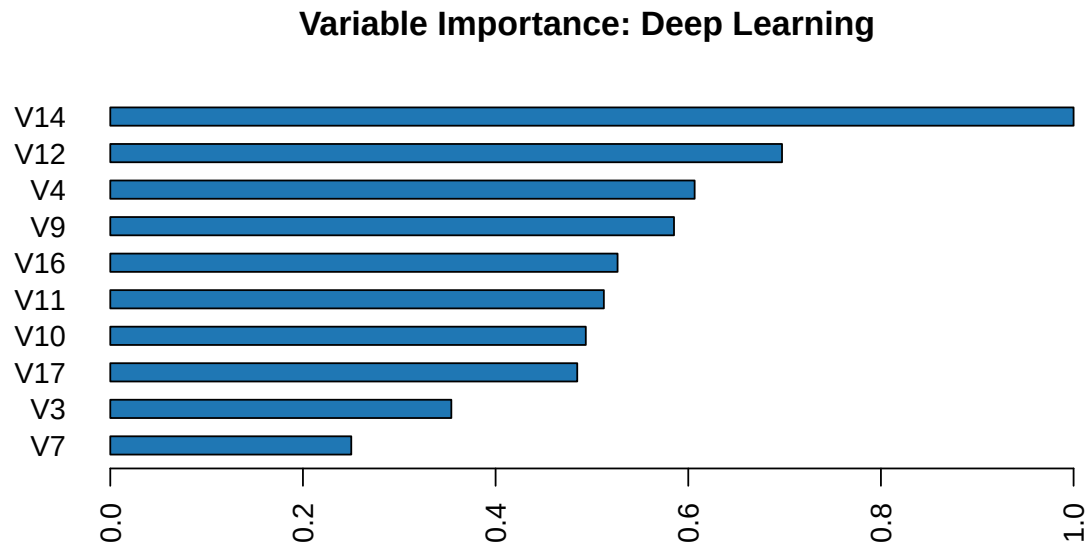

```
  #Curva ROC y AUC
prediobj3 = prediction(pred3[,3], ytest[,2])
plot(performance(prediobj3, "tpr","fpr"),main="CURVA COR TEST",
     xlab="Tasa de falsos positivos",
     ylab="Tasa de verdaderos positivos", col="red")
abline(a=0,b=1,col="sienna",lty=2)
auc3 = as.numeric(performance(prediobj3,"auc")@y.values)
cat("Area bajo la curva COR Test= ",auc3,"\n")
```

```
## Area bajo la curva COR Test=  0.9696655
```

```
text(0.8,0.2,paste("AUC=",round(auc3,3)),col="sienna" )
```




```
#Importancia de las variables
h2o.varimp_plot(modeloc3)
```



Modelo 4: Red con una capa oculta de 5 nodos

```
modeloc4 = h2o.deeplearning(
  x = 1:30,  #variables predictoras
  y = 31,    #variable respuesta
  training_frame = train.hex,   #datos de entrenamiento
  validation_frame = test.hex,  #datos de validacion/test
  distribution="multinomial",
  activation = 'RectifierWithDropout',
  hidden = 5, #tamaño de las capas ocultas
  hidden_dropout_ratio = 0.5,
  input_dropout_ratio = 0.2,
  epochs = 50, #iteraciones
  l1 = 1e-5,
  l2 = 1e-5,
  rho = 0.99,
  epsilon = 1e-8)
```

```
##   |                                                                              |
```

```
summary(modeloc4)
```

```
## Model Details:
## ==============
##
## H2OBinomialModel: deeplearning
## Model Key:  DeepLearning_model_R_1655376581048_610
## Status of Neuron Layers: predicting Class, 2-class classification, multinomial distribution, CrossEn
##   layer units             type dropout       l1       l2 mean_rate rate_rms
## 1     1    30            Input 20.00 %       NA       NA        NA       NA
## 2     2     5 RectifierDropout 50.00 % 0.000010 0.000010  0.004299 0.011634
## 3     3     2          Softmax      NA 0.000010 0.000010  0.004902 0.005841
##   momentum mean_weight weight_rms mean_bias bias_rms
## 1       NA          NA         NA        NA       NA
## 2 0.000000   -0.021915   0.101813  0.350376 0.895083
## 3 0.000000    0.629079   1.581670  0.019607 3.574263
##
## H2OBinomialMetrics: deeplearning
## ** Reported on training data. **
## ** Metrics reported on temporary training frame with 9910 samples **
##
## MSE:  0.0004828754
## RMSE:  0.02197442
## LogLoss:  0.006334879
## Mean Per-Class Error:  0.06260107
## AUC:  0.9901898
## AUCPR:  0.6994399
## Gini:  0.9803795
##
## Confusion Matrix (vertical: actual; across: predicted) for F1-optimal threshold:
##           0  1    Error     Rate
## 0      9892  2 0.000202  =2/9894
```

```
## 1         2 14 0.125000    =2/16
## Totals 9894 16 0.000404  =4/9910
##
## Maximum Metrics: Maximum metrics at their respective thresholds
##                         metric threshold       value idx
## 1                        max f1  0.425582    0.875000  10
## 2                        max f2  0.000842    0.882353  15
## 3                  max f0point5  0.425582    0.875000  10
## 4                  max accuracy  0.425582    0.999596  10
## 5                 max precision  0.425582    0.875000  10
## 6                    max recall  0.000050    1.000000 306
## 7               max specificity  1.000000    0.999798   0
## 8              max absolute_mcc  0.425582    0.874798  10
## 9   max min_per_class_accuracy  0.000842    0.937500  15
## 10 max mean_per_class_accuracy  0.000842    0.968447  15
## 11                      max tns  1.000000 9892.000000   0
## 12                      max fns  1.000000   13.000000   0
## 13                      max fps  0.000039 9894.000000 399
## 14                      max tps  0.000050   16.000000 306
## 15                      max tnr  1.000000    0.999798   0
## 16                      max fnr  1.000000    0.812500   0
## 17                      max fpr  0.000039    1.000000 399
## 18                      max tpr  0.000050    1.000000 306
##
## Gains/Lift Table: Extract with `h2o.gainsLift(<model>, <data>)` or `h2o.gainsLift(<model>, valid=<T/
## H2OBinomialMetrics: deeplearning
## ** Reported on validation data. **
## ** Metrics reported on full validation frame **
##
## MSE:  0.0006627211
## RMSE:  0.02574337
## LogLoss:  0.006155784
## Mean Per-Class Error:  0.1227353
## AUC:  0.9643099
## AUCPR:  0.7199693
## Gini:  0.9286197
##
## Confusion Matrix (vertical: actual; across: predicted) for F1-optimal threshold:
##            0   1    Error       Rate
## 0      85267  16 0.000188  =16/85283
## 1         39 120 0.245283    =39/159
## Totals 85306 136 0.000644  =55/85442
##
## Maximum Metrics: Maximum metrics at their respective thresholds
##                         metric threshold        value idx
## 1                        max f1  0.181633     0.813559  66
## 2                        max f2  0.109870     0.780645  69
## 3                  max f0point5  0.317426     0.853835  63
## 4                  max accuracy  0.181633     0.999356  66
## 5                 max precision  0.998469     0.922222  20
## 6                    max recall  0.000045     1.000000 383
## 7               max specificity  1.000000     0.999930   0
## 8              max absolute_mcc  0.181633     0.815731  66
## 9   max min_per_class_accuracy  0.000057     0.907883 335
```

```
## 10 max mean_per_class_accuracy  0.000089     0.925964 289
## 11                     max tns  1.000000 85277.000000   0
## 12                     max fns  1.000000   109.000000   0
## 13                     max fps  0.000036 85283.000000 399
## 14                     max tps  0.000045   159.000000 383
## 15                     max tnr  1.000000     0.999930   0
## 16                     max fnr  1.000000     0.685535   0
## 17                     max fpr  0.000036     1.000000 399
## 18                     max tpr  0.000045     1.000000 383
##
## Gains/Lift Table: Extract with `h2o.gainsLift(<model>, <data>)` or `h2o.gainsLift(<model>, valid=<T/
##
##
## Scoring History:
##             timestamp   duration training_speed   epochs iterations
## 1 2022-06-16 13:27:40  0.000 sec             NA  0.00000          0
## 2 2022-06-16 13:27:41  0.352 sec 925953 obs/sec  0.50161          1
## 3 2022-06-16 13:27:46  5.395 sec 892401 obs/sec 22.57804         45
## 4 2022-06-16 13:27:51 10.565 sec 875900 obs/sec 44.14980         88
## 5 2022-06-16 13:27:52 12.092 sec 875709 obs/sec 50.17105        100
## 6 2022-06-16 13:27:52 12.218 sec 875632 obs/sec 50.17105        100
##           samples training_rmse training_logloss training_r2 training_auc
## 1        0.000000            NA               NA          NA           NA
## 2   100003.000000       0.04017          0.01750    -0.00100      0.71084
## 3  4501271.000000       0.02631          0.00596     0.57057      0.98186
## 4  8801925.000000       0.02197          0.00633     0.70044      0.99019
## 5 10002352.000000       0.02197          0.00639     0.70069      0.99054
## 6 10002352.000000       0.02197          0.00633     0.70044      0.99019
##   training_pr_auc training_lift training_classification_error validation_rmse
## 1              NA            NA                           NA              NA
## 2         0.02697      18.58125                      0.00172         0.04314
## 3         0.71677      92.90625                      0.00040         0.02797
## 4         0.69944      92.90625                      0.00040         0.02574
## 5         0.69946      92.90625                      0.00040         0.02574
## 6         0.69944      92.90625                      0.00040         0.02574
##   validation_logloss validation_r2 validation_auc validation_pr_auc
## 1                 NA            NA             NA                NA
## 2            0.02174      -0.00204        0.60635           0.01892
## 3            0.00636       0.57871        0.93501           0.72104
## 4            0.00616       0.64321        0.96431           0.71997
## 5            0.00620       0.64323        0.96316           0.72223
## 6            0.00616       0.64321        0.96431           0.71997
##   validation_lift validation_classification_error
## 1              NA                             NA
## 2        14.45560                        0.00238
## 3        81.70554                        0.00064
## 4        81.70554                        0.00064
## 5        82.33405                        0.00063
## 6        81.70554                        0.00064
##
## Variable Importances: (Extract with `h2o.varimp`)
## =================================================
##
## Variable Importances:
```

```
##   variable relative_importance scaled_importance percentage
## 1      V14            1.000000          1.000000   0.090280
## 2      V12            0.935523          0.935523   0.084459
## 3       V4            0.881529          0.881529   0.079584
## 4      V10            0.838367          0.838367   0.075688
## 5       V7            0.607980          0.607980   0.054888
##
## ---
##    variable relative_importance scaled_importance percentage
## 25      V28            0.130084          0.130084   0.011744
## 26      V23            0.127169          0.127169   0.011481
## 27      V24            0.110508          0.110508   0.009977
## 28      V21            0.109412          0.109412   0.009878
## 29      V22            0.096070          0.096070   0.008673
## 30      V15            0.050515          0.050515   0.004561
```

```
#Veamos la capacidad de prediccion del modelo
predict_test4 = h2o.predict(modeloc4, newdata = test.hex)
```

```
##   |                                                                              |
```

```
pred4 = as.data.frame(predict_test4)
head(pred4)
```

```
##   predict        p0           p1
## 1       0 0.9999546 4.535540e-05
## 2       0 0.9999538 4.615003e-05
## 3       0 0.9999537 4.626592e-05
## 4       0 0.9999560 4.404832e-05
## 5       0 0.9999559 4.409676e-05
## 6       0 0.9999447 5.528241e-05
```

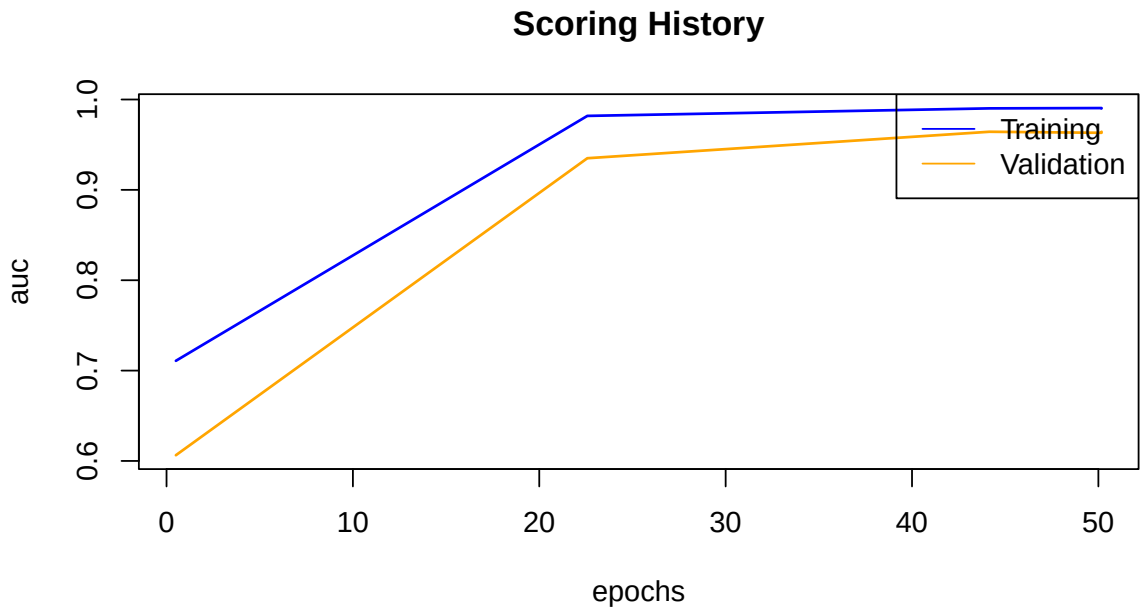
```
tabla4 = table(Obs = datos_test[,31], Pred = pred4[,1])
tabla4
```

```
##    Pred
## Obs     0     1
##   0 85267    16
##   1    39   120
```

```
aciertos4 = 100*diag(prop.table(tabla4,1))
acierto4 = 100*sum(diag(tabla4))/sum(tabla4)
acierto4
```

```
## [1] 99.93563
```

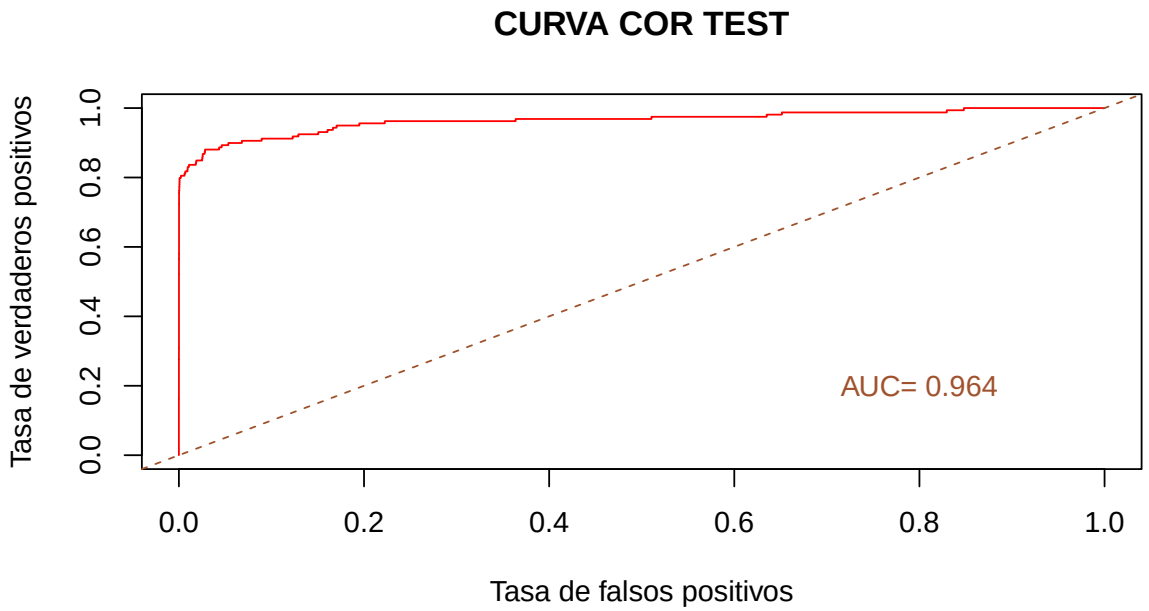
```
#Curva ROC
  #ROC para los conjuntos de entrenamiento y test
plot(modeloc4, metric="auc", lwd=1.5)
```



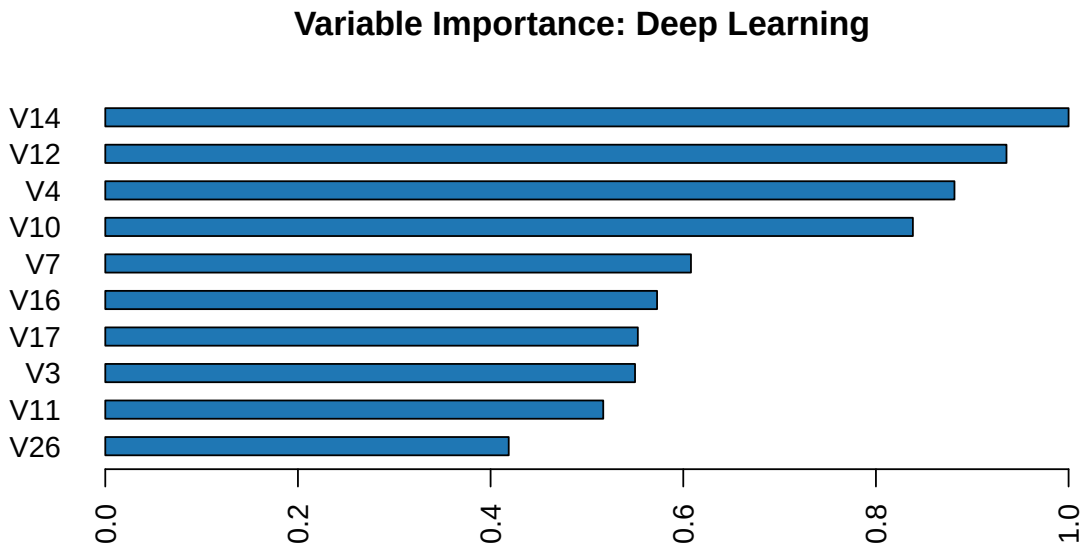

```
  #Curva ROC y AUC
prediobj4 = prediction(pred4[,3], ytest[,2])
plot(performance(prediobj4, "tpr","fpr"),main="CURVA COR TEST",
     xlab="Tasa de falsos positivos",
     ylab="Tasa de verdaderos positivos", col="red")
abline(a=0,b=1,col="sienna",lty=2)
auc4 = as.numeric(performance(prediobj4,"auc")@y.values)
cat("Area bajo la curva COR Test= ",auc4,"\n")
```

```
## Area bajo la curva COR Test=  0.964255
```

```
text(0.8,0.2,paste("AUC=",round(auc4,3)),col="sienna" )
```




```
#Importancia de las variables
h2o.varimp_plot(modeloc4)
```



Ejemplo de clasificación - Transacciones fraudulentas en Python

Raquel Beltrán Barba

DETECCIÓN DE TRANSACCIONES FRAUDULENTAS EN TARJETAS DE CRÉDITO

Primero cargamos las librerías que vamos a necesitar:

```
import numpy as np
import pandas as pd
import math
import sklearn
import sklearn.preprocessing
import datetime
import os
import matplotlib.pyplot as plt
import tensorflow as tf
import csv
import seaborn as sns
from sklearn.metrics import roc_curve
from sklearn.metrics import roc_auc_score
```

Lectura y modificación del conjunto de datos

```
datos= pd.read_csv('creditcard.csv', delimiter = ",", index_col = 0)
```

