Técnicas de clasificación *supervised learning*

PID_00264729

Xavi Font

Tiempo mínimo de dedicación recomendado: 5 horas

Universitat Oberta de Catalunya

Xavi Font

El encargo y la creación de este recurso de aprendizaje UOC han sido coordinados por el profesor: Jordi Ayza Graells (2019)

Primera edición: marzo 2019

Av. Tibidabo, 39-43, 08035 Barcelona
Diseño: Manel Andreu
Realización editorial: Oberta UOC Publishing, SL

Índice

Introducción

El primer paso para dominar las principales técnicas asociadas al *machine learning* (ML) y su aplicación en ámbitos del *business* y el *data analytics* es comprender la metodología que hay detrás del ML. El aspecto más relevante de los métodos de ML es que los algoritmos utilizados son entrenados y ajustados a partir de la naturaleza del problema tratado y de los datos facilitados.

La gran relevancia y notoriedad que han adquirido estos métodos está perfectamente justificada por el éxito de sus resultados en un gran abanico de aplicaciones. Veamos una lista de estas:

- **Satisfacción de clientes**: los algoritmos de ML ayudan a las empresas a hacer un seguimiento de la felicidad y fidelidad de sus clientes. Analizando la actividad del cliente, se puede predecir si es una potencial baja antes de que se dé y, por tanto, se pueden tomar medidas para corregirlo.

- **Tendencias del mercado**: ML puede seguir los patrones de *trading* y su volatilidad para proponer tendencias que a simple vista pasarían inadvertidas por un experto humano.

- **Evaluación del riesgo**: estudiando los patrones de gasto, los datos históricos y financieros, más otros datos relevantes, los algoritmos de ML pueden evaluar con precisión el riesgo tanto en la contratación de seguros como en las evaluaciones de concesión de hipotecas o créditos.

- **Sistemas de recomendación**: este tipo de aplicación que todos conocemos por su uso extendido en compañías punteras como Amazon o Netflix y otras relacionadas con servicios de *dating* permite hacer recomendaciones basadas en las preferencias del cliente. Compañías como Netflix estiman el ROI en mil millones de euros al año por su impacto en la retención de clientes.

Esta lista, sin ser exhaustiva, representa una aproximación al potencial que hay detrás de este tipo de servicios. Hay muchos más ejemplos dentro de la industria de seguridad (*cybersecurity*) o de la industria de salud, en el procesamiento del lenguaje natural o en la detección de rotura de componentes industriales.

En general, podemos clasificar los algoritmos de ML en tres subgrupos según el tipo de supervisión que estos tengan:

- **Algoritmos supervisados**: situación que vamos a cubrir en este módulo. La principal característica de este tipo de algoritmos es que el proceso de entrenamiento se realiza con la solución (está guiada por la solución). Es decir, el algoritmo recibe un conjunto de variables (una imagen, unas características de una pieza), más la solución asociada al problema presentada como una variable *target* (una etiqueta que describe el contenido de la imagen, una etiqueta con el nivel de calidad de la pieza). Entre estas técnicas podemos encontrar una gran variedad de propuestas.

- **Algoritmos no supervisados**: situación que vamos a cubrir en el módulo siguiente. La situación que resuelven este tipo de algoritmos es que no hay variable *target*. Las variables tienen el mismo peso o importancia. El objetivo que se suele perseguir es, por una parte, agrupar aquellas observaciones parecidas y, por otra, las que no los son colocarlas en grupos diferentes. La lista de posibles variantes de métodos no supervisados está descrita en otro módulo.

- **Aprendizaje reforzado**: método que se basa en agentes y su interacción con el entorno.

Objectivos

Los objetivos que persigue este módulo son los siguientes:

1. Entender las diferentes estrategias de aprendizaje así como el tipo de problemas que estas resuelven.
2. Diferenciar las dos aproximaciones de aprendizaje supervisado: métodos de regresión y métodos de clasificación.
3. Aplicar correctamente procedimientos de evaluación de los modelos ajustados para garantizar su correcto funcionamiento en entornos reales.