Estudio de las variables

```
datos.info()
```

```
## <class 'pandas.core.frame.DataFrame'>
## Float64Index: 284807 entries, 0.0 to 172792.0
## Data columns (total 30 columns):
##  #   Column  Non-Null Count   Dtype
## ---  ------  --------------   -----
##  0   V1      284807 non-null  float64
##  1   V2      284807 non-null  float64
##  2   V3      284807 non-null  float64
##  3   V4      284807 non-null  float64
##  4   V5      284807 non-null  float64
##  5   V6      284807 non-null  float64
##  6   V7      284807 non-null  float64
```

```
##  7   V8      284807 non-null  float64
##  8   V9      284807 non-null  float64
##  9   V10     284807 non-null  float64
##  10  V11     284807 non-null  float64
##  11  V12     284807 non-null  float64
##  12  V13     284807 non-null  float64
##  13  V14     284807 non-null  float64
##  14  V15     284807 non-null  float64
##  15  V16     284807 non-null  float64
##  16  V17     284807 non-null  float64
##  17  V18     284807 non-null  float64
##  18  V19     284807 non-null  float64
##  19  V20     284807 non-null  float64
##  20  V21     284807 non-null  float64
##  21  V22     284807 non-null  float64
##  22  V23     284807 non-null  float64
##  23  V24     284807 non-null  float64
##  24  V25     284807 non-null  float64
##  25  V26     284807 non-null  float64
##  26  V27     284807 non-null  float64
##  27  V28     284807 non-null  float64
##  28  Amount  284807 non-null  float64
##  29  Class   284807 non-null  int64
## dtypes: float64(29), int64(1)
## memory usage: 67.4 MB
```

```
datos.shape
```

```
## (284807, 30)
```

```
#Correlaciones entre las variables
corr_matrix = datos.corr(method='pearson')

    #A traves de la funcion tidy_corr_matrix obtenemos las variables con mayor correlacion
def tidy_corr_matrix(corr_mat):
    '''Función para convertir una matriz de correlación de pandas en formato tidy.'''
    corr_mat = corr_mat.stack().reset_index()
    corr_mat.columns = ['variable_1','variable_2','r']
    corr_mat = corr_mat.loc[corr_mat['variable_1'] != corr_mat['variable_2'], :]
    corr_mat['abs_r'] = np.abs(corr_mat['r'])
    corr_mat = corr_mat.sort_values('abs_r', ascending=False)

    return(corr_mat)

tidy_corr_matrix(corr_matrix).head(10)
```

```
##     variable_1 variable_2         r     abs_r
## 841     Amount         V2 -0.531409  0.531409
## 58          V2     Amount -0.531409  0.531409
## 846     Amount         V7  0.397311  0.397311
## 208         V7     Amount  0.397311  0.397311
## 844     Amount         V5 -0.386356  0.386356
## 148         V5     Amount -0.386356  0.386356
```

```
## 859     Amount        V20  0.339403  0.339403
## 598        V20     Amount  0.339403  0.339403
## 509        V17      Class -0.326481  0.326481
## 886      Class        V17 -0.326481  0.326481
```

Partición Entrenamiento/Test

```
X = datos.loc[:, datos.columns != 'Class']
y = datos["Class"]
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

print('X_train.shape = ',X_train.shape)
```

```
## X_train.shape =  (199364, 29)
```

```
print('y_train.shape = ', y_train.shape)
```

```
## y_train.shape =  (199364,)
```

```
print('X_test.shape = ', X_test.shape)
```

```
## X_test.shape =  (85443, 29)
```

```
print('y_test.shape = ',y_test.shape)
```

```
## y_test.shape =  (85443,)
```

Comenzamos a trabajar con los modelos de Redes Neuronales

```
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import accuracy_score

def plot_roc_curve(fper, tper):
    plt.plot(fper, tper, color='red', label='ROC')
    plt.plot([0, 1], [0, 1], color='green', linestyle='--')
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('Receiver Operating Characteristic Curve')
    plt.legend()
    plt.show()
```

Modelo 1: Red con dos capas ocultas de 25 nodos cada una

```
modeloc1 = MLPClassifier(hidden_layer_sizes=(25,25),
                learning_rate_init=0.01,
                solver = 'adam',
                max_iter = 1000,
                random_state = 7978)

modeloc1.fit(X=X_train, y=y_train)
```

```
## MLPClassifier(hidden_layer_sizes=(25, 25), learning_rate_init=0.01,
##               max_iter=1000, random_state=7978)
```

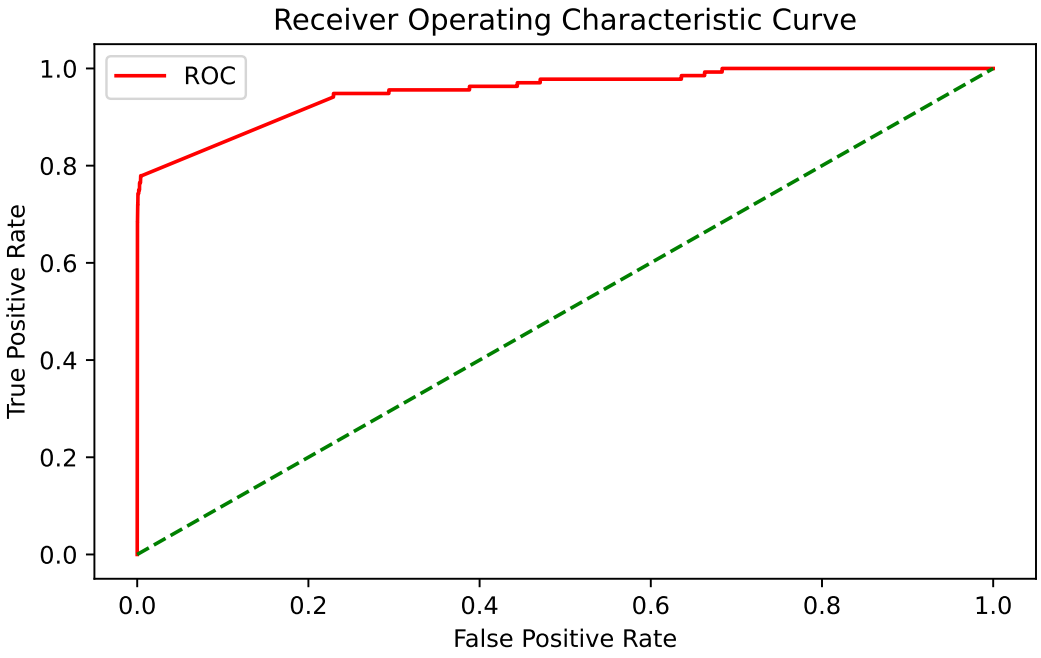
```
  #Capacidad de predicción
predictions = (modeloc1.predict(X_test) > 0.5).astype(int)
    #Matriz de confusión
sklearn.metrics.confusion_matrix(y_test, predictions)
```

```
## array([[85293,    14],
##        [   48,    88]], dtype=int64)
```

```
  #Accuracy score
accuracy_score(y_test, predictions)
```

```
## 0.9992743700478681
```

```
  #Curva ROC y AUC
prob = modeloc1.predict_proba(X_test)
probs = prob[:,1]
fpr, tpr, thresholds = roc_curve(y_test, probs)
plot_roc_curve(fpr, tpr)
```



```
auc = roc_auc_score(y_test, probs)
print('AUC: %.2f' % auc)
```

```
## AUC: 0.95
```

Modelo 2: Red con dos capas ocultas de 10 nodos cada una

```
modeloc2 = MLPClassifier(hidden_layer_sizes=(10,10),
                learning_rate_init=0.01,
                solver = 'adam',
                max_iter = 1000,
                random_state = 7978)

modeloc2.fit(X=X_train, y=y_train)
```

```
## MLPClassifier(hidden_layer_sizes=(10, 10), learning_rate_init=0.01,
##               max_iter=1000, random_state=7978)
```

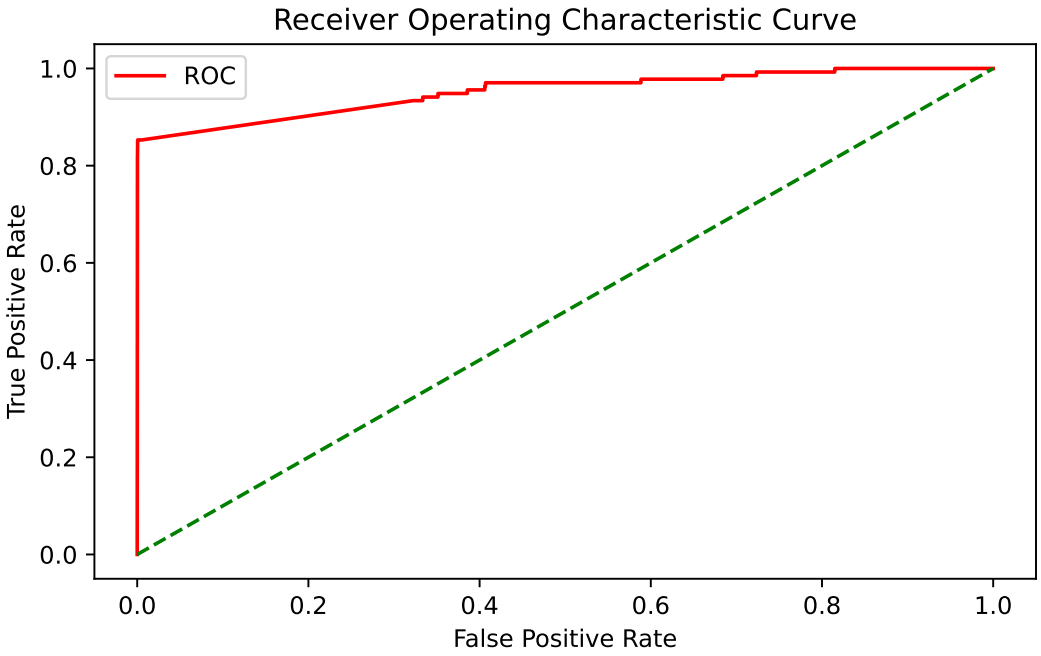
```
  #Capacidad de predicción
predictions2 = (modeloc2.predict(X_test) > 0.5).astype(int)
    #Matriz de confusión
sklearn.metrics.confusion_matrix(y_test, predictions2)
```

```
## array([[85280,    27],
##        [   24,   112]], dtype=int64)
```

```
  #Accuracy score
accuracy_score(y_test, predictions2)
```

```
## 0.999403110845827
```

```
  #Curva ROC y AUC
prob2 = modeloc2.predict_proba(X_test)
probs2 = prob2[:,1]
fpr2, tpr2, thresholds2 = roc_curve(y_test, probs2)
plot_roc_curve(fpr2, tpr2)
```




```
auc2 = roc_auc_score(y_test, probs2)
print('AUC: %.2f' % auc2)
```

```
## AUC: 0.95
```

Modelo 3: Red con una capa oculta de 25 nodos

```
modeloc3 = MLPClassifier(hidden_layer_sizes=(25),
                learning_rate_init=0.01,
                solver = 'adam',
                max_iter = 1000,
                random_state = 7978)

modeloc3.fit(X=X_train, y=y_train)
```

```
## MLPClassifier(hidden_layer_sizes=25, learning_rate_init=0.01, max_iter=1000,
##               random_state=7978)
```

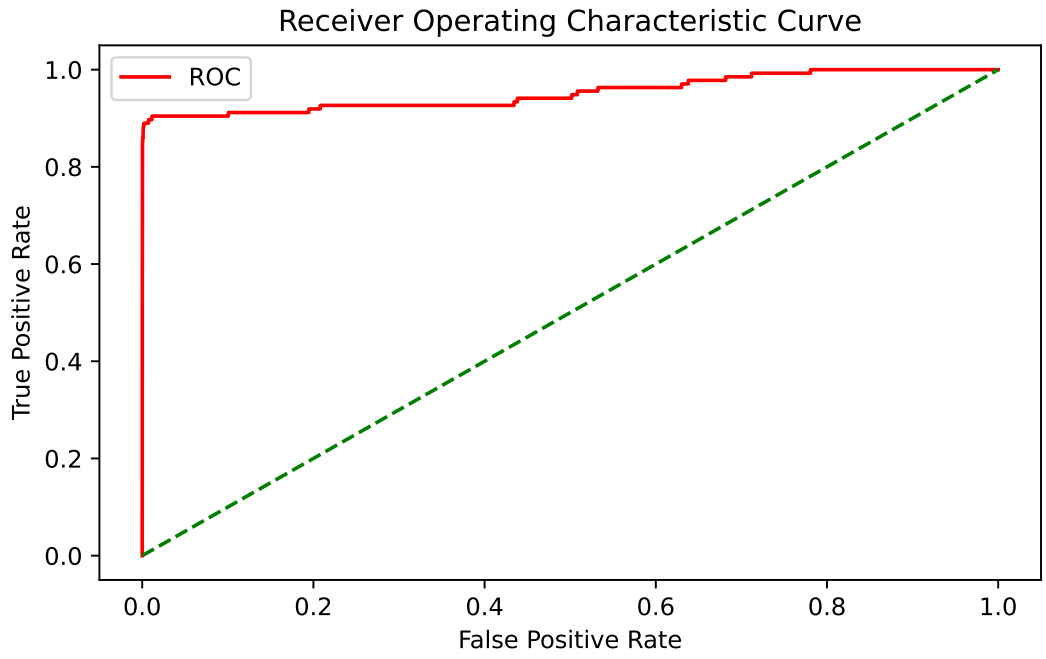
```
  #Capacidad de predicción
predictions3 = (modeloc3.predict(X_test) > 0.5).astype(int)
    #Matriz de confusión
sklearn.metrics.confusion_matrix(y_test, predictions3)
```

```
## array([[85279,    28],
##        [   20,   116]], dtype=int64)
```

```
  #Accuracy score
accuracy_score(y_test, predictions3)
```

```
## 0.9994382219725431
```

```
  #Curva ROC y AUC
prob3 = modeloc3.predict_proba(X_test)
probs3 = prob3[:,1]
fpr3, tpr3, thresholds3 = roc_curve(y_test, probs3)
plot_roc_curve(fpr3, tpr3)
```




```
auc3 = roc_auc_score(y_test, probs3)
print('AUC: %.2f' % auc3)
```

```
## AUC: 0.95
```

Modelo 4: Red con una capa oculta de 5 nodos

```
modeloc4 = MLPClassifier(hidden_layer_sizes=(5),
                         learning_rate_init=0.01,
                         solver = 'adam',
                         max_iter = 1000,
                         random_state = 7978)

modeloc4.fit(X=X_train, y=y_train)
```

```
## MLPClassifier(hidden_layer_sizes=5, learning_rate_init=0.01, max_iter=1000,
##               random_state=7978)
```

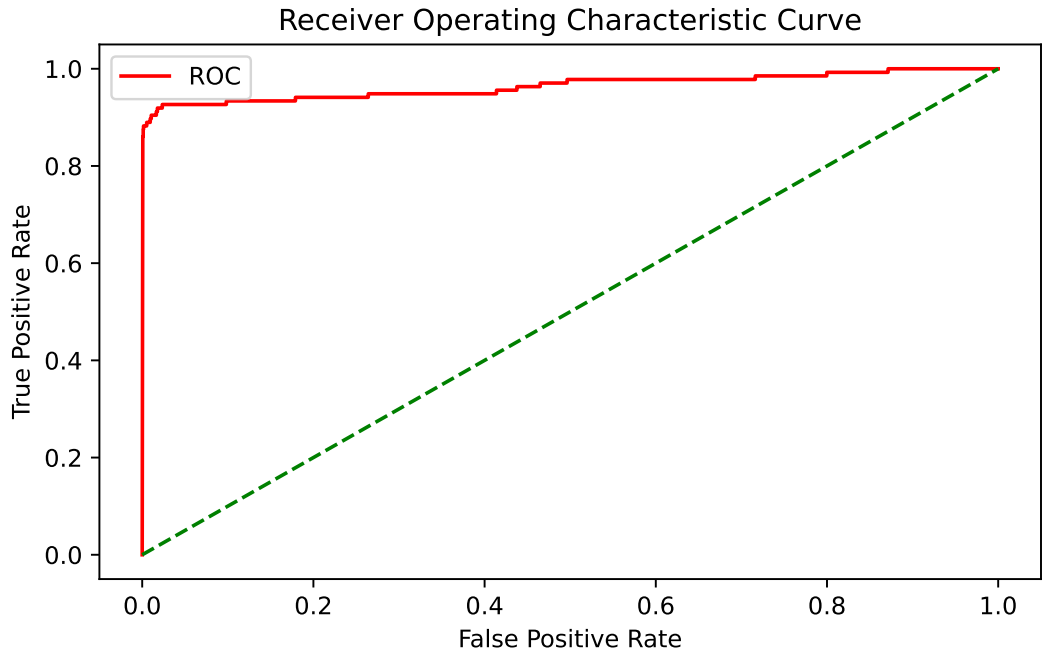
```
  #Capacidad de predicción
predictions4 = (modeloc4.predict(X_test) > 0.5).astype(int)
    #Matriz de confusión
sklearn.metrics.confusion_matrix(y_test, predictions4)
```

```
## array([[85271,    36],
##        [   22,   114]], dtype=int64)
```

```
  #Accuracy score
accuracy_score(y_test, predictions4)
```

```
## 0.9993211848834895
```

```
  #Curva ROC y AUC
prob4 = modeloc4.predict_proba(X_test)
probs4 = prob4[:,1]
fpr4, tpr4, thresholds4 = roc_curve(y_test, probs4)
plot_roc_curve(fpr4, tpr4)
```




```
auc4 = roc_auc_score(y_test, probs4)
print('AUC: %.2f' % auc4)
```

```
## AUC: 0.96
```

Ejemplo de regresión - Precio del vino español en RStudio

Raquel Beltrán Barba

PREDICCIÓN DEL PRECIO DEL VINO ESPAÑOL

Primero cargamos las librerías que vamos a necesitar:

```
set.seed(07978) #semilla
library(readr)
library(h2o)
library(corrplot)
library(kableExtra)
library(RSNNS)
```

Lectura y modificación del conjunto de datos

```
datos = read_csv("wines_SPA.csv")
datos = as.data.frame(datos)
datos = datos[,-6] #eliminamos la variable country, que siempre es España
str(datos)
```

```
## 'data.frame':    7500 obs. of  10 variables:
##  $ winery     : chr  "Teso La Monja" "Artadi" "Vega Sicilia" "Vega Sicilia" ...
##  $ wine       : chr  "Tinto" "Vina El Pison" "Unico" "Unico" ...
##  $ year       : chr  "2013" "2018" "2009" "1999" ...
##  $ rating     : num  4.9 4.9 4.8 4.8 4.8 4.8 4.8 4.8 4.8 4.8 ...
##  $ num_reviews: num  58 31 1793 1705 1309 ...
##  $ region     : chr  "Toro" "Vino de Espana" "Ribera del Duero" "Ribera del Duero" ...
##  $ price      : num  995 314 325 693 778 ...
##  $ type       : chr  "Toro Red" "Tempranillo" "Ribera Del Duero Red" "Ribera Del Duero Red" ...
##  $ body       : num  5 4 5 5 5 5 5 5 5 5 ...
##  $ acidity    : num  3 2 3 3 3 3 3 3 3 3 ...
```

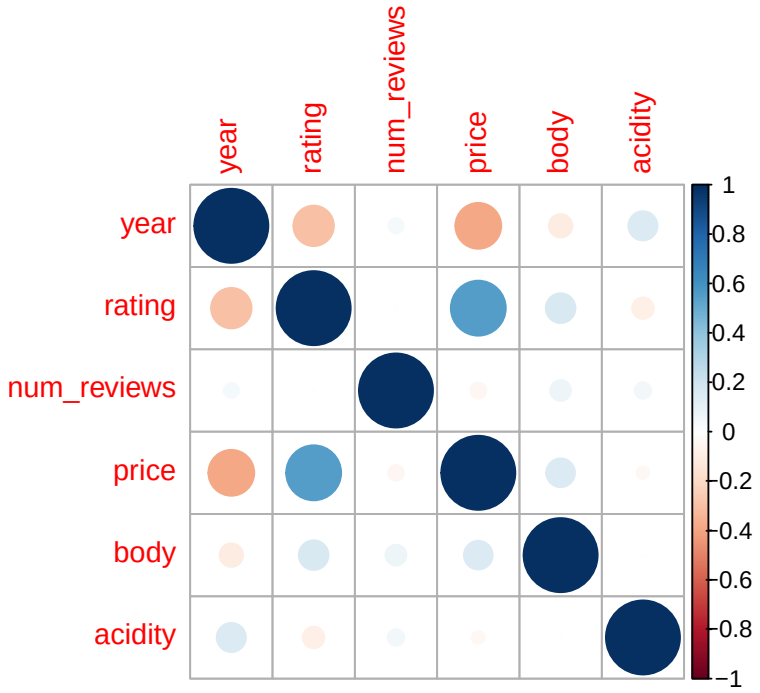
```
  # Cambiamos el tipo de variables que lo necesiten
datos[,1] = as.factor(datos[,1])
datos[,2] = as.factor(datos[,2])
datos[,3] = as.numeric(datos[,3])
datos[,6] = as.factor(datos[,6])
datos[,8] = as.factor(datos[,8])
datos = na.omit(datos)
```

Estudio de las variables

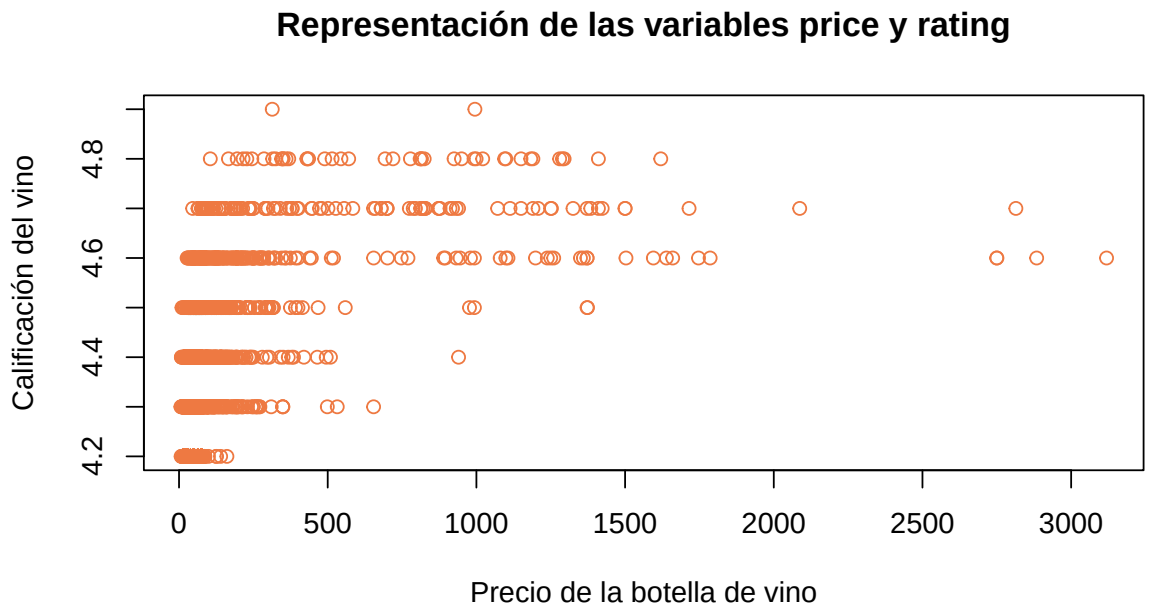
```
summary(datos)
```

```
##               winery                  wine           year          rating
##  Contino          : 414   Reserva       : 422   Min.   :1910   Min.   :4.20
##  Artadi           : 239   Gran Reserva  : 415   1st Qu.:2011   1st Qu.:4.20
##  La Rioja Alta    : 228   Rioja Reserva : 218   Median :2015   Median :4.20
##  Sierra Cantabria: 215   Corimbo I     : 202   Mean   :2013   Mean   :4.26
##  Vina Pedrosa     : 207   El Viejo      : 202   3rd Qu.:2017   3rd Qu.:4.30
##  Imperial         : 206   Rioja Graciano: 202   Max.   :2021   Max.   :4.90
##  (Other)          :4561   (Other)       :4409
##   num_reviews                  region         price
##  Min.   :   25.0   Rioja           :2221   Min.   :   6.26
##  1st Qu.:  388.0   Ribera del Duero:1280   1st Qu.:  19.98
##  Median :  402.0   Priorato        : 622   Median :  31.63
##  Mean   :  440.1   Toro            : 264   Mean   :  67.40
##  3rd Qu.:  417.0   Vino de Espana  : 238   3rd Qu.:  61.94
##  Max.   :16505.0   Rias Baixas     : 224   Max.   :3119.08
##                    (Other)         :1221
##                    type           body          acidity
##  Rioja Red            :2143   Min.   :2.000   Min.   :1.000
##  Ribera Del Duero Red:1277   1st Qu.:4.000   1st Qu.:3.000
##  Red                  : 786   Median :4.000   Median :3.000
##  Priorat Red          : 620   Mean   :4.164   Mean   :2.947
##  Tempranillo          : 267   3rd Qu.:5.000   3rd Qu.:3.000
##  Toro Red             : 261   Max.   :5.000   Max.   :3.000
##  (Other)              : 716
```

```
#Correlacion entre las variables
datos2 = datos[,-c(1,2,6,8)]
corr = cor(datos2)
correlaciones = round(corr,4)
corrplot(correlaciones)
```



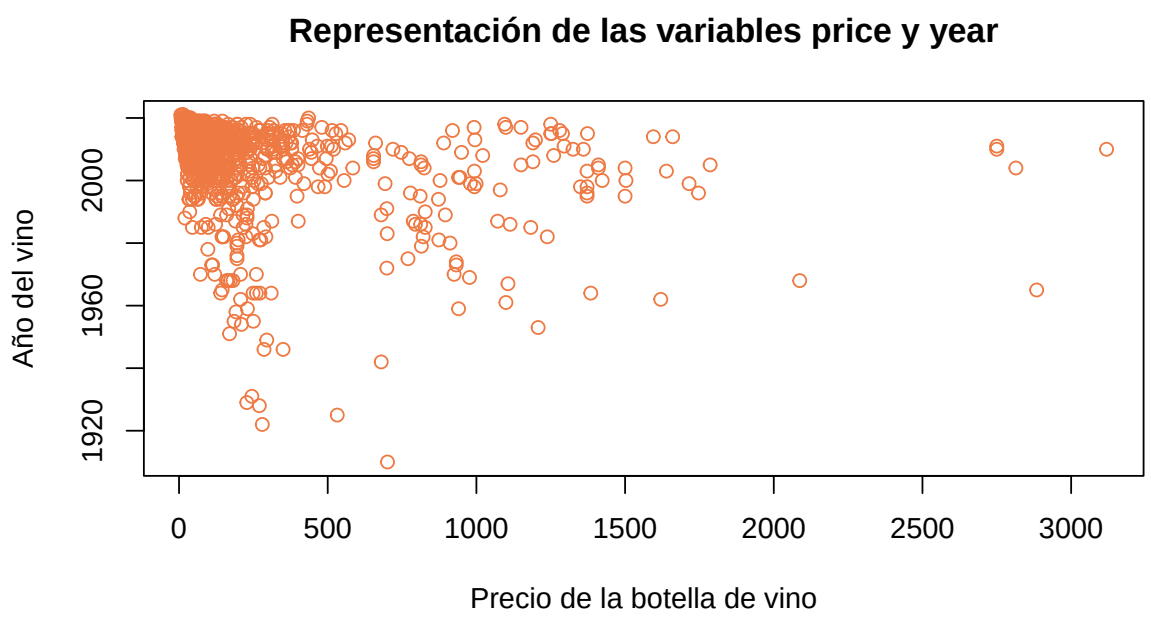

```
plot(datos$price, datos$rating,  xlab="Precio de la botella de vino",
     ylab = "Calificación del vino", col="sienna2",
     main="Representación de las variables price y rating")
```




```
cor(datos$price, datos$rating)
```

```
## [1] 0.5519432
```

```
plot(datos$price, datos$year,  xlab="Precio de la botella de vino",
     ylab = "Año del vino", col="sienna2",
     main="Representación de las variables price y year")
```




```
cor(datos$price, datos$year)
```

```
## [1] -0.3843515
```

Partición Entrenamiento/Test

```
n = nrow(datos)
nent = ceiling(0.7*n)    #n casos de entrenamiento
ntest = n-nent           #n casos de test
indin = 1:n
indient = sort(sample(indin,nent))
inditest = setdiff(indin,indient)
datos_ent = datos[indient,]
datos_test = datos[inditest,]
```

Comenzamos a trabajar con el modelo H2O

```
  #Inicializacion del servicio h2o
localH2O = h2o.init(nthreads = -1)
```

```
##  Connection successful!
```

```
##
## R is connected to the H2O cluster:
##     H2O cluster uptime:         1 hours 20 minutes
##     H2O cluster timezone:       Europe/Madrid
##     H2O data parsing timezone:  UTC
##     H2O cluster version:        3.36.0.4
##     H2O cluster version age:    2 months and 16 days
##     H2O cluster name:           H2O_started_from_R_raque_cki562
##     H2O cluster total nodes:    1
##     H2O cluster total memory:   3.10 GB
##     H2O cluster total cores:    8
##     H2O cluster allowed cores:  8
##     H2O cluster healthy:        TRUE
##     H2O Connection ip:          localhost
##     H2O Connection port:        54321
##     H2O Connection proxy:       NA
##     H2O Internal Security:      FALSE
##     R Version:                  R version 4.2.0 (2022-04-22 ucrt)
```

```
  #Convertir los data.frames en objetos que puedan ser procesados
train.hex = as.h2o(datos_ent)
```

```
##   |                                                                      |
```

```
test.hex = as.h2o(datos_test)
```

```
##   |                                                                      |
```

A continuación creo y estudio los distintos modelos propuestos:

Modelo 1: Red con dos capas ocultas de 8 nodos cada una

```
modelo_r1 = h2o.deeplearning(
  x = c(1,2,3,4,5,6,8,9,10), #variables predictoras
  y = 7, #variable respuesta
  training_frame = train.hex, #datos de entrenamiento
  validation_frame = test.hex, #datos de validacion/test
  activation = 'RectifierWithDropout',
  hidden = c(8,8), #tamaño de las capas ocultas
  hidden_dropout_ratio = c(0.5, 0.5),
  input_dropout_ratio = 0.2,
  epochs = 50, #iteraciones
  l1 = 1e-5,
  l2 = 1e-5,
  rho = 0.99,
  epsilon = 1e-8)
```

```
##   |                                                                              |
```

```
summary(modelo_r1)
```

```
## Model Details:
## ==============
##
## H2ORegressionModel: deeplearning
## Model Key:  DeepLearning_model_R_1655376581048_636
## Status of Neuron Layers: predicting price, regression, gaussian distribution, Quadratic loss, 7.969 w
##   layer units             type dropout       l1       l2 mean_rate rate_rms
## 1     1   985            Input 20.00 %       NA       NA        NA       NA
## 2     2     8 RectifierDropout 50.00 % 0.000010 0.000010  0.130510 0.327977
## 3     3     8 RectifierDropout 50.00 % 0.000010 0.000010  0.002446 0.007122
## 4     4     1           Linear      NA 0.000010 0.000010  0.001311 0.001460
##   momentum mean_weight weight_rms mean_bias bias_rms
## 1       NA          NA         NA        NA       NA
## 2 0.000000   -0.010822   0.071311  0.651801 0.841226
## 3 0.000000    0.007556   0.450483  0.339081 0.415686
## 4 0.000000   -0.267058   0.425221  0.032609 0.000000
##
## H2ORegressionMetrics: deeplearning
## ** Reported on training data. **
## ** Metrics reported on full training frame **
##
## MSE:  20686.41
## RMSE:  143.8277
## MAE:  32.30078
## RMSLE:  NaN
## Mean Residual Deviance :  20686.41
##
##
```

```
## H2ORegressionMetrics: deeplearning
## ** Reported on validation data. **
## ** Metrics reported on full validation frame **
##
## MSE:  10689.4
## RMSE:  103.3896
## MAE:  27.25684
## RMSLE:  NaN
## Mean Residual Deviance :  10689.4
##
##
##
##
## Scoring History:
##             timestamp   duration training_speed   epochs iterations
## 1 2022-06-16 14:10:14  0.000 sec             NA  0.00000          0
## 2 2022-06-16 14:10:14  0.179 sec 174139 obs/sec  5.00000          1
## 3 2022-06-16 14:10:15  1.361 sec 168477 obs/sec 50.00000         10
##         samples training_rmse training_deviance training_mae training_r2
## 1      0.000000            NA                NA           NA          NA
## 2  21245.000000     174.20626       30347.82061     42.48765     0.04018
## 3 212450.000000     143.82770       20686.40687     32.30078     0.34574
##   validation_rmse validation_deviance validation_mae validation_r2
## 1              NA                  NA             NA            NA
## 2       128.12518         16416.06189       34.65397       0.06037
## 3       103.38956         10689.40143       27.25684       0.38816
##
## Variable Importances: (Extract with `h2o.varimp`)
## =================================================
##
## Variable Importances:
##                variable relative_importance scaled_importance percentage
## 1   winery.Vega Sicilia            1.000000          1.000000   0.008638
## 2                rating            0.700464          0.700464   0.006051
## 3                  year            0.640521          0.640521   0.005533
## 4            wine.Unico            0.511693          0.511693   0.004420
## 5 winery.Carmelo Rodero            0.485138          0.485138   0.004191
##
## ---
##               variable relative_importance scaled_importance percentage
## 980         wine.Cyclo            0.038349          0.038349   0.000331
## 981        wine.Monada            0.037606          0.037606   0.000325
## 982   wine.missing(NA)            0.000000          0.000000   0.000000
## 983 winery.missing(NA)            0.000000          0.000000   0.000000
## 984 region.missing(NA)            0.000000          0.000000   0.000000
## 985   type.missing(NA)            0.000000          0.000000   0.000000
```

```
#Coeficiente R^2
h2o.r2(modelo_r1)
```

```
## [1] 0.3457417
```

```
#Capacidad de predicción
predict_test = h2o.predict(modelo_r1, newdata = test.hex)
```

```
##   |                                                                      |
```

```
pred = as.data.frame(predict_test)
head(pred)
```

```
##    predict
## 1 317.0526
## 2 353.6677
## 3 366.3926
## 4 324.5679
## 5 298.3987
## 6 295.0417
```

Modelo 2: Red con una capa oculta y 15 nodos

```
modelo_r6 = h2o.deeplearning(
  x = c(1,2,3,4,5,6,8,9,10), #variables predictoras
  y = 7, #variable respuesta
  training_frame = train.hex, #datos de entrenamiento
  validation_frame = test.hex, #datos de validacion/test
  activation = 'RectifierWithDropout',
  hidden = 15, #tamaño de las capas ocultas
  hidden_dropout_ratio =  0.5,
  input_dropout_ratio = 0.2,
  epochs = 50, #iteraciones
  l1 = 1e-5,
  l2 = 1e-5,
  rho = 0.99,
  epsilon = 1e-8)
```

```
##   |                                                                      |
```

```
summary(modelo_r6)
```

```
## Model Details:
## ==============
##
## H2ORegressionModel: deeplearning
## Model Key:  DeepLearning_model_R_1655376581048_637
## Status of Neuron Layers: predicting price, regression, gaussian distribution, Quadratic loss, 14.806
##   layer units             type dropout       l1       l2 mean_rate rate_rms
## 1     1   985            Input 20.00 %       NA       NA        NA       NA
## 2     2    15 RectifierDropout 50.00 % 0.000010 0.000010  0.028270 0.140231
## 3     3     1           Linear      NA 0.000010 0.000010  0.000692 0.000697
##   momentum mean_weight weight_rms mean_bias bias_rms
## 1       NA          NA         NA        NA       NA
```

```
## 2 0.000000   -0.007734   0.066690  0.013158 0.125649
## 3 0.000000    0.048538   0.562107  0.045499 0.000000
##
## H2ORegressionMetrics: deeplearning
## ** Reported on training data. **
## ** Metrics reported on full training frame **
##
## MSE:  10832.55
## RMSE:  104.0795
## MAE:  31.70916
## RMSLE:  NaN
## Mean Residual Deviance :  10832.55
##
##
## H2ORegressionMetrics: deeplearning
## ** Reported on validation data. **
## ** Metrics reported on full validation frame **
##
## MSE:  6039.789
## RMSE:  77.71608
## MAE:  31.8142
## RMSLE:  NaN
## Mean Residual Deviance :  6039.789
##
##
##
##
## Scoring History:
##             timestamp   duration training_speed   epochs iterations
## 1 2022-06-16 14:10:18  0.000 sec             NA  0.00000          0
## 2 2022-06-16 14:10:18  0.232 sec 147534 obs/sec  5.00000          1
## 3 2022-06-16 14:10:19  1.608 sec 146114 obs/sec 50.00000         10
##         samples training_rmse training_deviance training_mae training_r2
## 1      0.000000            NA                NA           NA          NA
## 2  21245.000000     138.48764       19178.82519     44.43237     0.39342
## 3 212450.000000     104.07953       10832.54840     31.70916     0.65739
##   validation_rmse validation_deviance validation_mae validation_r2
## 1              NA                  NA             NA            NA
## 2       104.65263         10952.17383       40.70121       0.37312
## 3        77.71608          6039.78949       31.81420       0.65429
##
## Variable Importances: (Extract with `h2o.varimp`)
## =================================================
##
## Variable Importances:
##                   variable relative_importance scaled_importance percentage
## 1 winery.Dominio de Pingus            1.000000          1.000000   0.007099
## 2         wine.Valbuena 5o            0.916805          0.916805   0.006508
## 3        wine.Gran Reserva            0.755750          0.755750   0.005365
## 4   winery.Alvaro Palacios            0.738480          0.738480   0.005242
## 5      winery.Vega Sicilia            0.690146          0.690146   0.004899
##
## ---
##                            variable relative_importance scaled_importance
```

```
## 980         wine.Vatan Tinta de Toro            0.060717          0.060717
## 981 winery.Bodega Juan Carlos Sancha            0.057424          0.057424
## 982                  wine.missing(NA)           0.000000          0.000000
## 983               winery.missing(NA)            0.000000          0.000000
## 984               region.missing(NA)            0.000000          0.000000
## 985                  type.missing(NA)           0.000000          0.000000
##     percentage
## 980   0.000431
## 981   0.000408
## 982   0.000000
## 983   0.000000
## 984   0.000000
## 985   0.000000
```

```
#Coeficiente R^2
h2o.r2(modelo_r6)
```

```
## [1] 0.6573941
```

```
#Capacidad de predicción
predict_test6 = h2o.predict(modelo_r6, newdata = test.hex)
```

```
##   |                                                                              |
```

```
pred6 = as.data.frame(predict_test6)
head(pred6)
```

```
##    predict
## 1 624.4466
## 2 721.6425
## 3 779.8993
## 4 673.9582
## 5 665.1923
## 6 659.3607
```

Modelo 3: Red con una capa oculta y 60 nodos

```
modelo_r7 = h2o.deeplearning(
  x = c(1,2,3,4,5,6,8,9,10), #variables predictoras
  y = 7, #variable respuesta
  training_frame = train.hex, #datos de entrenamiento
  validation_frame = test.hex, #datos de validacion/test
  activation = 'RectifierWithDropout',
  hidden = 60, #tamaño de las capas ocultas
  hidden_dropout_ratio =  0.5,
  input_dropout_ratio = 0.2,
  epochs = 50, #iteraciones
  l1 = 1e-5,
  l2 = 1e-5,
  rho = 0.99,
  epsilon = 1e-8)
```

```
##   |                                                                      |
```

```
summary(modelo_r7)
```

```
## Model Details:
## ==============
##
## H2ORegressionModel: deeplearning
## Model Key:  DeepLearning_model_R_1655376581048_638
## Status of Neuron Layers: predicting price, regression, gaussian distribution, Quadratic loss, 59.221
##   layer units             type dropout       l1       l2 mean_rate rate_rms
## 1     1   985            Input 20.00 %       NA       NA        NA       NA
## 2     2    60 RectifierDropout 50.00 % 0.000010 0.000010  0.067521 0.219405
## 3     3     1           Linear      NA 0.000010 0.000010  0.001444 0.001616
##   momentum mean_weight weight_rms mean_bias bias_rms
## 1       NA          NA         NA        NA       NA
## 2 0.000000   -0.003965   0.059617 -0.056053 0.117727
## 3 0.000000   -0.078516   0.347052 -0.060683 0.000000
##
## H2ORegressionMetrics: deeplearning
## ** Reported on training data. **
## ** Metrics reported on full training frame **
##
## MSE:  8727.75
## RMSE:  93.42243
## MAE:  23.52113
## RMSLE:  NaN
## Mean Residual Deviance :  8727.75
##
##
## H2ORegressionMetrics: deeplearning
## ** Reported on validation data. **
## ** Metrics reported on full validation frame **
##
## MSE:  4035.965
## RMSE:  63.52925
## MAE:  23.59744
## RMSLE:  NaN
## Mean Residual Deviance :  4035.965
##
##
##
##
## Scoring History:
##             timestamp   duration training_speed   epochs iterations
## 1 2022-06-16 14:10:21  0.000 sec             NA  0.00000          0
## 2 2022-06-16 14:10:22  0.746 sec  44632 obs/sec  5.00000          1
## 3 2022-06-16 14:10:25  4.090 sec  58109 obs/sec 50.00000         10
##         samples training_rmse training_deviance training_mae training_r2
## 1      0.000000            NA                NA           NA          NA
## 2  21245.000000     134.70202       18144.63423     62.28928     0.42613
## 3 212450.000000      93.42243        8727.75019     23.52113     0.72396
##   validation_rmse validation_deviance validation_mae validation_r2
## 1              NA                  NA             NA            NA
```

```
## 2       103.48855         10709.88053       59.66295       0.38698
## 3        63.52925          4035.96518       23.59744       0.76899
##
## Variable Importances: (Extract with `h2o.varimp`)
## =================================================
##
## Variable Importances:
##                   variable relative_importance scaled_importance percentage
## 1 winery.Dominio de Pingus            1.000000          1.000000   0.006261
## 2         wine.Valbuena 5o            0.762863          0.762863   0.004777
## 3   winery.Alvaro Palacios            0.694873          0.694873   0.004351
## 4              wine.Pingus            0.682976          0.682976   0.004276
## 5           type.Rioja Red            0.636426          0.636426   0.003985
##
## ---
##               variable relative_importance scaled_importance percentage
## 980   winery.Casa Rojo            0.111478          0.111478   0.000698
## 981       winery.Pinea            0.111180          0.111180   0.000696
## 982   wine.missing(NA)            0.000000          0.000000   0.000000
## 983 winery.missing(NA)            0.000000          0.000000   0.000000
## 984 region.missing(NA)            0.000000          0.000000   0.000000
## 985   type.missing(NA)            0.000000          0.000000   0.000000
```

```
#Coeficiente R^2
h2o.r2(modelo_r7)
```

```
## [1] 0.7239635
```

```
#Capacidad de predicción
predict_test7 = h2o.predict(modelo_r7, newdata = test.hex)
```

```
##   |                                                                              |
```

```
pred7 = as.data.frame(predict_test7)
head(pred7)
```

```
##    predict
## 1 577.5897
## 2 693.2814
## 3 759.6178
## 4 631.7047
## 5 592.0771
## 6 580.7862
```