Otros objetivos que se deben desarrollar:

- Entender y aplicar los modelos de regresión tradicionales tanto desde la perspectiva estadística como desde la perspectiva de ML.
- Saber aplicar métodos avanzados de regresión más adaptados a situaciones próximas al *big data* (regresión penalizada).
- Entender y aplicar métodos de *deep learning* usando la API de keras. En la aplicación de soluciones *deep learning*:
  - Ajustar y preparar correctamente los datos como un tensor listo para ser preparado para el *training* y el *testing*.
  - Definir la arquitectura de la red.
  - Proponer las opciones más indicadas para el problema que queremos resolver.
  - Compilar el modelo de red.
  - Entrenar sobre el conjunto de datos de *training*.
  - Verificar sobre el conjunto de datos de *test*.

En general, seleccionaremos los modelos con las predicciones más precisas sobre los datos de test. Los modelos que evaluamos no pretenden descubrir la verdad subyacente, sino resolver un problema de aprendizaje. Se apreciarán aquellos métodos que sean de propósito general, totalmente automáticos y que se adapten correctamente al problema descrito.

Se tendrán en consideración aspectos de velocidad de computación y de escalado para poder dar garantías en caso de aplicarse en entornos de *big data*.

1. Problemas de aprendizaje supervisado

1.1. Introducción

El primer reto que debemos entender corresponde a las técnicas de aprendizaje supervisado. Dentro del conjunto de técnicas de *machine learning* (ML), diferenciamos claramente las llamadas técnicas de clasificación o de aprendizaje supervisado y las que son de *clustering* (agrupación) o de aprendizaje no supervisado.

Vamos a definir de manera clara y sencilla ambos conceptos.

1.2. Introducción a los procesos de aprendizaje

Los métodos supervisados se caracterizan por realizar el proceso de aprendizaje a través de información complementaria. Esta información complementaria se denomina en ocasiones *target* o variable respuesta, por ser la variable que deseamos estudiar y analizar.

Este proceso de aprendizaje guiado, gracias a esta etiqueta, permite que los métodos se adapten, corrijan y desarrollen bajo esta premisa.

Los métodos no supervisados se caracterizan por que ninguna de las variables de nuestro conjunto de variables tiene más importancia que el resto. No tenemos una variable *target* que ayude en la tarea que resuelven los métodos no supervisados de *clustering*.

Esta tarea consiste en identificar observaciones similares en el mismo clúster, mientras que observaciones no similares se asignan a clústeres diferentes.

1.2.1. Ejemplos de aprendizaje

Podríamos escribir un libro con ejemplos de aprendizaje. Los medios de comunicación han presentado multitud de noticias hablando de ejemplos relativos

a aprendizaje automático: AlphaGo (ejemplo de *reinforcement learning*), identificación de melanoma por dos equipos ML algorithm frente a dermatólogos, entre otros.

Esta pequeña lista es para que veamos el gran abanico de problemas que podemos tratar.

- **Ejemplo 1**: Tenemos un conjunto de fotografías de vehículos que queremos clasificar como turismo, motocicleta, camioneta o camión. ¿Qué información debe recibir el proceso? Una imagen a color como entrada y adicionalmente una etiqueta describiendo la clase a la que pertenece este vehículo. ver nvidia ejemplo

- **Ejemplo 2**: Tenemos características de diferentes vehículos y nos interesa estudiar qué variables pueden tener relación con la variable de consumo (mpg).

- **Ejemplo 3**: Tenemos lecturas de la actividad cerebral de diferentes usuarios a través de un casco EEG y una etiqueta que nos indica si este usuario es alcohólico o no.

- **Ejemplo 4**: Tenemos datos relativos a los procesos de compra de nuestros clientes y una variable respuesta que nos indica si la cliente está embarazada o no.

- **Ejemplo 5**: Tenemos datos meteorológicos y otra variable que indica errores producidos en uno de los dispositivos de un transbordador espacial.

- **Ejemplo 6**: Disponemos de datos relativos a análisis clínicos y características anatómicas del paciente y una etiqueta que nos indica si es o no diabético.

En general, podemos interpretar datos de entrada que actúan como explicativos: X (representa la entrada a nuestro proceso de aprendizaje), y la variable *target* o respuesta: Y (representa la salida).