Modelo 4: Red con tres capas ocultas y 30 nodos en cada una

```
modelo_r8 = h2o.deeplearning(
  x = c(1,2,3,4,5,6,8,9,10), #variables predictoras
  y = 7, #variable respuesta
  training_frame = train.hex, #datos de entrenamiento
```

```
  validation_frame = test.hex, #datos de validacion/test
  activation = 'RectifierWithDropout',
  hidden = c(30,30,30), #tamaño de las capas ocultas
  hidden_dropout_ratio =  c(0.5,0.5,0.5),
  input_dropout_ratio = 0.2,
  epochs = 50, #iteraciones
  l1 = 1e-5,
  l2 = 1e-5,
  rho = 0.99,
  epsilon = 1e-8)
```

```
##   |                                                                              |
```

```
summary(modelo_r8)
```

```
## Model Details:
## ==============
##
## H2ORegressionModel: deeplearning
## Model Key:  DeepLearning_model_R_1655376581048_639
## Status of Neuron Layers: predicting price, regression, gaussian distribution, Quadratic loss, 31.471
##   layer units             type dropout       l1       l2 mean_rate rate_rms
## 1     1   985            Input 20.00 %       NA       NA        NA       NA
## 2     2    30 RectifierDropout 50.00 % 0.000010 0.000010  0.046700 0.139084
## 3     3    30 RectifierDropout 50.00 % 0.000010 0.000010  0.001555 0.002554
## 4     4    30 RectifierDropout 50.00 % 0.000010 0.000010  0.002076 0.003003
## 5     5     1           Linear      NA 0.000010 0.000010  0.000343 0.000377
##   momentum mean_weight weight_rms mean_bias bias_rms
## 1       NA          NA         NA        NA       NA
## 2 0.000000   -0.000272   0.061949  0.434200 0.477555
## 3 0.000000   -0.014735   0.275033  1.179456 0.408683
## 4 0.000000   -0.013421   0.199597  1.024396 0.174945
## 5 0.000000    0.073572   0.182249 -0.212218 0.000000
##
## H2ORegressionMetrics: deeplearning
## ** Reported on training data. **
## ** Metrics reported on full training frame **
##
## MSE:  9068.249
## RMSE:  95.22735
## MAE:  38.95959
## RMSLE:  0.7092156
## Mean Residual Deviance :  9068.249
##
##
## H2ORegressionMetrics: deeplearning
## ** Reported on validation data. **
## ** Metrics reported on full validation frame **
##
## MSE:  4372.927
## RMSE:  66.12811
## MAE:  38.40372
```

```
## RMSLE:  0.7345151
## Mean Residual Deviance :  4372.927
##
##
##
##
## Scoring History:
##             timestamp   duration training_speed   epochs iterations
## 1 2022-06-16 14:10:27  0.000 sec             NA  0.00000          0
## 2 2022-06-16 14:10:28  0.491 sec  61579 obs/sec  5.00000          1
## 3 2022-06-16 14:10:31  3.589 sec  63952 obs/sec 50.00000         10
##         samples training_rmse training_deviance training_mae training_r2
## 1      0.000000            NA                NA           NA          NA
## 2  21245.000000     140.09576       19626.82118     60.35365     0.37925
## 3 212450.000000      95.22735        9068.24902     38.95959     0.71319
##   validation_rmse validation_deviance validation_mae validation_r2
## 1              NA                  NA             NA            NA
## 2       107.35802         11525.74373       55.65108       0.34029
## 3        66.12811          4372.92705       38.40372       0.74970
##
## Variable Importances: (Extract with ‘h2o.varimp‘)
## =================================================
##
## Variable Importances:
##                             variable relative_importance scaled_importance
## 1            winery.Dominio de Pingus            1.000000          1.000000
## 2                        wine.Pingus            0.816952          0.816952
## 3                    wine.Valbuena 5o            0.771449          0.771449
## 4              winery.Alvaro Palacios            0.769421          0.769421
## 5 wine.L’Ermita Velles Vinyes Priorat            0.620544          0.620544
##   percentage
## 1   0.007521
## 2   0.006145
## 3   0.005802
## 4   0.005787
## 5   0.004667
##
## ---
##                     variable relative_importance scaled_importance percentage
## 980 winery.Alonso & Pedrajo            0.073429          0.073429   0.000552
## 981           winery.Ostatu            0.064309          0.064309   0.000484
## 982        wine.missing(NA)            0.000000          0.000000   0.000000
## 983      winery.missing(NA)            0.000000          0.000000   0.000000
## 984      region.missing(NA)            0.000000          0.000000   0.000000
## 985        type.missing(NA)            0.000000          0.000000   0.000000
```

```
#Coeficiente R^2
h2o.r2(modelo_r8)
```

```
## [1] 0.7131944
```

```
#Capacidad de predicción
predict_test8 = h2o.predict(modelo_r8, newdata = test.hex)
```

```
##   |                                                                      |
```

```
pred8 = as.data.frame(predict_test8)
head(pred8)
```

```
##    predict
## 1 573.2946
## 2 662.1828
## 3 717.1470
## 4 620.0163
## 5 538.0074
## 6 528.7337
```

Modelo 5: Red con dos capas ocultas y 646 nodos en cada una

```
modelo_r9 = h2o.deeplearning(
  x = c(1,2,3,4,5,6,8,9,10), #variables predictoras
  y = 7, #variable respuesta
  training_frame = train.hex, #datos de entrenamiento
  validation_frame = test.hex, #datos de validacion/test
  activation = 'RectifierWithDropout',
  hidden = c(646,646), #tamaño de las capas ocultas
  hidden_dropout_ratio = c(0.5, 0.5),
  input_dropout_ratio = 0.2,
  epochs = 50, #iteraciones
  l1 = 1e-5,
  l2 = 1e-5,
  rho = 0.99,
  epsilon = 1e-8)
```

```
##   |                                                                      |
```

```
summary(modelo_r9)
```

```
## Model Details:
## ==============
##
## H2ORegressionModel: deeplearning
## Model Key:  DeepLearning_model_R_1655376581048_640
## Status of Neuron Layers: predicting price, regression, gaussian distribution, Quadratic loss, 1.055.
##   layer units             type dropout       l1       l2 mean_rate rate_rms
## 1     1   985            Input 20.00 %       NA       NA        NA       NA
## 2     2   646 RectifierDropout 50.00 % 0.000010 0.000010  0.077130 0.116971
## 3     3   646 RectifierDropout 50.00 % 0.000010 0.000010  0.072170 0.122753
## 4     4     1           Linear      NA 0.000010 0.000010  0.001125 0.001447
##   momentum mean_weight weight_rms mean_bias bias_rms
## 1       NA          NA         NA        NA       NA
## 2 0.000000   -0.000202   0.038519  0.293305 0.333873
## 3 0.000000   -0.007781   0.043375  0.917886 0.142385
## 4 0.000000    0.013689   0.038704  0.333240 0.000000
```

```
##
## H2ORegressionMetrics: deeplearning
## ** Reported on training data. **
## ** Metrics reported on full training frame **
##
## MSE:  6133.898
## RMSE:  78.31921
## MAE:  17.86795
## RMSLE:  0.3011813
## Mean Residual Deviance :  6133.898
##
##
## H2ORegressionMetrics: deeplearning
## ** Reported on validation data. **
## ** Metrics reported on full validation frame **
##
## MSE:  2254.371
## RMSE:  47.48021
## MAE:  16.81704
## RMSLE:  0.347031
## Mean Residual Deviance :  2254.371
##
##
##
##
## Scoring History:
##             timestamp          duration training_speed   epochs iterations
## 1 2022-06-16 14:10:32         0.000 sec             NA  0.00000          0
## 2 2022-06-16 14:10:46        17.706 sec   1182 obs/sec  3.86656          1
## 3 2022-06-16 14:11:31  1 min  2.196 sec   1807 obs/sec 23.19981          6
## 4 2022-06-16 14:12:09  1 min 41.786 sec   2025 obs/sec 42.52342         11
## 5 2022-06-16 14:12:28  2 min  0.630 sec   2066 obs/sec 50.25841         13
## 6 2022-06-16 14:12:33  2 min  4.362 sec   2064 obs/sec 50.25841         13
##         samples training_rmse training_deviance training_mae training_r2
## 1      0.000000            NA                NA           NA          NA
## 2  16429.000000     148.44882       22037.05213    104.88763     0.30302
## 3  98576.000000      92.58290        8571.59323     45.09455     0.72890
## 4 180682.000000      78.31921        6133.89846     17.86795     0.80600
## 5 213548.000000      79.91894        6387.03766     30.14154     0.79799
## 6 213548.000000      78.31921        6133.89846     17.86795     0.80600
##   validation_rmse validation_deviance validation_mae validation_r2
## 1              NA                  NA             NA            NA
## 2       131.77844         17365.55737      103.39622       0.00602
## 3        71.13992          5060.88887       44.72735       0.71032
## 4        47.48021          2254.37077       16.81704       0.87096
## 5        50.76789          2577.37903       29.17022       0.85248
## 6        47.48021          2254.37077       16.81704       0.87096
##
## Variable Importances: (Extract with `h2o.varimp`)
## =================================================
##
## Variable Importances:
##                            variable relative_importance scaled_importance
## 1                    wine.Valbuena 5o            1.000000          1.000000
```

```
## 2                          wine.Pingus            0.960417          0.960417
## 3            winery.Dominio de Pingus            0.960208          0.960208
## 4 wine.L'Ermita Velles Vinyes Priorat            0.944771          0.944771
## 5              winery.Alvaro Palacios            0.826865          0.826865
##   percentage
## 1   0.002917
## 2   0.002802
## 3   0.002801
## 4   0.002756
## 5   0.002412
##
## ---
##               variable relative_importance scaled_importance percentage
## 980               body            0.246708          0.246708   0.000720
## 981           acidity             0.225072          0.225072   0.000657
## 982   wine.missing(NA)             0.000000          0.000000   0.000000
## 983 winery.missing(NA)             0.000000          0.000000   0.000000
## 984 region.missing(NA)             0.000000          0.000000   0.000000
## 985   type.missing(NA)             0.000000          0.000000   0.000000
```

```
#Coeficiente R^2
h2o.r2(modelo_r9)
```

```
## [1] 0.8060004
```

```
#Capacidad de predicción
predict_test9 = h2o.predict(modelo_r9, newdata = test.hex)
```

```
##   |                                                                              |
```

```
pred9 = as.data.frame(predict_test9)
head(pred9)
```

```
##    predict
## 1 446.5705
## 2 578.5609
## 3 681.8919
## 4 536.4653
## 5 493.3382
## 6 487.8236
```

Modelo 6: Red con una capa oculta y 646 nodos

```
modelo_r10 = h2o.deeplearning(
  x = c(1,2,3,4,5,6,8,9,10), #variables predictoras
  y = 7, #variable respuesta
  training_frame = train.hex, #datos de entrenamiento
  validation_frame = test.hex, #datos de validacion/test
  activation = 'RectifierWithDropout',
  hidden = 646, #tamaño de las capas ocultas
```

```
  hidden_dropout_ratio = 0.5,
  input_dropout_ratio = 0.2,
  epochs = 50, #iteraciones
  l1 = 1e-5,
  l2 = 1e-5,
  rho = 0.99,
  epsilon = 1e-8)
```

```
##   |                                                                              |
```

```
summary(modelo_r10)
```

```
## Model Details:
## ==============
##
## H2ORegressionModel: deeplearning
## Model Key:  DeepLearning_model_R_1655376581048_641
## Status of Neuron Layers: predicting price, regression, gaussian distribution, Quadratic loss, 637.603
##   layer units             type dropout       l1       l2 mean_rate rate_rms
## 1     1   985            Input 20.00 %       NA       NA        NA       NA
## 2     2   646 RectifierDropout 50.00 % 0.000010 0.000010  0.128543 0.250927
## 3     3     1           Linear      NA 0.000010 0.000010  0.000886 0.001382
##   momentum mean_weight weight_rms mean_bias bias_rms
## 1       NA          NA         NA        NA       NA
## 2 0.000000   -0.001946   0.040877 -0.049348 0.121334
## 3 0.000000    0.003879   0.103668 -0.073644 0.000000
##
## H2ORegressionMetrics: deeplearning
## ** Reported on training data. **
## ** Metrics reported on full training frame **
##
## MSE:  6781.641
## RMSE:  82.35072
## MAE:  23.8623
## RMSLE:  NaN
## Mean Residual Deviance :  6781.641
##
##
## H2ORegressionMetrics: deeplearning
## ** Reported on validation data. **
## ** Metrics reported on full validation frame **
##
## MSE:  3305.347
## RMSE:  57.49214
## MAE:  24.29633
## RMSLE:  NaN
## Mean Residual Deviance :  3305.347
##
##
##
##
## Scoring History:
```

```
##             timestamp   duration training_speed   epochs iterations
## 1 2022-06-16 14:12:40  0.000 sec             NA  0.00000          0
## 2 2022-06-16 14:12:45  6.778 sec   4282 obs/sec  5.00000          1
## 3 2022-06-16 14:13:05 26.817 sec   6325 obs/sec 35.00000          7
## 4 2022-06-16 14:13:14 36.249 sec   6793 obs/sec 50.00000         10
## 5 2022-06-16 14:13:16 37.762 sec   6785 obs/sec 50.00000         10
##         samples training_rmse training_deviance training_mae training_r2
## 1      0.000000            NA                NA           NA          NA
## 2  21245.000000     120.96864       14633.41070     29.97495     0.53718
## 3 148715.000000      82.35072        6781.64108     23.86230     0.78551
## 4 212450.000000      81.11868        6580.24046     30.37342     0.79188
## 5 212450.000000      82.35072        6781.64108     23.86230     0.78551
##   validation_rmse validation_deviance validation_mae validation_r2
## 1              NA                  NA             NA            NA
## 2        87.40656          7639.90702       27.43820       0.56270
## 3        57.49214          3305.34652       24.29633       0.81081
## 4        60.03133          3603.76114       31.31636       0.79373
## 5        57.49214          3305.34652       24.29633       0.81081
##
## Variable Importances: (Extract with `h2o.varimp`)
## =================================================
##
## Variable Importances:
##                              variable relative_importance scaled_importance
## 1                        wine.Pingus            1.000000          1.000000
## 2            winery.Dominio de Pingus            0.958878          0.958878
## 3 wine.L'Ermita Velles Vinyes Priorat            0.867255          0.867255
## 4              winery.Alvaro Palacios            0.780745          0.780745
## 5                    wine.Valbuena 5o            0.778824          0.778824
##   percentage
## 1   0.004150
## 2   0.003980
## 3   0.003599
## 4   0.003240
## 5   0.003232
##
## ---
##                              variable relative_importance scaled_importance
## 980        wine.Belondrade y Lurton            0.208563          0.208563
## 981 wine.Seleccion Especial Verdejo            0.207591          0.207591
## 982                wine.missing(NA)            0.000000          0.000000
## 983              winery.missing(NA)            0.000000          0.000000
## 984              region.missing(NA)            0.000000          0.000000
## 985                type.missing(NA)            0.000000          0.000000
##     percentage
## 980   0.000866
## 981   0.000862
## 982   0.000000
## 983   0.000000
## 984   0.000000
## 985   0.000000
```

```
#Coeficiente R^2
h2o.r2(modelo_r10)
```

```
## [1] 0.785514
```

```
#Capacidad de predicción
predict_test10 = h2o.predict(modelo_r10, newdata = test.hex)
```

```
##   |                                                                      |
```

```
pred10 = as.data.frame(predict_test10)
head(pred10)
```

```
##    predict
## 1 586.0657
## 2 713.1365
## 3 814.7280
## 4 670.7010
## 5 574.9026
## 6 565.6825
```

Ejemplo de regresión - Precio del vino español en Python

Raquel Beltrán Barba

PREDICCIÓN DEL PRECIO DEL VINO ESPAÑOL

Primero cargamos las librerías que vamos a necesitar:

```
import numpy as np
import pandas as pd
import math
import sklearn
import sklearn.preprocessing
import datetime
import os
import matplotlib.pyplot as plt
import tensorflow as tf
import csv
import seaborn as sns
from sklearn.preprocessing import OneHotEncoder
from sklearn.preprocessing import LabelEncoder
```

Lectura y modificación del conjunto de datos

```
#os.chdir("C:\\Users\\raque\\OneDrive\\Escritorio\\TFG\\Bases.de.datos\\vino2")
datos = pd.read_csv('wines_SPA.csv', delimiter = ",")
        #Eliminamos la variable country, ya que toma el valor España siempre
del(datos['country'])
datos['year'] = datos['year'].replace('N.V.', np.NaN)
datos = datos.dropna()
datos['year'] = datos['year'].astype(np.int64)

for col in datos.columns:
    if datos[col].dtype == 'object':
        #print(str(col))
        label = LabelEncoder()
        label = label.fit(datos[col])
        datos[col] = label.transform(datos[col].astype(str))
datos.head()
```

```
##    winery  wine  year  rating  num_reviews  region   price  type  body  acidity
## 0     344   607  2013     4.9           58      58  995.00    19   5.0      3.0
## 1      28   653  2018     4.9           31      61  313.50    18   4.0      2.0
## 2     366   622  2009     4.8         1793      46  324.95    11   5.0      3.0
## 3     366   622  1999     4.8         1705      46  692.96    11   5.0      3.0
## 4     366   622  1996     4.8         1309      46  778.06    11   5.0      3.0
```

Estudio de las variables

```
datos.info()
```

```
## <class 'pandas.core.frame.DataFrame'>
## Int64Index: 6070 entries, 0 to 7499
## Data columns (total 10 columns):
##  #   Column       Non-Null Count  Dtype
## ---  ------       --------------  -----
##  0   winery       6070 non-null   int32
##  1   wine         6070 non-null   int32
##  2   year         6070 non-null   int64
##  3   rating       6070 non-null   float64
##  4   num_reviews  6070 non-null   int64
##  5   region       6070 non-null   int32
##  6   price        6070 non-null   float64
##  7   type         6070 non-null   int32
##  8   body         6070 non-null   float64
##  9   acidity      6070 non-null   float64
## dtypes: float64(4), int32(4), int64(2)
## memory usage: 426.8 KB
```

```
datos.shape
```

```
## (6070, 10)
```

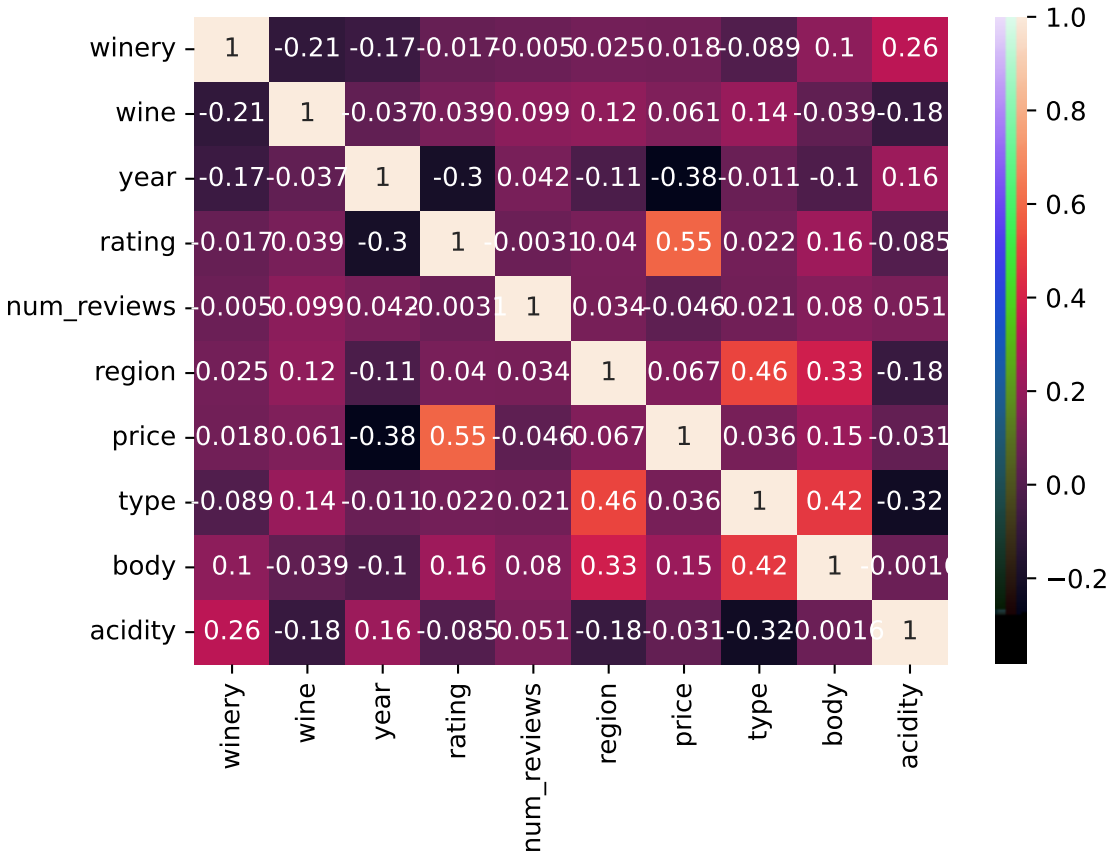
```
#Estudio de las variables categóricas
cat_feat = datos[['winery', 'wine', 'year','region', 'type']]
cat_feat.describe()
```

```
##             winery         wine         year       region         type
## count  6070.000000  6070.000000  6070.000000  6070.000000  6070.000000
## mean    211.177430   373.687479  2012.955189    41.828995    10.988303
## std     103.685401   180.621140     7.145244    14.142791     3.627947
## min       0.000000     0.000000  1910.000000     0.000000     0.000000
## 25%     130.000000   208.000000  2011.000000    43.000000    10.000000
## 50%     214.000000   395.000000  2015.000000    46.000000    11.000000
## 75%     304.000000   531.000000  2017.000000    48.000000    12.000000
## max     394.000000   676.000000  2021.000000    62.000000    20.000000
```

```
#Correlaciones entre las variables
corr_matrix = datos.corr(method='pearson')
corr_matrix
```

```
##                winery      wine      year  ...      type      body   acidity
## winery       1.000000 -0.205883 -0.166178  ... -0.089298  0.104001  0.255655
## wine        -0.205883  1.000000 -0.036589  ...  0.141347 -0.039152 -0.182456
## year        -0.166178 -0.036589  1.000000  ... -0.010680 -0.100687  0.155387
## rating      -0.017197  0.039051 -0.299519  ...  0.022493  0.161058 -0.085140
## num_reviews -0.005016  0.099387  0.042120  ...  0.020928  0.079627  0.050742
## region       0.025006  0.117132 -0.112547  ...  0.463734  0.327776 -0.176325
## price        0.018154  0.060671 -0.384351  ...  0.036404  0.150861 -0.030566
## type        -0.089298  0.141347 -0.010680  ...  1.000000  0.421510 -0.319898
## body         0.104001 -0.039152 -0.100687  ...  0.421510  1.000000 -0.001638
## acidity      0.255655 -0.182456  0.155387  ... -0.319898 -0.001638  1.000000
##
## [10 rows x 10 columns]
```

```
sns.heatmap(corr_matrix, annot=True)
plt.show()
```

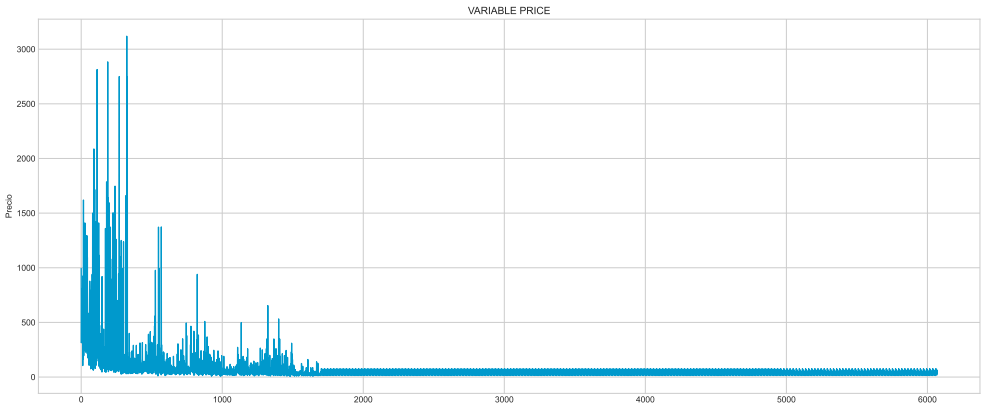


```
    #A traves de la funcion tidy_corr_matrix obtenemos las variables con mayor correlacion
def tidy_corr_matrix(corr_mat):
    '''Función para convertir una matriz de correlación de pandas en formato tidy.'''
    corr_mat = corr_mat.stack().reset_index()
    corr_mat.columns = ['variable_1','variable_2','r']
    corr_mat = corr_mat.loc[corr_mat['variable_1'] != corr_mat['variable_2'], :]
    corr_mat['abs_r'] = np.abs(corr_mat['r'])
    corr_mat = corr_mat.sort_values('abs_r', ascending=False)

    return(corr_mat)

tidy_corr_matrix(corr_matrix).head(10)
```

```
##    variable_1 variable_2         r     abs_r
## 63      price     rating  0.551943  0.551943
## 36     rating      price  0.551943  0.551943
## 75       type     region  0.463734  0.463734
## 57     region       type  0.463734  0.463734
## 87       body       type  0.421510  0.421510
## 78       type       body  0.421510  0.421510
## 26       year      price -0.384351  0.384351
## 62      price       year -0.384351  0.384351
## 58     region       body  0.327776  0.327776
## 85       body     region  0.327776  0.327776
```

```
  #Representación de las variables price y rating
fig, ax = plt.subplots(1, 1, figsize=(6,4))
ax.scatter(x=datos.price, y=datos.rating, alpha= 0.8)
ax.set_xlabel('Precio de la botella de vino')
ax.set_ylabel('Valoración del vino');
```

```
    # Representacion de la variable Amount
PRECIO = datos["price"]
x = range(0,PRECIO.shape[0])
plt.style.use('seaborn-whitegrid')
plt.rcParams["figure.figsize"] = (20,8)
fig = plt.figure()
ax = plt.axes()
ax.plot(x,PRECIO,color='#0099CC')
#plt.xlabel("")
plt.ylabel("Precio")
ax.set_title("VARIABLE PRICE")
```



Partición Entrenamiento/Test

```
X = datos.loc[:, datos.columns != 'price']
y = datos["price"]
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

print('X_train.shape = ',X_train.shape)
```

```
## X_train.shape =  (4249, 9)
```

```
print('y_train.shape = ', y_train.shape)
```

```
## y_train.shape =  (4249,)
```

```
print('X_test.shape = ', X_test.shape)
```

```
## X_test.shape =  (1821, 9)
```

```
print('y_test.shape = ',y_test.shape)
```

```
## y_test.shape =  (1821,)
```

Comenzamos a trabajar con los modelos de Redes Neuronales

```
from sklearn.neural_network import MLPRegressor
from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score
```

Modelo 1: Red con dos capas ocultas de 8 nodos cada una

```
modelor1 = MLPRegressor(hidden_layer_sizes=(8,8),
                learning_rate_init=0.01,
                solver = 'adam',
                max_iter = 1000,
                random_state = 7978)

modelor1.fit(X=X_train, y=y_train)
```

```
## MLPRegressor(hidden_layer_sizes=(8, 8), learning_rate_init=0.01, max_iter=1000,
##              random_state=7978)
```

```
#Capacidad de predicción
predictions = modelor1.predict(X_test)

rmse1 = mean_squared_error(
        y_true = y_test,
        y_pred = predictions,
        squared = False
       )
rmse1
```

```
## 173.92509404735327
```

```
#Calculo del coeficiente de determinación
r2 = r2_score(y_test, predictions)
print(r2)
```

```
## 0.20151012395131318
```

Modelo 2: Red con una capa oculta y 15 nodos

```
modelor6 = MLPRegressor(hidden_layer_sizes=(15),
                learning_rate_init=0.01,
                solver = 'adam',
                max_iter = 1000,
                random_state = 7978)

modelor6.fit(X=X_train, y=y_train)
```

```
## MLPRegressor(hidden_layer_sizes=15, learning_rate_init=0.01, max_iter=1000,
##              random_state=7978)
```

```
#Capacidad de predicción
predictions6 = modelor6.predict(X_test)

rmse6 = mean_squared_error(
        y_true = y_test,
        y_pred = predictions6,
        squared = False)
rmse6
```

```
## 172.67282617709498
```

```
#Calculo del coeficiente de determinación
r2_6 = r2_score(y_test, predictions6)
print(r2_6)
```

```
## 0.2129670500228832
```

Modelo 3: Red con una capa oculta y 60 nodos

```
modelor7 = MLPRegressor(hidden_layer_sizes=(60),
                learning_rate_init=0.01,
                solver = 'adam',
                max_iter = 1000,
                random_state = 7978)

modelor7.fit(X=X_train, y=y_train)
```

```
## MLPRegressor(hidden_layer_sizes=60, learning_rate_init=0.01, max_iter=1000,
##              random_state=7978)
```

```
#Capacidad de predicción
predictions7 = modelor7.predict(X_test)

rmse7 = mean_squared_error(
        y_true = y_test,
        y_pred = predictions7,
        squared = False)
rmse7
```

```
## 168.26769170429267
```

```
#Calculo del coeficiente de determinación
r2_7 = r2_score(y_test, predictions7)
print(r2_7)
```

```
## 0.2526115283371466
```

Modelo 4: Red con tres capas ocultas y 30 nodos en cada una

```
modelor8 = MLPRegressor(hidden_layer_sizes=(30,30,30),
                learning_rate_init=0.01,
                solver = 'adam',
                max_iter = 1000,
                random_state = 7978)

modelor8.fit(X=X_train, y=y_train)
```

```
## MLPRegressor(hidden_layer_sizes=(30, 30, 30), learning_rate_init=0.01,
##              max_iter=1000, random_state=7978)
```

```
#Capacidad de predicción
predictions8 = modelor8.predict(X_test)

rmse8 = mean_squared_error(
        y_true = y_test,
        y_pred = predictions8,
        squared = False)
rmse8
```

```
## 165.83794762305337
```

```
#Calculo del coeficiente de determinación
r2_8 = r2_score(y_test, predictions8)
print(r2_8)
```

```
## 0.2740399043928258
```

Modelo 5: Red con dos capas ocultas y 646 nodos en cada una

```
modelor9 = MLPRegressor(hidden_layer_sizes=(646,646),
                learning_rate_init=0.01,
                solver = 'adam',
                max_iter = 1000,
                random_state = 7978)

modelor9.fit(X=X_train, y=y_train)
```

```
## MLPRegressor(hidden_layer_sizes=(646, 646), learning_rate_init=0.01,
##              max_iter=1000, random_state=7978)
```

```
#Capacidad de predicción
predictions9 = modelor9.predict(X_test)

rmse9 = mean_squared_error(
        y_true = y_test,
        y_pred = predictions9,
        squared = False)
rmse9
```

```
## 171.8045659845118
```

```
#Calculo del coeficiente de determinación
r2_9 = r2_score(y_test, predictions9)
print(r2_9)
```

```
## 0.22086211184958515
```

Modelo 6: Red con una capa oculta y 646 nodos

```
modelor10 = MLPRegressor(hidden_layer_sizes=(646),
                learning_rate_init=0.01,
                solver = 'adam',
                max_iter = 1000,
                random_state = 7978)

modelor10.fit(X=X_train, y=y_train)
```

```
## MLPRegressor(hidden_layer_sizes=646, learning_rate_init=0.01, max_iter=1000,
##              random_state=7978)
```

```
#Capacidad de predicción
predictions10 = modelor10.predict(X_test)

rmse10 = mean_squared_error(
        y_true = y_test,
        y_pred = predictions10,
        squared = False)
rmse10
```

```
## 172.7361514220133
```

```
#Calculo del coeficiente de determinación
r2_10 = r2_score(y_test, predictions10)
print(r2_10)
```

```
## 0.2123896784202578
```

Ejemplo de clasificación de imágenes - Clasificación de prendas de ropa en RStudio

Raquel Beltrán Barba

CLASIFICACIÓN DE IMÁGENES DE PRENDAS DE ROPA

Primero cargamos las librerías que vamos a necesitar:

```
set.seed(7978)
library(h2o)
library(pixmap)
```

Lectura y modificación del conjunto de datos

```
load("Prendas.RData")
class_names = c('T-shirt/top', 'Trouser', 'Pullover', 'Dress', 'Coat',
                'Sandal', 'Shirt', 'Sneaker', 'Bag', 'Ankle boot')
dim(train_images) #imagenes del conjunto de entrenamiento
```

```
## [1] 60000    28    28
```

```
dim(train_labels) #clases del conjunto de entrenamiento
```

```
## [1] 60000
```

```
dim(test_images) #imagenes del conjunto test
```

```
## [1] 10000    28    28
```

```
dim(test_labels) #clases del conjunto test
```

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

Veamos la cantidad de imágenes que hay de cada prenda en cada conjunto de datos:

```
table(train_labels)
```

```
## train_labels
##    0    1    2    3    4    5    6    7    8    9
## 6000 6000 6000 6000 6000 6000 6000 6000 6000 6000
```

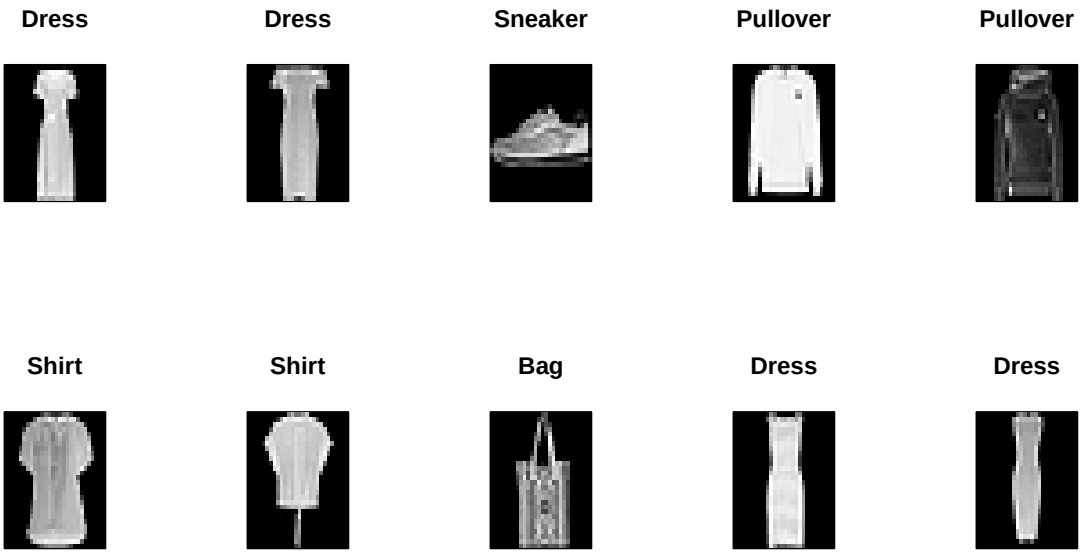
```
table(test_labels)
```

```
## test_labels
##    0    1    2    3    4    5    6    7    8    9
## 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000
```

Tenemos los píxeles tomando valores del 0 al 255. Ajustemos bien los valores:

```
train_images = train_images / 255
test_images = test_images / 255
```

```
#Dibujamos 10 prendas
par(mfrow=c(2,5))
for (i in 51:60) {
  img <- train_images[i, , ]
  #para que no salga "tumbado"
  img <- t(apply(img, 2, rev))
  image(1:28, 1:28, img, col = gray((0:255)/255),
        xlab="",ylab="",
        xaxt = 'n', yaxt = 'n',
        main = paste(class_names[train_labels[i] + 1]))
}
```




```
par(mfcol=c(1,1))
```

Preparamos las matrices de entrenamiento y test:

```
dimentrain = dim(train_images)
dimentest = dim(test_images)
X_train = matrix(train_images, dimentrain[1], prod(dimentrain[2:3]))
X_test = matrix(test_images, dimentest[1], prod(dimentest[2:3]))

datos_ent = data.frame(Y=factor(train_labels), X_train)
datos_test = data.frame(Y=factor(test_labels), X_test)
```

Comenzamos a trabajar con el modelo H2O

```
  #Inicializacion del servicio h2o
localH2O = h2o.init(nthreads = -1)
```

```
##  Connection successful!
##
## R is connected to the H2O cluster:
##     H2O cluster uptime:         1 hours 4 minutes
##     H2O cluster timezone:       Europe/Madrid
##     H2O data parsing timezone:  UTC
##     H2O cluster version:        3.36.0.4
##     H2O cluster version age:    2 months and 17 days
##     H2O cluster name:           H2O_started_from_R_raque_qzy359
##     H2O cluster total nodes:    1
##     H2O cluster total memory:   3.58 GB
##     H2O cluster total cores:    8
##     H2O cluster allowed cores:  8
##     H2O cluster healthy:        TRUE
##     H2O Connection ip:          localhost
##     H2O Connection port:        54321
##     H2O Connection proxy:       NA
##     H2O Internal Security:      FALSE
##     R Version:                  R version 4.2.0 (2022-04-22 ucrt)
```

```
  #Convertir los data.frames en objetos que puedan ser procesados
train.hex = as.h2o(datos_ent)
```

```
##   |                                                                      |
```

```
test.hex = as.h2o(datos_test)
```

```
##   |                                                                      |
```

A continuación creo y estudio los distintos modelos propuestos:

Modelo 1: Red con dos capas ocultas de 100 nodos cada una

```
modeloci1 = h2o.deeplearning(
  x = 2:785, #variables predictoras
  y = 1, #variable respuesta
  training_frame = train.hex, #datos de entrenamiento
  validation_frame = test.hex, #datos de validacion/test
  distribution="multinomial",
  activation = 'RectifierWithDropout',
  hidden = c(100,100), #tamaño de las capas ocultas
  hidden_dropout_ratio = c(0.5, 0.5),
  input_dropout_ratio = 0.2,
  epochs = 50, #iteraciones
  l1 = 1e-5,
  l2 = 1e-5,
  rho = 0.99,
  epsilon = 1e-8,
  train_samples_per_iteration = 1000)
```

```
##   |                                                                              |
```

```
summary(modeloci1)
```

```
## Model Details:
## ==============
##
## H2OMultinomialModel: deeplearning
## Model Key:  DeepLearning_model_R_1655405240903_3
## Status of Neuron Layers: predicting Y, 10-class classification, multinomial distribution, CrossEntro
##   layer units             type dropout       l1       l2 mean_rate rate_rms
## 1     1   784            Input 20.00 %       NA       NA        NA       NA
## 2     2   100 RectifierDropout 50.00 % 0.000010 0.000010  0.005689 0.016441
## 3     3   100 RectifierDropout 50.00 % 0.000010 0.000010  0.000393 0.000244
## 4     4    10          Softmax      NA 0.000010 0.000010  0.013507 0.049712
##   momentum mean_weight weight_rms mean_bias bias_rms
## 1       NA          NA         NA        NA       NA
## 2 0.000000    0.005050   0.105289 -0.942319 0.357806
## 3 0.000000   -0.042390   0.094660  0.532490 0.167582
## 4 0.000000   -0.803013   0.840348 -4.059170 1.121022
##
## H2OMultinomialMetrics: deeplearning
## ** Reported on training data. **
## ** Metrics reported on temporary training frame with 9926 samples **
##
## Training Set Metrics:
## =====================
##
## MSE: (Extract with `h2o.mse`) 0.08827516
## RMSE: (Extract with `h2o.rmse`) 0.2971114
## Logloss: (Extract with `h2o.logloss`) 0.2761827
## Mean Per-Class Error: 0.1022634
## AUC: (Extract with `h2o.auc`) NaN
## AUCPR: (Extract with `h2o.aucpr`) NaN
## Confusion Matrix: Extract with `h2o.confusionMatrix(<model>,train = TRUE)`)
```

```
## ========================================================================
## Confusion Matrix: Row labels: Actual class; Column labels: Predicted class
##           0   1   2   3    4   5    6    7   8   9  Error            Rate
## 0       846   0   5  20    4   0  100    0   9   0 0.1402 =     138 / 984
## 1         5 980   1  12    4   0    1    0   0   0 0.0229 =    23 / 1.003
## 2        11   0 792   6  101   0   76    0   7   0 0.2024 =     201 / 993
## 3        40   1   3 894   52   0   26    0   3   0 0.1227 =   125 / 1.019
## 4         2   0  59  16  871   0   53    0   3   0 0.1325 =   133 / 1.004
## 5         0   0   0   0    0 946    0   15   0   5 0.0207 =      20 / 966
## 6       133   0  54  13   65   0  755    0   6   0 0.2641 =   271 / 1.026
## 7         0   0   0   0    0  16    0  956   5  20 0.0411 =      41 / 997
## 8         1   0   0   4    3   0    9    0 936   0 0.0178 =      17 / 953
## 9         0   1   0   0    0  17    0   39   0 924 0.0581 =      57 / 981
## Totals 1038 982 914 965 1100 979 1020 1010 969 949 0.1034 = 1.026 / 9.926
##
## Hit Ratio Table: Extract with `h2o.hit_ratio_table(<model>,train = TRUE)`
## ========================================================================
## Top-10 Hit Ratios:
##     k hit_ratio
## 1   1  0.896635
## 2   2  0.971288
## 3   3  0.990832
## 4   4  0.995467
## 5   5  0.998287
## 6   6  0.999496
## 7   7  0.999799
## 8   8  0.999899
## 9   9  0.999899
## 10 10  1.000000
##
##
##
##
## H2OMultinomialMetrics: deeplearning
## ** Reported on validation data. **
## ** Metrics reported on full validation frame **
##
## Validation Set Metrics:
## =====================
##
## Extract validation frame with `h2o.getFrame("datos_test_sid_bfc4_3")`
## MSE: (Extract with `h2o.mse`) 0.1046617
## RMSE: (Extract with `h2o.rmse`) 0.3235146
## Logloss: (Extract with `h2o.logloss`) 0.3518885
## Mean Per-Class Error: 0.1242
## AUC: (Extract with `h2o.auc`) NaN
## AUCPR: (Extract with `h2o.aucpr`) NaN
## Confusion Matrix: Extract with `h2o.confusionMatrix(<model>,valid = TRUE)`)
## ========================================================================
## Confusion Matrix: Row labels: Actual class; Column labels: Predicted class
##           0   1   2   3    4    5   6    7    8   9  Error            Rate
## 0       849   0   4  23    6    0 106    0   12   0 0.1510 =    151 / 1.000
## 1         5 960   2  26    4    0   2    0    1   0 0.0400 =     40 / 1.000
## 2        19   0 748  12  129    0  90    0    2   0 0.2520 =    252 / 1.000
```

```
## 3        31   5   2 879   44    0  33    0    6   0 0.1210 =    121 / 1.000
## 4         1   1  69  20  847    0  58    0    4   0 0.1530 =    153 / 1.000
## 5         0   0   0   1    0  960   0   28    2   9 0.0400 =     40 / 1.000
## 6       141   0  76  26   78    0 663    0   16   0 0.3370 =    337 / 1.000
## 7         0   0   0   0    0   21   0  953    0  26 0.0470 =     47 / 1.000
## 8         0   0   1   6    5    3  13    5  967   0 0.0330 =     33 / 1.000
## 9         0   0   0   0    0   16   1   50    1 932 0.0680 =     68 / 1.000
## Totals 1046 966 902 993 1113 1000 966 1036 1011 967 0.1242 = 1.242 / 10.000
##
## Hit Ratio Table: Extract with `h2o.hit_ratio_table(<model>,valid = TRUE)`
## =======================================================================
## Top-10 Hit Ratios:
##     k hit_ratio
## 1   1  0.875800
## 2   2  0.961700
## 3   3  0.987700
## 4   4  0.994400
## 5   5  0.996700
## 6   6  0.998900
## 7   7  0.999400
## 8   8  0.999500
## 9   9  1.000000
## 10 10  1.000000
##
##
##
##
##
##
## Scoring History:
##             timestamp   duration training_speed  epochs iterations
## 1 2022-06-16 21:51:36  0.000 sec             NA 0.00000          0
## 2 2022-06-16 21:51:37  1.933 sec   3917 obs/sec 0.01672          1
## 3 2022-06-16 21:51:47 11.762 sec   5880 obs/sec 0.91427         55
## 4 2022-06-16 21:51:55 19.463 sec   5906 obs/sec 1.59677         96
## 5 2022-06-16 21:52:03 27.607 sec   5945 obs/sec 2.33145        140
##         samples training_rmse training_logloss training_r2
## 1      0.000000            NA               NA          NA
## 2   1003.000000       0.51049          1.02025     0.96818
## 3  54856.000000       0.38084          0.45094     0.98229
## 4  95806.000000       0.37462          0.43306     0.98286
## 5 139887.000000       0.35677          0.40163     0.98446
##   training_classification_error training_auc training_pr_auc validation_rmse
## 1                            NA           NA              NA             NA
## 2                       0.29770           NA              NA        0.51249
## 3                       0.16180           NA              NA        0.39188
## 4                       0.15948           NA              NA        0.38618
## 5                       0.15112           NA              NA        0.36981
##   validation_logloss validation_r2 validation_classification_error
## 1                 NA            NA                             NA
## 2            1.03580       0.96816                        0.30160
## 3            0.48458       0.98139                        0.17510
## 4            0.46475       0.98192                        0.17270
## 5            0.44013       0.98342                        0.15810
```

```
##   validation_auc validation_pr_auc
## 1             NA                NA
## 2             NA                NA
## 3             NA                NA
## 4             NA                NA
## 5             NA                NA
##
## ---
##              timestamp          duration training_speed   epochs iterations
## 55 2022-06-16 21:58:26  6 min 50.315 sec   7481 obs/sec 45.93960       2756
## 56 2022-06-16 21:58:33  6 min 57.544 sec   7504 obs/sec 46.89355       2813
## 57 2022-06-16 21:58:41  7 min  5.716 sec   7519 obs/sec 47.85420       2870
## 58 2022-06-16 21:58:53  7 min 17.984 sec   7543 obs/sec 49.44197       2965
## 59 2022-06-16 21:58:59  7 min 23.970 sec   7531 obs/sec 50.01042       2999
## 60 2022-06-16 21:59:00  7 min 24.868 sec   7530 obs/sec 50.01042       2999
##           samples training_rmse training_logloss training_r2
## 55 2756376.000000       0.29711          0.27618     0.98922
## 56 2813613.000000       0.29694          0.27512     0.98923
## 57 2871252.000000       0.30054          0.28416     0.98897
## 58 2966518.000000       0.29802          0.27817     0.98915
## 59 3000625.000000       0.29917          0.27917     0.98907
## 60 3000625.000000       0.29711          0.27618     0.98922
##    training_classification_error training_auc training_pr_auc validation_rmse
## 55                       0.10336           NA              NA         0.32351
## 56                       0.10397           NA              NA         0.32357
## 57                       0.10689           NA              NA         0.32680
## 58                       0.10558           NA              NA         0.32385
## 59                       0.10699           NA              NA         0.32459
## 60                       0.10336           NA              NA         0.32351
##    validation_logloss validation_r2 validation_classification_error
## 55            0.35189       0.98731                          0.12420
## 56            0.35236       0.98731                          0.12410
## 57            0.35948       0.98705                          0.12790
## 58            0.35324       0.98729                          0.12660
## 59            0.35618       0.98723                          0.12410
## 60            0.35189       0.98731                          0.12420
##    validation_auc validation_pr_auc
## 55             NA                NA
## 56             NA                NA
## 57             NA                NA
## 58             NA                NA
## 59             NA                NA
## 60             NA                NA
##
## Variable Importances: (Extract with `h2o.varimp`)
## =================================================
##
## Variable Importances:
##   variable relative_importance scaled_importance percentage
## 1      X10            1.000000          1.000000   0.003296
## 2       X7            0.945543          0.945543   0.003117
## 3      X11            0.939218          0.939218   0.003096
## 4     X393            0.928550          0.928550   0.003061
## 5     X420            0.920275          0.920275   0.003033
```

```
##
## ---
##     variable relative_importance scaled_importance percentage
## 779     X754            0.212351          0.212351   0.000700
## 780     X752            0.208398          0.208398   0.000687
## 781     X753            0.205195          0.205195   0.000676
## 782       X3            0.191665          0.191665   0.000632
## 783       X2            0.182903          0.182903   0.000603
## 784       X1            0.044675          0.044675   0.000147
```

```
#Veamos la capacidad de prediccion del modelo
predict_test = h2o.predict(modeloci1, newdata = test.hex)
```

```
##   |                                                                              |
```

```
pred = as.data.frame(predict_test)
head(pred)
```

```
##   predict           p0           p1           p2           p3           p4
## 1       9 6.533477e-13 5.631926e-08 2.491224e-12 1.438972e-11 8.536326e-14
## 2       2 7.957988e-09 2.287007e-12 9.837590e-01 4.121921e-09 2.664519e-03
## 3       1 3.599128e-17 1.000000e+00 1.580581e-17 1.666026e-14 3.299421e-21
## 4       1 5.282717e-17 1.000000e+00 6.155320e-18 3.987692e-12 1.897297e-20
## 5       6 1.513993e-01 1.333160e-06 2.326269e-02 6.590632e-04 4.559894e-03
## 6       1 1.662744e-12 1.000000e+00 4.660663e-13 6.862070e-12 6.009946e-16
##             p5           p6           p7           p8           p9
## 1 1.950509e-02 1.395939e-11 8.242404e-02 1.412731e-08 8.980708e-01
## 2 6.959787e-11 1.357650e-02 2.894066e-16 6.068196e-13 6.776240e-15
## 3 4.457402e-29 4.853807e-19 1.054202e-26 2.643065e-30 4.356704e-34
## 4 1.205489e-26 1.250632e-18 5.030017e-25 1.157239e-29 6.658220e-30
## 5 4.801637e-06 8.190816e-01 1.550704e-07 1.029735e-03 1.402954e-06
## 6 3.465253e-23 2.992207e-14 1.311359e-21 1.563813e-23 2.207320e-26
```

```
tabla=table(Obs=datos_test[,1], Pred=pred[,1])
aciertos=100*diag(prop.table(tabla,1))
acierto=100*sum(diag(tabla))/sum(tabla)
cat("Acierto test=",acierto,"% \n")
```

```
## Acierto test= 87.58 %
```

Modelo 2: Red con una capa oculta de 100 nodos

```
modeloci2 = h2o.deeplearning(
  x = 2:785, #variables predictoras
  y = 1, #variable respuesta
  training_frame = train.hex, #datos de entrenamiento
  validation_frame = test.hex, #datos de validacion/test
  distribution="multinomial",
  activation = 'RectifierWithDropout',
  hidden = 100, #tamaño de las capas ocultas
```

```
  hidden_dropout_ratio = 0.5,
  input_dropout_ratio = 0.2,
  epochs = 50, #iteraciones
  l1 = 1e-5,
  l2 = 1e-5,
  rho = 0.99,
  epsilon = 1e-8,
  train_samples_per_iteration = 1000)
```

```
##   |                                                                              |
```

```
summary(modeloci2)
```

```
## Model Details:
## ==============
##
## H2OMultinomialModel: deeplearning
## Model Key:  DeepLearning_model_R_1655405240903_4
## Status of Neuron Layers: predicting Y, 10-class classification, multinomial distribution, CrossEntro
##   layer units             type dropout       l1       l2 mean_rate rate_rms
## 1     1   784            Input 20.00 %       NA       NA        NA       NA
## 2     2   100 RectifierDropout 50.00 % 0.000010 0.000010  0.003981 0.014435
## 3     3    10          Softmax      NA 0.000010 0.000010  0.002567 0.015898
##   momentum mean_weight weight_rms mean_bias bias_rms
## 1       NA          NA         NA        NA       NA
## 2 0.000000    0.019458   0.103411 -0.613521 0.296499
## 3 0.000000   -0.158774   0.435821 -0.444324 0.766678
##
## H2OMultinomialMetrics: deeplearning
## ** Reported on training data. **
## ** Metrics reported on temporary training frame with 9934 samples **
##
## Training Set Metrics:
## =====================
##
## MSE: (Extract with `h2o.mse`) 0.0888415
## RMSE: (Extract with `h2o.rmse`) 0.2980629
## Logloss: (Extract with `h2o.logloss`) 0.365476
## Mean Per-Class Error: 0.1064714
## AUC: (Extract with `h2o.auc`) NaN
## AUCPR: (Extract with `h2o.aucpr`) NaN
## Confusion Matrix: Extract with `h2o.confusionMatrix(<model>,train = TRUE)`)
## =========================================================================
## Confusion Matrix: Row labels: Actual class; Column labels: Predicted class
##           0   1   2   3    4   5   6    7    8   9  Error            Rate
## 0       838   1   9  24    4   1  83    0   11   0 0.1370 =     133 / 971
## 1         1 914   1  11    2   0   2    0    1   0 0.0193 =      18 / 932
## 2        16   1 825   4  124   0  46    0    7   0 0.1935 =   198 / 1.023
## 3        17   3   3 840   38   0  14    0    1   0 0.0830 =      76 / 916
## 4         1   1  58  17  907   0  47    0    3   0 0.1228 =   127 / 1.034
## 5         0   0   0   0    0 933   0   26    3  12 0.0421 =      41 / 974
## 6       161   1  82  13  104   0 673    0   11   0 0.3560 =   372 / 1.045
```

```
## 7         0   0   0   0    0  15   0  959    2  27 0.0439 =    44 / 1.003
## 8         2   0   1   3    6   1  12    2 1016   1 0.0268 =    28 / 1.044
## 9         0   0   0   0    0   9   0   28    3 952 0.0403 =      40 / 992
## Totals 1036 921 979 912 1185 959 877 1015 1058 992 0.1084 = 1.077 / 9.934
##
## Hit Ratio Table: Extract with `h2o.hit_ratio_table(<model>,train = TRUE)`
## =======================================================================
## Top-10 Hit Ratios:
##     k hit_ratio
## 1   1  0.891584
## 2   2  0.972418
## 3   3  0.991846
## 4   4  0.996275
## 5   5  0.998591
## 6   6  0.999799
## 7   7  1.000000
## 8   8  1.000000
## 9   9  1.000000
## 10 10  1.000000
##
##
##
##
## H2OMultinomialMetrics: deeplearning
## ** Reported on validation data. **
## ** Metrics reported on full validation frame **
##
## Validation Set Metrics:
## =====================
##
## Extract validation frame with `h2o.getFrame("datos_test_sid_bfc4_3")`
## MSE: (Extract with `h2o.mse`) 0.1061318
## RMSE: (Extract with `h2o.rmse`) 0.3257787
## Logloss: (Extract with `h2o.logloss`) 0.4877739
## Mean Per-Class Error: 0.1277
## AUC: (Extract with `h2o.auc`) NaN
## AUCPR: (Extract with `h2o.aucpr`) NaN
## Confusion Matrix: Extract with `h2o.confusionMatrix(<model>,valid = TRUE)`)
## =========================================================================
## Confusion Matrix: Row labels: Actual class; Column labels: Predicted class
##           0   1   2   3    4   5   6    7    8    9  Error             Rate
## 0       859   1  11  31    1   0  86    0   11    0 0.1410 =    141 / 1.000
## 1         4 964   1  24    3   0   4    0    0    0 0.0360 =     36 / 1.000
## 2        23   0 765   8  138   1  61    0    4    0 0.2350 =    235 / 1.000
## 3        21   6  10 875   46   1  34    0    7    0 0.1250 =    125 / 1.000
## 4         0   0  71  20  854   0  51    0    4    0 0.1460 =    146 / 1.000
## 5         0   0   0   0    0 940   0   32    2   26 0.0600 =     60 / 1.000
## 6       161   1  97  29   97   0 598    0   17    0 0.4020 =    402 / 1.000
## 7         0   0   0   0    0  17   0  953    1   29 0.0470 =     47 / 1.000
## 8         2   0   3   3    6   4  11    4  967    0 0.0330 =     33 / 1.000
## 9         0   0   0   0    0  10   1   41    0  948 0.0520 =     52 / 1.000
## Totals 1070 972 958 990 1145 973 846 1030 1013 1003 0.1277 = 1.277 / 10.000
##
## Hit Ratio Table: Extract with `h2o.hit_ratio_table(<model>,valid = TRUE)`
```

```
## ================================================================
## Top-10 Hit Ratios:
##     k hit_ratio
## 1   1  0.872300
## 2   2  0.962900
## 3   3  0.988300
## 4   4  0.994300
## 5   5  0.996700
## 6   6  0.998900
## 7   7  0.999600
## 8   8  0.999800
## 9   9  0.999800
## 10 10  1.000000
##
##
##
##
##
##
## Scoring History:
##             timestamp   duration training_speed  epochs iterations
## 1 2022-06-16 21:59:04  0.000 sec             NA 0.00000          0
## 2 2022-06-16 21:59:04  1.600 sec   5326 obs/sec 0.01660          1
## 3 2022-06-16 21:59:12  9.722 sec   6479 obs/sec 0.82802         50
## 4 2022-06-16 21:59:19 16.136 sec   6757 obs/sec 1.51555         91
## 5 2022-06-16 21:59:25 22.425 sec   6910 obs/sec 2.19795        132
##         samples training_rmse training_logloss training_r2
## 1      0.000000            NA               NA          NA
## 2    996.000000       0.50238          1.29162     0.96924
## 3  49681.000000       0.35960          0.57171     0.98424
## 4  90933.000000       0.34622          0.51983     0.98539
## 5 131877.000000       0.33992          0.49333     0.98592
##   training_classification_error training_auc training_pr_auc validation_rmse
## 1                            NA           NA              NA              NA
## 2                       0.28679           NA              NA         0.50812
## 3                       0.15673           NA              NA         0.37064
## 4                       0.14536           NA              NA         0.36503
## 5                       0.14073           NA              NA         0.35646
##   validation_logloss validation_r2 validation_classification_error
## 1                 NA            NA                              NA
## 2            1.33320       0.96870                         0.29300
## 3            0.62872       0.98335                         0.16370
## 4            0.59503       0.98385                         0.16140
## 5            0.55343       0.98460                         0.15290
##   validation_auc validation_pr_auc
## 1             NA                NA
## 2             NA                NA
## 3             NA                NA
## 4             NA                NA
## 5             NA                NA
##
## ---
##              timestamp          duration training_speed   epochs iterations
## 23 2022-06-16 22:01:26  2 min 23.859 sec   7737 obs/sec 16.55967        993
```

```
## 24 2022-06-16 22:01:34  2 min 31.341 sec   7730 obs/sec 17.42373       1045
## 25 2022-06-16 22:01:41  2 min 38.049 sec   7728 obs/sec 18.19172       1091
## 26 2022-06-16 22:01:48  2 min 45.299 sec   7714 obs/sec 19.00263       1140
## 27 2022-06-16 22:01:55  2 min 52.231 sec   7707 obs/sec 19.78487       1187
## 28 2022-06-16 22:02:02  2 min 59.403 sec   7693 obs/sec 20.56895       1234
##           samples training_rmse training_logloss training_r2
## 23  993580.000000       0.30006          0.37638     0.98903
## 24 1045424.000000       0.30058          0.38186     0.98899
## 25 1091503.000000       0.29903          0.36926     0.98910
## 26 1140158.000000       0.29931          0.38207     0.98908
## 27 1187092.000000       0.29683          0.36773     0.98926
## 28 1234137.000000       0.29806          0.36548     0.98917
##    training_classification_error training_auc training_pr_auc validation_rmse
## 23                       0.11003           NA              NA         0.32969
## 24                       0.11013           NA              NA         0.32753
## 25                       0.10721           NA              NA         0.32893
## 26                       0.10892           NA              NA         0.32635
## 27                       0.10701           NA              NA         0.32573
## 28                       0.10842           NA              NA         0.32578
##    validation_logloss validation_r2 validation_classification_error
## 23            0.49549       0.98683                          0.13100
## 24            0.49497       0.98700                          0.12950
## 25            0.49882       0.98689                          0.12880
## 26            0.50072       0.98709                          0.12760
## 27            0.49585       0.98714                          0.12660
## 28            0.48777       0.98714                          0.12770
##    validation_auc validation_pr_auc
## 23             NA                NA
## 24             NA                NA
## 25             NA                NA
## 26             NA                NA
## 27             NA                NA
## 28             NA                NA
##
## Variable Importances: (Extract with `h2o.varimp`)
## =================================================
##
## Variable Importances:
##   variable relative_importance scaled_importance percentage
## 1     X757            1.000000          1.000000   0.005490
## 2       X1            0.993509          0.993509   0.005455
## 3       X2            0.955436          0.955436   0.005246
## 4      X10            0.776039          0.776039   0.004261
## 5       X9            0.766070          0.766070   0.004206
##
## ---
##     variable relative_importance scaled_importance percentage
## 779     X297            0.128928          0.128928   0.000708
## 780     X754            0.122951          0.122951   0.000675
## 781     X618            0.120283          0.120283   0.000660
## 782      X87            0.113576          0.113576   0.000624
## 783     X114            0.112173          0.112173   0.000616
## 784     X142            0.111147          0.111147   0.000610
```

```
#Veamos la capacidad de prediccion del modelo
predict_test2 = h2o.predict(modeloci2, newdata = test.hex)
```

```
##   |                                                                      |
```

```
pred2 = as.data.frame(predict_test2)
head(pred2,4)
```

```
##   predict           p0           p1           p2           p3           p4
## 1       9 1.157963e-17 3.537491e-19 3.725923e-20 6.696815e-17 3.860077e-18
## 2       2 1.687550e-09 2.330673e-21 9.996998e-01 5.472725e-19 2.681854e-04
## 3       1 1.801652e-21 1.000000e+00 1.817907e-26 6.373260e-27 6.796131e-29
## 4       1 1.738659e-21 1.000000e+00 1.102648e-24 2.313187e-19 1.214712e-26
##             p5           p6           p7           p8           p9
## 1 2.709787e-02 1.630179e-15 1.889634e-02 4.029542e-07 9.540054e-01
## 2 2.735577e-21 3.200418e-05 1.360418e-33 1.396326e-13 4.936233e-28
## 3 1.192100e-42 9.129306e-31 1.144433e-40 2.194691e-24 1.176611e-43
## 4 1.162717e-36 4.324537e-26 3.026459e-38 3.093612e-28 3.336533e-37
```

```
tabla2 = table(Obs=datos_test[,1], Pred=pred2[,1])
aciertos2 = 100*diag(prop.table(tabla2,1))
acierto2 = 100*sum(diag(tabla2))/sum(tabla2)
cat("Acierto test=", acierto2, "% \n")
```

```
## Acierto test= 87.23 %
```