A partir de las variables explicativas, que en principio deben ser relevantes y suficientes para el problema que se resuelve, los métodos de clasificación utilizan estos datos junto con la información de la variable respuesta para generar un modelo que permita predecir esta respuesta de manera aceptable.

Los métodos supervisados que generan una respuesta pueden dividirse en aquellos donde la respuesta es un valor real (por ejemplo, euros, temperatura, peso, años) y los que generan como respuesta una etiqueta o grupo (por ejemplo, marca de coche: SEAT, SKODA, NISSAN; presencia de defecto en la

botella analizada: OK, POSIBLE DEFECTO, DEFECTO; si el usuario presenta enfermedad: DIABÉTICO, SANO).

Si la respuesta es un valor numérico, estamos ante métodos de regresión. Estos son métodos con una larga historia y uso en multitud de ámbitos de la ciencia y la tecnología. Si la respuesta no es numérica, es decir, si el clasificador da como respuesta una etiqueta, clase, grupo o categoría, entonces estamos ante la tipología típica de un problema de clasificación.

Los métodos no supervisados, en general, se caracterizan por que todas las variables o características de nuestro conjunto de datos tienen el mismo peso o la misma importancia. Este tipo de métodos no disponen de una información adicional (variable *target* o respuesta) como los métodos supervisados y, por tanto, se les denomina métodos no supervisados.

Dentro esta esta categoría de métodos nos podemos encontrar técnicas como las de *clustering* o el análisis de componentes principales.

1.2.2. Proceso involucrado en los métodos de aprendizaje

Las etapas involucradas en los problemas de aprendizaje supervisado suelen ser seis.

Primera: Definición de objetivos

La primera etapa permite especificar qué debe solucionar nuestro proyecto. Estos objetivos vienen definidos por los responsables estratégicos del negocio, por requisitos de los mandos intermedios o incluso por demandas no satisfechas de los clientes.

Ejemplos de estas situaciones:

- Cómo predecir el precio de nuestra cartera de pisos en función de las características principales de estos.
- De qué depende el precio de venta de un vehículo de segunda mano.
- Cuánto pago por cada cv de potencia cuando compro un vehículo, o cuánto pago por metro cuadrado al comprar un piso.
- Cómo podemos identificar piezas que no cumplen los estándares de calidad marcados por el departamento calidad.
- Cómo evitar que un cliente se dé de baja de nuestros servicios.
- Qué tipo de recomendación deberíamos dar para que nuestra cartera de clientes cubra sus necesidades de forma más eficiente.
- Cómo agrupamos a nuestros clientes.

En esta etapa, se pueden especificar puntos imprescindibles de monitorización o de medidas para garantizar el seguimiento del objetivo planteado. El uso de KPI o de medidas de ROI puede ser de interés.

Segunda: Adquisición de datos

Uno de los puntos imprescindibles para realizar este tipo de proyectos es el de disponer de datos. Algunas preguntas que debemos resolver son las siguientes:

- ¿Qué datos tenemos disponibles?
- ¿Son datos suficientes para completar el objetivo?
- ¿Necesitamos datos adicionales, cuáles son y dónde podemos encontrarlos?
- ¿Costes asociados a la adquisición?
- ¿Corrección y limpieza de errores?

Debemos verificar que los datos que finalmente tenemos a nuestra disposición son lo suficientemente válidos y correctos para resolver el problema planteado.

Tercera: Visualización y análisis previo

Esta etapa debe garantizar que los datos están correctamente tratados para garantizar que no hay errores. Además, debe verificarse que están en el formato adecuado para ser utilizados correctamente por los métodos que debamos utilizar.

Esta es una etapa que puede tener una clara incidencia en el tiempo total del proyecto, hasta llegar a ocupar más del 50 % de este.

Una vez que disponemos de todos los datos en el formato correcto, se puede empezar a realizar un análisis exploratorio de estos:

- Distribución de los datos.
- Identificación de medidas extremas.
- Visualización de relaciones entre variables.
- Evolución de características de tipo temporal.
- Estructura de los datos y/o variables utilizadas.
- Cálculo de estadísticos de interés.

Todas estas operaciones se pueden realizar sobre algunas variables (aquellas que determinamos por su importancia) o con todo el conjunto de las variables de nuestros datos.

Cuarta: Construcción de los modelos tipo

Es recomendable aplicar diferentes modelos a nuestro problema. Para cada uno de estos modelos se realizará su estimación y verificación. Entre los modelos que podemos aplicar tenemos:

- Clasificación por Naive Bayes
- Regresión lineal
- Regresión logística
- Redes neuronales y métodos de *deep learning*
- Agrupación de observaciones
- Recomendaciones o reglas de asociación

Cada modelo tendrá su especificidad y es nuestra responsabilidad entender bajo qué condiciones se pueden aplicar.

Quinta: Evaluación y selección del modelo

Para cada modelo en consideración se evaluará su rendimiento y se seleccionará aquel que mejor rendimiento ofrezca. Existe una gran lista de posibles medidas de evaluación del modelo. Según el tipo de aprendizaje y de la técnica que vayamos utilizar, usaremos unas u otras.

Posible lista de medidas:

- matriz de confusión
- roc
- precision y recall
- AIC, BIC, Cp
- MSE, RMSE, R^2, R^2_{adj}, FIV

En algunas ocasiones la técnica utilizada permite generar una buena selección del modelo. Por ejemplo, en el caso de regresión, el modelo nos puede indicar que deberíamos descartar una serie de atributos que de entrada imaginábamos que podían tener efecto en la explicación de la variable respuesta.

La evaluación del modelo puede desarrollarse desde un prisma más estadístico o desde una vertiente más de *machine learning*. En los siguientes puntos analizaremos ambas estrategias.

Sexta: Interpretación y comunicación del modelo

En esta etapa entendemos que tenemos un único modelo seleccionado. Este modelo representa la mejor opción disponible y es el que debe ser aplicado para resolver nuestro problema o posteriormente puesto en explotación una vez reciba el OK de dirección.

Metodología CRISP - DM

En ciertos entornos y en algunas industrias se sigue la metodología denominada CRISP (*cross-industry standard process for data mining*), que en líneas generales se parece a la descrita en este subapartado.

La facilidad de un modelo para ser explicado es de vital importancia en multitud de ámbitos. No solo queremos buenas soluciones, sino que además buscamos aquellos modelos que permitan esta interpretación. Nos encontraremos con esta disyuntiva en los modelos de *deep learning*, donde los resultados son ciertamente muy buenos, pero carecen de la capacidad de poder dar una interpretación de cómo toman la decisión.

Enlace de interés

Para más información podéis consultar el siguiente enlace: https://www.the-modeling-agency.com/crisp-dm.pdf

Otro punto prioritario es la forma como se comunica el proyecto. Qué tipo de mensaje, de gráficos, de palabras van a definir nuestro proceso comunicativo del proyecto.

1.3. Métodos de aprendizaje

Los métodos de *machine learning* se pueden clasificar en función de alguna de las características que ocurren en los procesos de aprendizaje. Una de estas agrupaciones ampliamente utilizada corresponde a cómo se produce el aprendizaje. Existen los siguientes subgrupos que detallamos a continuación:

- Métodos supervisados
- Métodos semisupervisados
- Métodos no supervisados
- Métodos reforzados

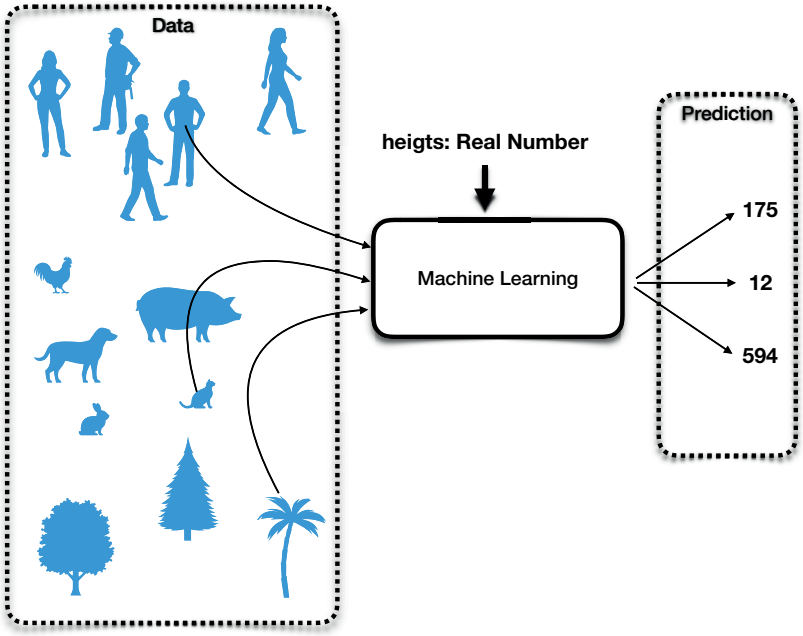
1.3.1. Métodos de aprendizaje supervisado