Modelo 3: Red con una capa oculta de 530 nodos

```
modeloci3 = h2o.deeplearning(
  x = 2:785, #variables predictoras
  y = 1, #variable respuesta
  training_frame = train.hex, #datos de entrenamiento
  validation_frame = test.hex, #datos de validacion/test
  distribution="multinomial",
  activation = 'RectifierWithDropout',
  hidden = 530, #tamaño de las capas ocultas
  hidden_dropout_ratio = 0.5,
  input_dropout_ratio = 0.2,
  epochs = 50, #iteraciones
  l1 = 1e-5,
  l2 = 1e-5,
  rho = 0.99,
  epsilon = 1e-8,
  train_samples_per_iteration = 1000)
```

```
##   |                                                                      |
```

```
summary(modeloci3)
```

```
## Model Details:
## ==============
##
## H2OMultinomialModel: deeplearning
## Model Key:  DeepLearning_model_R_1655405240903_5
## Status of Neuron Layers: predicting Y, 10-class classification, multinomial distribution, CrossEntrop
##   layer units             type dropout       l1       l2 mean_rate rate_rms
## 1     1   784            Input 20.00 %       NA       NA        NA       NA
## 2     2   530 RectifierDropout 50.00 % 0.000010 0.000010  0.009000 0.026806
## 3     3    10          Softmax      NA 0.000010 0.000010  0.006614 0.047154
##   momentum mean_weight weight_rms mean_bias bias_rms
## 1       NA          NA         NA        NA       NA
## 2 0.000000    0.016567   0.082530 -0.173596 0.172776
## 3 0.000000   -0.090682   0.195459 -0.357397 0.419422
##
## H2OMultinomialMetrics: deeplearning
## ** Reported on training data. **
## ** Metrics reported on temporary training frame with 9958 samples **
##
## Training Set Metrics:
## =====================
##
## MSE: (Extract with ‘h2o.mse‘) 0.06541779
## RMSE: (Extract with ‘h2o.rmse‘) 0.255769
## Logloss: (Extract with ‘h2o.logloss‘) 0.2729181
## Mean Per-Class Error: 0.08024638
## AUC: (Extract with ‘h2o.auc‘) NaN
## AUCPR: (Extract with ‘h2o.aucpr‘) NaN
## Confusion Matrix: Extract with ‘h2o.confusionMatrix(<model>,train = TRUE)‘)
## ========================================================================
## Confusion Matrix: Row labels: Actual class; Column labels: Predicted class
##          0    1    2    3    4   5   6   7    8    9  Error          Rate
## 0      864    1   11   20    0   0  81   0    3    0 0.1184 =   116 / 980
## 1        0  993    0   11    0   0   2   0    1    0 0.0139 =  14 / 1.007
## 2        5    0  873    6   79   0  51   0    1    0 0.1399 = 142 / 1.015
## 3        7    6    4  940   23   0  12   0    1    0 0.0534 =    53 / 993
## 4        0    0   65   24  877   0  49   0    2    0 0.1377 = 140 / 1.017
## 5        0    0    0    0    0 954   0  13    2    2 0.0175 =    17 / 971
## 6       91    2   75   13   42   0 753   0    7    0 0.2340 =   230 / 983
## 7        0    0    0    0    0  11   0 927    0   32 0.0443 =    43 / 970
## 8        2    0    0    3    4   0   2   0  995    1 0.0119 =  12 / 1.007
## 9        0    0    0    0    0   6   0  26    0  983 0.0315 =  32 / 1.015
## Totals 969 1002 1028 1017 1025 971 950 966 1012 1018 0.0802 = 799 / 9.958
##
## Hit Ratio Table: Extract with ‘h2o.hit_ratio_table(<model>,train = TRUE)‘
## =======================================================================
## Top-10 Hit Ratios:
##     k hit_ratio
## 1   1  0.919763
## 2   2  0.985740
## 3   3  0.995983
```

```
## 4   4  0.998695
## 5   5  0.999297
## 6   6  0.999799
## 7   7  1.000000
## 8   8  1.000000
## 9   9  1.000000
## 10 10  1.000000
##
##
##
##
## H2OMultinomialMetrics: deeplearning
## ** Reported on validation data. **
## ** Metrics reported on full validation frame **
##
## Validation Set Metrics:
## =====================
##
## Extract validation frame with `h2o.getFrame("datos_test_sid_bfc4_3")`
## MSE: (Extract with `h2o.mse`) 0.09352067
## RMSE: (Extract with `h2o.rmse`) 0.3058115
## Logloss: (Extract with `h2o.logloss`) 0.4617749
## Mean Per-Class Error: 0.1132
## AUC: (Extract with `h2o.auc`) NaN
## AUCPR: (Extract with `h2o.aucpr`) NaN
## Confusion Matrix: Extract with `h2o.confusionMatrix(<model>,valid = TRUE)`)
## =========================================================================
## Confusion Matrix: Row labels: Actual class; Column labels: Predicted class
##          0   1    2   3    4    5   6   7    8    9  Error             Rate
## 0      835   1   12  16    5    1 121   0    9    0 0.1650 =    165 / 1.000
## 1        1 974    1  18    2    0   3   0    1    0 0.0260 =     26 / 1.000
## 2       12   1  825  11   93    1  55   0    2    0 0.1750 =    175 / 1.000
## 3       18   3   12 899   36    1  28   0    3    0 0.1010 =    101 / 1.000
## 4        0   0   91  25  822    0  60   0    2    0 0.1780 =    178 / 1.000
## 5        0   0    0   0    0  964   0  24    1   11 0.0360 =     36 / 1.000
## 6      119   1   96  27   62    0 683   0   12    0 0.3170 =    317 / 1.000
## 7        0   0    0   0    0   15   0 938    1   46 0.0620 =     62 / 1.000
## 8        2   0    4   3    5    5   8   3  970    0 0.0300 =     30 / 1.000
## 9        0   0    0   0    0   16   1  25    0  958 0.0420 =     42 / 1.000
## Totals 987 980 1041 999 1025 1003 959 990 1001 1015 0.1132 = 1.132 / 10.000
##
## Hit Ratio Table: Extract with `h2o.hit_ratio_table(<model>,valid = TRUE)`
## =======================================================================
## Top-10 Hit Ratios:
##     k hit_ratio
## 1   1  0.886800
## 2   2  0.967000
## 3   3  0.988400
## 4   4  0.995800
## 5   5  0.998000
## 6   6  0.999500
## 7   7  0.999600
## 8   8  0.999700
## 9   9  0.999700
```

```
## 10 10  1.000000
##
##
##
##
##
##
## Scoring History:
##             timestamp          duration training_speed  epochs iterations
## 1 2022-06-16 22:02:06         0.000 sec             NA 0.00000          0
## 2 2022-06-16 22:02:08         4.967 sec   1086 obs/sec 0.01542          1
## 3 2022-06-16 22:02:42        39.126 sec   1241 obs/sec 0.66105         40
## 4 2022-06-16 22:03:13  1 min 10.150 sec   1303 obs/sec 1.29878         78
## 5 2022-06-16 22:03:45  1 min 42.351 sec   1340 obs/sec 1.98442        119
##         samples training_rmse training_logloss training_r2
## 1      0.000000            NA               NA          NA
## 2    925.000000       0.49192          1.51693     0.97076
## 3  39663.000000       0.34907          0.64204     0.98528
## 4  77927.000000       0.36614          0.65886     0.98380
## 5 119065.000000       0.31672          0.48225     0.98788
##   training_classification_error training_auc training_pr_auc validation_rmse
## 1                            NA           NA              NA              NA
## 2                       0.27084           NA              NA         0.50379
## 3                       0.14441           NA              NA         0.36923
## 4                       0.16118           NA              NA         0.38716
## 5                       0.12111           NA              NA         0.34350
##   validation_logloss validation_r2 validation_classification_error
## 1                 NA            NA                              NA
## 2            1.59270       0.96924                         0.28600
## 3            0.75022       0.98348                         0.15980
## 4            0.79530       0.98183                         0.18090
## 5            0.61237       0.98570                         0.14250
##   validation_auc validation_pr_auc
## 1             NA                NA
## 2             NA                NA
## 3             NA                NA
## 4             NA                NA
## 5             NA                NA
##
## ---
##             timestamp          duration training_speed   epochs iterations
## 28 2022-06-16 22:15:59 13 min 56.243 sec   1599 obs/sec 19.99365       1203
## 29 2022-06-16 22:16:33 14 min 30.511 sec   1597 obs/sec 20.77952       1250
## 30 2022-06-16 22:17:07 15 min  4.564 sec   1602 obs/sec 21.66733       1303
## 31 2022-06-16 22:17:40 15 min 37.025 sec   1606 obs/sec 22.51250       1354
## 32 2022-06-16 22:18:13 16 min  9.796 sec   1611 obs/sec 23.37468       1406
## 33 2022-06-16 22:18:16 16 min 13.044 sec   1611 obs/sec 23.37468       1406
##          samples training_rmse training_logloss training_r2
## 28 1199619.000000       0.25806          0.30233     0.99195
## 29 1246771.000000       0.25934          0.29907     0.99187
## 30 1300040.000000       0.25426          0.28529     0.99219
## 31 1350750.000000       0.25324          0.27582     0.99225
## 32 1402481.000000       0.25266          0.27346     0.99229
## 33 1402481.000000       0.25577          0.27292     0.99210
```

```
##    training_classification_error training_auc training_pr_auc validation_rmse
## 28                       0.07984           NA              NA         0.30768
## 29                       0.08184           NA              NA         0.31055
## 30                       0.07923           NA              NA         0.30764
## 31                       0.07823           NA              NA         0.30809
## 32                       0.07732           NA              NA         0.30475
## 33                       0.08024           NA              NA         0.30581
##    validation_logloss validation_r2 validation_classification_error
## 28            0.47973       0.98853                          0.11490
## 29            0.49342       0.98831                          0.11670
## 30            0.47417       0.98853                          0.11300
## 31            0.46881       0.98849                          0.11460
## 32            0.47279       0.98874                          0.11130
## 33            0.46177       0.98866                          0.11320
##    validation_auc validation_pr_auc
## 28             NA                NA
## 29             NA                NA
## 30             NA                NA
## 31             NA                NA
## 32             NA                NA
## 33             NA                NA
##
## Variable Importances: (Extract with `h2o.varimp`)
## =================================================
##
## Variable Importances:
##   variable relative_importance scaled_importance percentage
## 1      X10            1.000000          1.000000   0.003786
## 2       X9            0.959899          0.959899   0.003634
## 3      X11            0.921462          0.921462   0.003489
## 4      X12            0.919376          0.919376   0.003481
## 5      X13            0.908859          0.908859   0.003441
##
## ---
##     variable relative_importance scaled_importance percentage
## 779      X30            0.172565          0.172565   0.000653
## 780     X114            0.161225          0.161225   0.000610
## 781       X3            0.157096          0.157096   0.000595
## 782     X757            0.146777          0.146777   0.000556
## 783       X2            0.110829          0.110829   0.000420
## 784       X1            0.021193          0.021193   0.000080
```

```
#Veamos la capacidad de prediccion del modelo
predict_test3 = h2o.predict(modeloci3, newdata = test.hex)
```

```
##   |                                                                              |
```

```
pred3 = as.data.frame(predict_test3)
head(pred3,4)
```

```
##   predict           p0           p1           p2           p3           p4
## 1       9 3.283290e-22 3.615924e-23 5.839722e-20 4.668637e-22 1.074106e-23
## 2       2 2.121440e-07 1.491553e-26 9.999834e-01 1.161766e-19 4.564160e-08
```

```
## 3       1 2.040079e-25 1.000000e+00 1.110727e-31 2.216117e-30 9.448231e-36
## 4       1 6.773238e-28 1.000000e+00 3.584727e-30 2.955823e-23 9.973473e-31
##             p5           p6           p7           p8           p9
## 1 9.046457e-03 1.145250e-18 3.473930e-02 9.618516e-08 9.562141e-01
## 2 2.029324e-17 1.638753e-05 2.633151e-26 3.356881e-15 2.740232e-27
## 3 7.676945e-44 6.173300e-35 7.970359e-36 1.580196e-33 2.075984e-42
## 4 1.175270e-38 1.138642e-29 3.350260e-35 3.561182e-38 9.353615e-36
```

```
tabla3 = table(Obs=datos_test[,1], Pred=pred3[,1])
aciertos3 = 100*diag(prop.table(tabla3,1))
acierto3 = 100*sum(diag(tabla3))/sum(tabla3)
cat("Acierto test=", acierto3, "% \n")
```

```
## Acierto test= 88.68 %
```

Modelo 4: Red con dos capas ocultas de 530 nodos cada una

```
modeloci4 = h2o.deeplearning(
  x = 2:785, #variables predictoras
  y = 1, #variable respuesta
  training_frame = train.hex, #datos de entrenamiento
  validation_frame = test.hex, #datos de validacion/test
  distribution="multinomial",
  activation = 'RectifierWithDropout',
  hidden = c(530,530), #tamaño de las capas ocultas
  hidden_dropout_ratio = c(0.5,0.5),
  input_dropout_ratio = 0.2,
  epochs = 50, #iteraciones
  l1 = 1e-5,
  l2 = 1e-5,
  rho = 0.99,
  epsilon = 1e-8,
  train_samples_per_iteration = 1000)
```

```
##   |                                                                              |
```

```
summary(modeloci4)
```

```
## Model Details:
## ==============
##
## H2OMultinomialModel: deeplearning
## Model Key:  DeepLearning_model_R_1655405240903_6
## Status of Neuron Layers: predicting Y, 10-class classification, multinomial distribution, CrossEntro
##   layer units             type dropout       l1       l2 mean_rate rate_rms
## 1     1   784            Input 20.00 %       NA       NA        NA       NA
## 2     2   530 RectifierDropout 50.00 % 0.000010 0.000010  0.018207 0.047337
## 3     3   530 RectifierDropout 50.00 % 0.000010 0.000010  0.001331 0.000805
## 4     4    10          Softmax      NA 0.000010 0.000010  0.052282 0.161100
##   momentum mean_weight weight_rms mean_bias bias_rms
```

```
## 1       NA          NA         NA        NA       NA
## 2 0.000000    0.003164   0.093914 -0.418371 0.225066
## 3 0.000000   -0.016838   0.051317  0.798828 0.105783
## 4 0.000000   -0.255730   0.376309 -3.797295 1.213228
##
## H2OMultinomialMetrics: deeplearning
## ** Reported on training data. **
## ** Metrics reported on temporary training frame with 9951 samples **
##
## Training Set Metrics:
## =====================
##
## MSE: (Extract with `h2o.mse`) 0.05812722
## RMSE: (Extract with `h2o.rmse`) 0.2410959
## Logloss: (Extract with `h2o.logloss`) 0.1861938
## Mean Per-Class Error: 0.07170141
## AUC: (Extract with `h2o.auc`) NaN
## AUCPR: (Extract with `h2o.aucpr`) NaN
## Confusion Matrix: Extract with `h2o.confusionMatrix(<model>,train = TRUE)`)
## ========================================================================
## Confusion Matrix: Row labels: Actual class; Column labels: Predicted class
##           0   1   2    3    4   5   6   7    8   9  Error          Rate
## 0       957   0   6    3    0   0  67   0    3   0 0.0763 =  79 / 1.036
## 1         1 943   1   12    0   0   0   0    0   0 0.0146 =    14 / 957
## 2        13   0 872    5  100   0  55   0    4   0 0.1687 = 177 / 1.049
## 3        21   0   2  981   25   0   7   0    1   0 0.0540 =  56 / 1.037
## 4         4   0  30   17  895   0  26   0    3   0 0.0821 =    80 / 975
## 5         0   0   0    0    0 913   0  29    0   5 0.0359 =    34 / 947
## 6        97   1  48   10   70   0 776   0    4   0 0.2286 = 230 / 1.006
## 7         0   0   0    0    0   2   0 938    0  20 0.0229 =    22 / 960
## 8         2   0   0    1    0   0   2   0 1013   0 0.0049 =   5 / 1.018
## 9         0   0   0    0    0   5   0  23    0 938 0.0290 =    28 / 966
## Totals 1095 944 959 1029 1090 920 933 990 1028 963 0.0729 = 725 / 9.951
##
## Hit Ratio Table: Extract with `h2o.hit_ratio_table(<model>,train = TRUE)`
## =======================================================================
## Top-10 Hit Ratios:
##     k hit_ratio
## 1   1  0.927143
## 2   2  0.985931
## 3   3  0.997186
## 4   4  0.999196
## 5   5  0.999498
## 6   6  1.000000
## 7   7  1.000000
## 8   8  1.000000
## 9   9  1.000000
## 10 10  1.000000
##
##
##
##
## H2OMultinomialMetrics: deeplearning
## ** Reported on validation data. **
```

```
## ** Metrics reported on full validation frame **
##
## Validation Set Metrics:
## =====================
##
## Extract validation frame with `h2o.getFrame("datos_test_sid_bfc4_3")`
## MSE: (Extract with `h2o.mse`) 0.0865025
## RMSE: (Extract with `h2o.rmse`) 0.2941131
## Logloss: (Extract with `h2o.logloss`) 0.3163414
## Mean Per-Class Error: 0.1061
## AUC: (Extract with `h2o.auc`) NaN
## AUCPR: (Extract with `h2o.aucpr`) NaN
## Confusion Matrix: Extract with `h2o.confusionMatrix(<model>,valid = TRUE)`)
## =========================================================================
## Confusion Matrix: Row labels: Actual class; Column labels: Predicted class
##           0   1   2   3    4   5   6    7    8   9  Error             Rate
## 0       850   0   8  18    6   0 107    1   10   0 0.1500 =    150 / 1.000
## 1         1 970   1  20    3   0   4    0    1   0 0.0300 =     30 / 1.000
## 2        17   0 796   9  110   1  67    0    0   0 0.2040 =    204 / 1.000
## 3        26   3   8 897   33   1  29    0    3   0 0.1030 =    103 / 1.000
## 4         1   0  56  25  882   0  33    0    3   0 0.1180 =    118 / 1.000
## 5         0   0   0   0    0 946   0   35    1  18 0.0540 =     54 / 1.000
## 6       124   1  76  18   74   0 697    0   10   0 0.3030 =    303 / 1.000
## 7         0   0   0   0    0   6   0  969    0  25 0.0310 =     31 / 1.000
## 8         2   0   1   6    5   3   1    3  979   0 0.0210 =     21 / 1.000
## 9         0   0   0   0    0   7   1   39    0 953 0.0470 =     47 / 1.000
## Totals 1021 974 946 993 1113 964 939 1047 1007 996 0.1061 = 1.061 / 10.000
##
## Hit Ratio Table: Extract with `h2o.hit_ratio_table(<model>,valid = TRUE)`
## =======================================================================
## Top-10 Hit Ratios:
##     k hit_ratio
## 1   1  0.893900
## 2   2  0.971100
## 3   3  0.991400
## 4   4  0.996000
## 5   5  0.997900
## 6   6  0.999000
## 7   7  0.999300
## 8   8  0.999400
## 9   9  0.999700
## 10 10  1.000000
##
##
##
##
##
##
## Scoring History:
##             timestamp          duration training_speed  epochs iterations
## 1 2022-06-16 22:18:23         0.000 sec             NA 0.00000          0
## 2 2022-06-16 22:18:25         8.973 sec    746 obs/sec 0.01665          1
## 3 2022-06-16 22:19:34  1 min 17.729 sec   1171 obs/sec 1.23402         74
## 4 2022-06-16 22:20:44  2 min 28.673 sec   1154 obs/sec 2.42783        146
```

```
## 5 2022-06-16 22:22:04  3 min 47.681 sec   1217 obs/sec 4.02277        242
##         samples training_rmse training_logloss training_r2
## 1      0.000000            NA               NA          NA
## 2    999.000000       0.49467          1.26555     0.97043
## 3  74041.000000       0.34762          0.38099     0.98540
## 4 145670.000000       0.33313          0.34993     0.98659
## 5 241366.000000       0.32441          0.32695     0.98728
##   training_classification_error training_auc training_pr_auc validation_rmse
## 1                            NA           NA              NA              NA
## 2                       0.27736           NA              NA         0.49648
## 3                       0.14119           NA              NA         0.36420
## 4                       0.13406           NA              NA         0.35118
## 5                       0.12632           NA              NA         0.34306
##   validation_logloss validation_r2 validation_classification_error
## 1                 NA            NA                              NA
## 2            1.31214       0.97012                         0.27690
## 3            0.43176       0.98392                         0.15780
## 4            0.39918       0.98505                         0.15190
## 5            0.37833       0.98573                         0.14180
##   validation_auc validation_pr_auc
## 1             NA                NA
## 2             NA                NA
## 3             NA                NA
## 4             NA                NA
## 5             NA                NA
##
## ---
##              timestamp          duration training_speed   epochs iterations
## 37 2022-06-16 22:53:19 35 min  3.622 sec   1440 obs/sec 45.30805       2718
## 38 2022-06-16 22:54:32 36 min 14.378 sec   1442 obs/sec 46.94538       2816
## 39 2022-06-16 22:55:31 37 min 13.176 sec   1441 obs/sec 48.19757       2891
## 40 2022-06-16 22:56:26 38 min  9.860 sec   1444 obs/sec 49.49197       2969
## 41 2022-06-16 22:56:53 38 min 36.771 sec   1444 obs/sec 50.00910       3000
## 42 2022-06-16 22:57:00 38 min 42.999 sec   1444 obs/sec 50.00910       3000
##           samples training_rmse training_logloss training_r2
## 37 2718483.000000       0.24135          0.18758     0.99296
## 38 2816723.000000       0.24240          0.19088     0.99290
## 39 2891854.000000       0.24103          0.18842     0.99298
## 40 2969518.000000       0.24110          0.18619     0.99298
## 41 3000546.000000       0.24240          0.18729     0.99290
## 42 3000546.000000       0.24110          0.18619     0.99298
##    training_classification_error training_auc training_pr_auc validation_rmse
## 37                       0.07276           NA              NA         0.29439
## 38                       0.07326           NA              NA         0.29606
## 39                       0.07125           NA              NA         0.29396
## 40                       0.07286           NA              NA         0.29411
## 41                       0.07286           NA              NA         0.29577
## 42                       0.07286           NA              NA         0.29411
##    validation_logloss validation_r2 validation_classification_error
## 37            0.32058       0.98950                         0.10790
## 38            0.32823       0.98938                         0.10780
## 39            0.31689       0.98953                         0.10600
## 40            0.31634       0.98951                         0.10610
## 41            0.31750       0.98940                         0.10720
```

```
## 42            0.31634       0.98951                          0.10610
##    validation_auc validation_pr_auc
## 37             NA                NA
## 38             NA                NA
## 39             NA                NA
## 40             NA                NA
## 41             NA                NA
## 42             NA                NA
##
## Variable Importances: (Extract with `h2o.varimp`)
## =================================================
##
## Variable Importances:
##   variable relative_importance scaled_importance percentage
## 1     X420            1.000000          1.000000   0.002748
## 2     X393            0.983969          0.983969   0.002704
## 3     X281            0.881438          0.881438   0.002422
## 4     X506            0.868741          0.868741   0.002387
## 5     X365            0.845617          0.845617   0.002324
##
## ---
##     variable relative_importance scaled_importance percentage
## 779      X28            0.029176          0.029176   0.000080
## 780      X30            0.021147          0.021147   0.000058
## 781       X3            0.018525          0.018525   0.000051
## 782     X757            0.013392          0.013392   0.000037
## 783       X2            0.011534          0.011534   0.000032
## 784       X1            0.009129          0.009129   0.000025
```

```
#Veamos la capacidad de prediccion del modelo
predict_test4 = h2o.predict(modeloci4, newdata = test.hex)
```

```
##   |                                                                              |
```

```
pred4 = as.data.frame(predict_test4)
head(pred4,4)
```

```
##   predict           p0           p1           p2           p3           p4
## 1       9 7.919340e-16 7.291762e-12 6.145541e-17 6.884049e-13 7.683121e-16
## 2       2 2.448637e-09 1.406075e-20 9.997409e-01 8.951538e-15 1.909838e-04
## 3       1 1.556960e-31 1.000000e+00 3.041578e-36 2.529840e-27 9.520722e-30
## 4       1 2.421211e-27 1.000000e+00 2.237646e-29 1.305511e-19 7.259909e-26
##             p5           p6           p7           p8           p9
## 1 2.026691e-04 4.959951e-16 1.091203e-02 1.445659e-15 9.888853e-01
## 2 3.302274e-19 6.808777e-05 1.738888e-19 5.849070e-16 7.428638e-17
## 3 5.161501e-42 5.436355e-34 5.225841e-37 2.455047e-43 3.352116e-35
## 4 9.402315e-37 1.102479e-26 1.145700e-33 1.468964e-37 2.288110e-32
```

```
tabla4 = table(Obs=datos_test[,1], Pred=pred4[,1])
aciertos4 = 100*diag(prop.table(tabla4,1))
acierto4 = 100*sum(diag(tabla4))/sum(tabla4)
cat("Acierto test=", acierto4, "% \n")
```

```
## Acierto test= 89.39 %
```