Los métodos de aprendizaje supervisado se caracterizan por que en el conjunto de datos hay una información adicional que será la que va a permitir el ajuste del modelo a estos datos. Esta información adicional la podemos imaginar como una etiqueta asociada a cada una de las observaciones que tenemos disponibles. De esta manera, el algoritmo puede corregir y adaptar su configuración gracias a la información contenida en estas etiquetas.

Si el contenido de las etiquetas de nuestra variable *target* son valores numéricos (reales), es decir, representa un valor numérico, estamos ante un problema de regresión. Imaginad que son alturas de diferentes objetos, nuestro problema puede ser cómo a partir de los datos disponibles (peso, tipo objeto, color...)

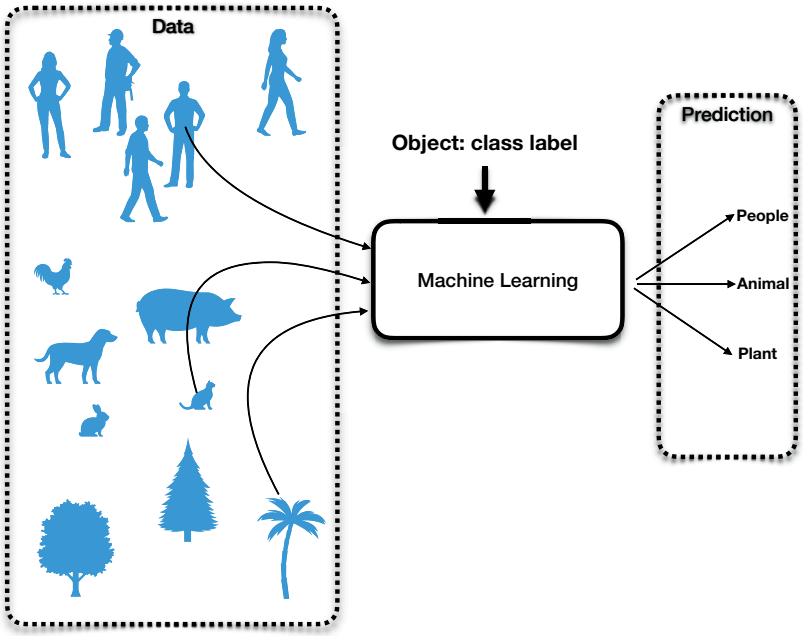
podemos predecir su altura. En el siguiente gráfico se muestra este concepto de método supervisado de regresión (ver figura 1).

Figura 1. Método supervisado - regresión



La otra situación que podemos encontrar en el contenido de la variable *target* es una categoría (un factor). En el caso de que las etiquetas representen una categoría, entonces estamos ante un problema de clasificación.