Ejemplo de clasificación de imágenes - Clasificación de prendas de ropa en Python

Raquel Beltrán Barba

CLASIFICACIÓN DE IMÁGENES DE PRENDAS DE ROPA

Primero cargamos las librerías que vamos a necesitar:

```
import numpy as np
import pandas as pd
import math
import sklearn
import sklearn.preprocessing
import datetime
import keras
import os
import matplotlib.pyplot as plt
import tensorflow as tf
import csv
from keras.datasets import fashion_mnist
from keras.utils import plot_model
from sklearn.preprocessing import LabelBinarizer
```

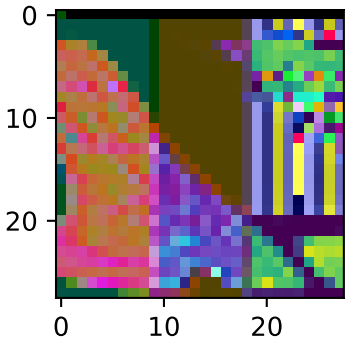
Lectura y adaptación del conjunto de datos

```
#os.chdir("C:\\Users\\raque\\OneDrive\\Escritorio\\TFG\\Bases.de.datos\\FashionMnist")
(X_train, y_train), (X_test, y_test) = fashion_mnist.load_data()
```

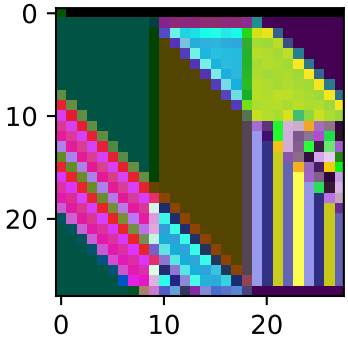
```
# Redimensionamos los datos
X_train = X_train.reshape((X_train.shape[0], 28 * 28 * 1))
X_test = X_test.reshape((X_test.shape[0], 28 * 28 * 1))
    #Normalizamos los datos
X_train = X_train.astype('float32') / 255.0
X_test = X_test.astype('float32') / 255.0

#Debemos convertir las etiquetas de las prendas en vectores one-hot-encoder
label_binarizer = LabelBinarizer()
y_train = label_binarizer.fit_transform(y_train)
y_test = label_binarizer.fit_transform(y_test)
```

Representación de algunas imágenes

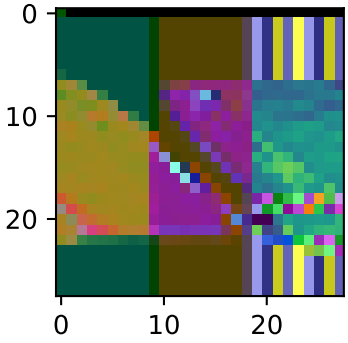
```
plt.imshow(X_train[150].reshape(28,28), cmap='viridis', interpolation='none')
```



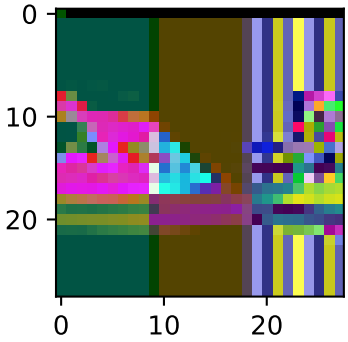

```
plt.imshow(X_train[151].reshape(28,28), cmap='viridis', interpolation='none')
```



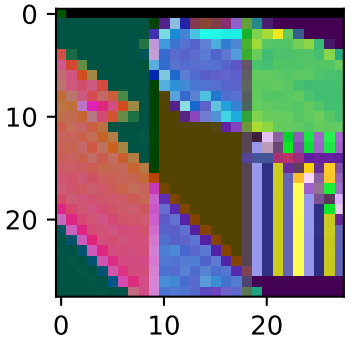

```
plt.imshow(X_train[152].reshape(28,28), cmap='viridis', interpolation='none')
```



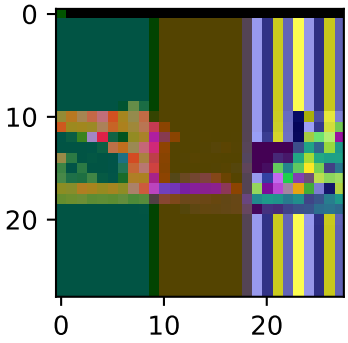

```
plt.imshow(X_train[153].reshape(28,28), cmap='viridis', interpolation='none')
```



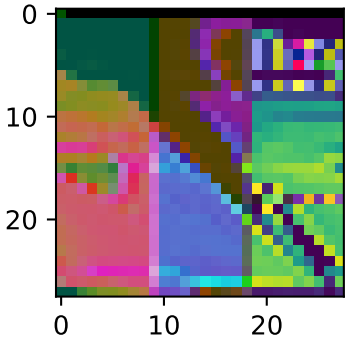

```
plt.imshow(X_train[154].reshape(28,28), cmap='viridis', interpolation='none')
```



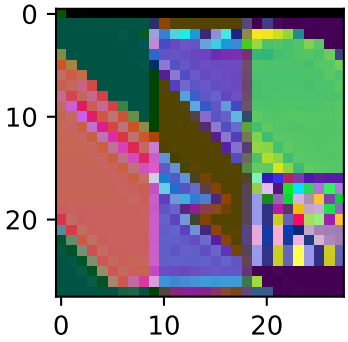

```
plt.imshow(X_train[155].reshape(28,28), cmap='viridis', interpolation='none')
```



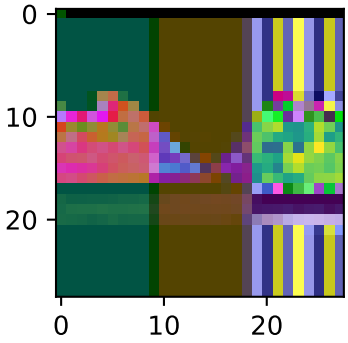

```
plt.imshow(X_train[156].reshape(28,28), cmap='viridis', interpolation='none')
```



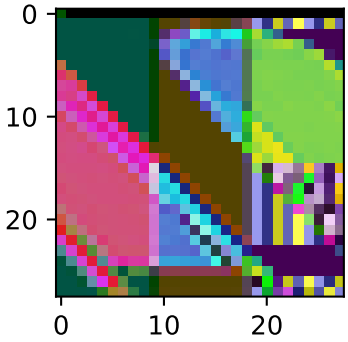

```
plt.imshow(X_train[157].reshape(28,28), cmap='viridis', interpolation='none')
```



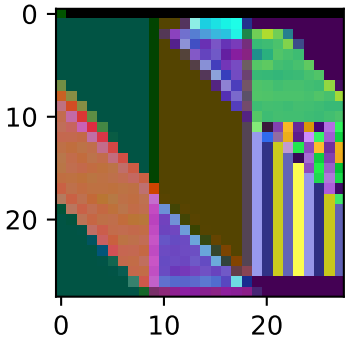

```
plt.imshow(X_train[158].reshape(28,28), cmap='viridis', interpolation='none')
```



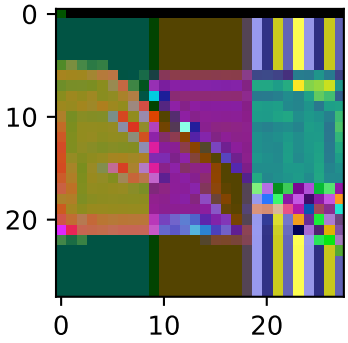

```
plt.imshow(X_train[159].reshape(28,28), cmap='viridis', interpolation='none')
```




```
plt.imshow(X_train[160].reshape(28,28), cmap='viridis', interpolation='none')
```




```
plt.imshow(X_train[161].reshape(28,28), cmap='viridis', interpolation='none')
```



Comenzamos a trabajar con los modelos de Redes Neuronales

```
from keras import Sequential
from keras.layers import Dense
from keras.optimizers import Adam
from sklearn.metrics import classification_report
```

Modelo 1: Red con dos capas ocultas de 100 nodos cada una

```
modelci1 = Sequential()
modelci1.add(Dense(100, input_shape=(28 * 28 * 1,), activation='relu')) #primera capa oculta
modelci1.add(Dense(100, activation='relu')) #segunda capa oculta
modelci1.add(Dense(10, activation='softmax')) #capa de salida
```

```
  #Entrenamos la red
adam = Adam()
modelci1.compile(loss='categorical_crossentropy', optimizer=adam, metrics=['accuracy'])
modelci_1 = modelci1.fit(X_train, y_train, validation_data=(X_test, y_test), epochs=50)
```

```
  #Veamos la capacidad de predicción
predictions = modelci1.predict(X_test, batch_size=64)
```

```
##
##   1/157 [..............................] - ETA: 19s
##  35/157 [=====>........................] - ETA: 0s
##  74/157 [=============>................] - ETA: 0s
## 113/157 [====================>.........] - ETA: 0s
## 150/157 [===========================>..] - ETA: 0s
## 157/157 [==============================] - 0s 1ms/step
```

```
print(classification_report(y_test.argmax(axis=1),
                            predictions.argmax(axis=1),
                            target_names=['T-shirt/top', 'Trouser', 'Pullover', 'Dress',
                            'Coat', 'Sandal', 'Shirt', 'Sneaker', 'Bag', 'Ankle boot']))
```

```
##               precision    recall  f1-score   support
##
##  T-shirt/top       0.80      0.86      0.83      1000
##      Trouser       0.98      0.98      0.98      1000
##     Pullover       0.77      0.87      0.81      1000
##        Dress       0.91      0.89      0.90      1000
##         Coat       0.84      0.80      0.82      1000
##       Sandal       0.97      0.97      0.97      1000
##        Shirt       0.76      0.65      0.70      1000
##      Sneaker       0.94      0.97      0.96      1000
##          Bag       0.97      0.97      0.97      1000
##   Ankle boot       0.97      0.94      0.96      1000
##
##     accuracy                           0.89     10000
##    macro avg       0.89      0.89      0.89     10000
## weighted avg       0.89      0.89      0.89     10000
```

```
score = modelci1.evaluate(X_test,y_test,verbose=0)
print('Test Accuracy : {:.4f}'.format(score[1]))
```

```
## Test Accuracy : 0.8904
```

Modelo 2: Red con una capa oculta de 100 nodos

```
modelci2 = Sequential()
modelci2.add(Dense(100, input_shape=(28 * 28 * 1,), activation='relu')) #primera capa oculta
modelci2.add(Dense(10, activation='softmax')) #capa de salida
```

```
  #Entrenamos la red
adam = Adam()
modelci2.compile(loss='categorical_crossentropy', optimizer=adam, metrics=['accuracy'])
modelci_2 = modelci2.fit(X_train, y_train, validation_data=(X_test, y_test), epochs=50)
```

```
  #Veamos la capacidad de predicción
predictions2 = modelci2.predict(X_test, batch_size=64)
```

```
##
##   1/157 [..............................] - ETA: 9s
##  40/157 [======>.......................] - ETA: 0s
##  80/157 [==============>...............] - ETA: 0s
## 118/157 [=====================>........] - ETA: 0s
## 151/157 [============================>..] - ETA: 0s
## 157/157 [==============================] - 0s 1ms/step
```

```
print(classification_report(y_test.argmax(axis=1),
                            predictions2.argmax(axis=1),
                            target_names=['T-shirt/top', 'Trouser', 'Pullover', 'Dress',
                            'Coat', 'Sandal', 'Shirt', 'Sneaker', 'Bag', 'Ankle boot']))
```

```
##               precision    recall  f1-score   support
##
##  T-shirt/top       0.83      0.83      0.83      1000
##      Trouser       0.98      0.98      0.98      1000
##     Pullover       0.78      0.81      0.79      1000
##        Dress       0.92      0.85      0.88      1000
##         Coat       0.74      0.88      0.81      1000
##       Sandal       0.97      0.95      0.96      1000
##        Shirt       0.77      0.64      0.70      1000
##      Sneaker       0.91      0.97      0.94      1000
##          Bag       0.98      0.97      0.97      1000
##   Ankle boot       0.96      0.94      0.95      1000
##
##     accuracy                           0.88     10000
##    macro avg       0.88      0.88      0.88     10000
## weighted avg       0.88      0.88      0.88     10000
```

```
score2 = modelci2.evaluate(X_test,y_test,verbose=0)
print('Test Accuracy : {:.4f}'.format(score2[1]))
```

```
## Test Accuracy : 0.8823
```

Modelo 3: Red con una capa oculta de 530 nodos

```
modelci3 = Sequential()
modelci3.add(Dense(530, input_shape=(28 * 28 * 1,), activation='relu')) #primera capa oculta
modelci3.add(Dense(10, activation='softmax')) #capa de salida
```

```
  #Entrenamos la red
adam = Adam()
modelci3.compile(loss='categorical_crossentropy', optimizer=adam, metrics=['accuracy'])
modelci_3 = modelci3.fit(X_train, y_train, validation_data=(X_test, y_test), epochs=50)
```

```
  #Veamos la capacidad de predicción
predictions3 = modelci3.predict(X_test, batch_size=64)
```

```
##
##   1/157 [..............................] - ETA: 8s
##  24/157 [===>..........................] - ETA: 0s
##  48/157 [========>.....................] - ETA: 0s
##  72/157 [============>.................] - ETA: 0s
##  97/157 [================>.............] - ETA: 0s
## 123/157 [====================>.......] - ETA: 0s
## 150/157 [==========================>..] - ETA: 0s
## 157/157 [==============================] - 0s 2ms/step
```

```
print(classification_report(y_test.argmax(axis=1),
                            predictions3.argmax(axis=1),
                            target_names=['T-shirt/top', 'Trouser', 'Pullover', 'Dress',
                            'Coat', 'Sandal', 'Shirt', 'Sneaker', 'Bag', 'Ankle boot']))
```

```
##               precision    recall  f1-score   support
##
##  T-shirt/top       0.88      0.74      0.80      1000
##      Trouser       0.97      0.99      0.98      1000
##     Pullover       0.81      0.79      0.80      1000
##        Dress       0.92      0.88      0.90      1000
##         Coat       0.82      0.80      0.81      1000
##       Sandal       0.97      0.98      0.97      1000
##        Shirt       0.63      0.79      0.70      1000
##      Sneaker       0.97      0.95      0.96      1000
##          Bag       0.98      0.97      0.97      1000
##   Ankle boot       0.96      0.96      0.96      1000
##
##     accuracy                           0.88     10000
##    macro avg       0.89      0.88      0.89     10000
## weighted avg       0.89      0.88      0.89     10000
```

```
score3 = modelci3.evaluate(X_test,y_test,verbose=0)
print('Test Accuracy : {:.4f}'.format(score3[1]))
```

```
## Test Accuracy : 0.8848
```

Modelo 4: Red con dos capas ocultas de 530 nodos cada una

```
modelci4 = Sequential()
modelci4.add(Dense(530, input_shape=(28 * 28 * 1,), activation='relu')) #primera capa oculta
modelci4.add(Dense(530, activation='relu')) #segunda capa oculta
modelci4.add(Dense(10, activation='softmax')) #capa de salida
```

```
  #Entrenamos la red
adam = Adam()
modelci4.compile(loss='categorical_crossentropy', optimizer=adam, metrics=['accuracy'])
modelci_4 = modelci4.fit(X_train, y_train, validation_data=(X_test, y_test), epochs=50)
```

```
  #Veamos la capacidad de predicción
predictions4 = modelci4.predict(X_test, batch_size=64)
```

```
##
##   1/157 [..............................] - ETA: 8s
##  25/157 [===>..........................] - ETA: 0s
##  49/157 [========>.....................] - ETA: 0s
##  72/157 [============>.................] - ETA: 0s
##  94/157 [================>.............] - ETA: 0s
## 115/157 [====================>.........] - ETA: 0s
## 137/157 [=========================>....] - ETA: 0s
## 157/157 [==============================] - 0s 2ms/step
```

```
print(classification_report(y_test.argmax(axis=1),
                            predictions4.argmax(axis=1),
                            target_names=['T-shirt/top', 'Trouser', 'Pullover', 'Dress',
                            'Coat', 'Sandal', 'Shirt', 'Sneaker', 'Bag', 'Ankle boot']))
```

```
##               precision    recall  f1-score   support
##
##  T-shirt/top       0.84      0.85      0.84      1000
##      Trouser       1.00      0.98      0.99      1000
##     Pullover       0.80      0.86      0.83      1000
##        Dress       0.89      0.93      0.91      1000
##         Coat       0.85      0.82      0.83      1000
##       Sandal       0.98      0.96      0.97      1000
##        Shirt       0.74      0.70      0.72      1000
##      Sneaker       0.94      0.97      0.95      1000
##          Bag       0.99      0.97      0.98      1000
##   Ankle boot       0.96      0.94      0.95      1000
##
##     accuracy                           0.90     10000
##    macro avg       0.90      0.90      0.90     10000
## weighted avg       0.90      0.90      0.90     10000
```

```
score4 = modelci4.evaluate(X_test,y_test,verbose=0)
print('Test Accuracy : {:.4f}'.format(score4[1]))
```

```
## Test Accuracy : 0.8982
```