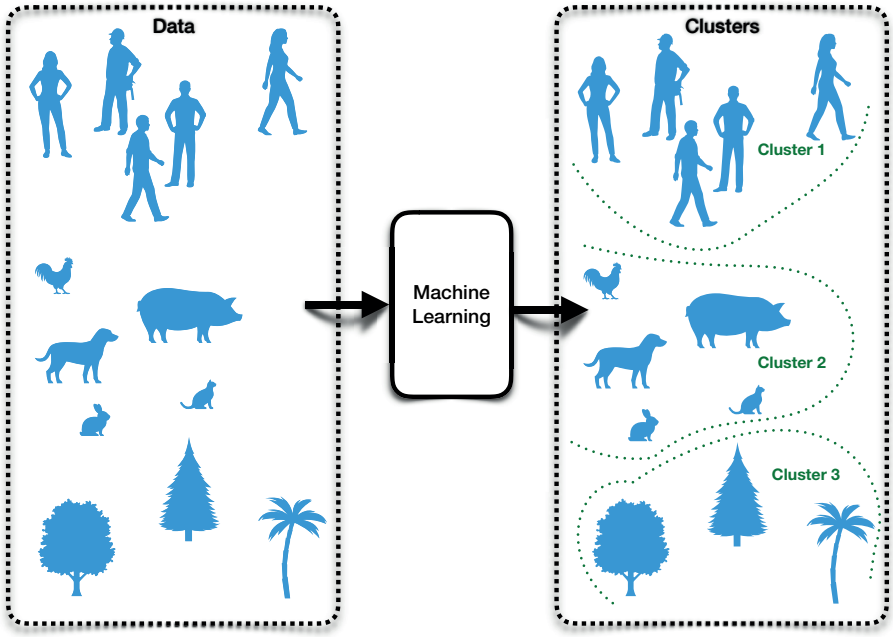
Figura 2. Método supervisado - clasificación



1.3.2. Métodos de aprendizaje no supervisado

En el caso de los métodos no supervisados, no se dispone de información adicional (no tenemos ninguna variable que podamos identificar como *target*). En estas situaciones donde todas las variables tienen la misma importancia y se persigue el objetivo de agrupar, los métodos no supervisados son la solución. Esta aproximación permite obtener conocimiento de la estructura original de los datos realizando agrupaciones de aquellas observaciones similares (se asignarán al mismo clúster), mientras que las observaciones no similares serán asignadas a clústeres distintos.

Figura 3. Método no supervisado - clúster



Métodos de aprendizaje reforzado

Este tipo de metodología no será tratada, pero ha tenido y tiene mucho interés en la comunidad científica por su capacidad de adaptarse a problemas complejos. Básicamente, se basa en programar un agente para que este aprenda en un entorno (que representa el problema) realizando diferentes operaciones y viendo y evaluando los resultados. De esta manera, puede aprender estrategias óptimas para el problema en cuestión.

¿Qué tipos de problemas están indicados para estos métodos? Se pueden resolver una gran variedad de problemas diferentes mediante *reinforcement learning* (RL). Debido a que los agentes de RL pueden aprender sin la supervisión de un experto, el tipo de problemas que mejor se adapta a RL son problemas complejos donde no parece haber una solución directa o fácilmente programable:

- ***Game playing***: identificar las estrategias de movimiento o acciones que realizar en un juego. La aproximación RL permite eliminar la necesidad de especificar reglas manualmente; los agentes aprenden simplemente jugando el juego. Para juegos de dos jugadores como el Go, los agentes pueden entrenarse jugando contra otros jugadores (humanos o no).

- ***Complex control problems***: la resolución de problemas de control, como la programación del ascensor de un edificio, o la resolución de sistemas de control para navegación aérea, o la resolución de sistemas de fabricación automatizados. Todas estas situaciones se caracterizan por representar problemas de control difíciles y no lineales.

1.4. *Overfitting* y *underfitting*

Debemos tener precaución en la aplicación de los métodos de ML para no incurrir en problemas de *overfitting* o *underfitting*. Estas situaciones deben evitarse para garantizar que el modelo generaliza suficientemente.

Esta situación queda de manifiesto en las siguientes representaciones gráficas. En la figura 4 podemos apreciar unos datos y un primer ajuste claramente insuficiente por no representar lo que los datos muestran. El ajuste lineal no generaliza correctamente. En la figura 5 podemos entender que los dos ajustes parecen razonables. Es decir, el modelo parece que se ajusta a los datos y obtiene una buena generalización. La siguientes representaciones que configuran ajustes polinómicos de orden 4, 5, 6 y 7, respectivamente, ofrecen visiones cada vez más exageradas de *overfitting*.

Figura 4. Datos que ajustar y modelo lineal (ejemplo de *underfitting*)

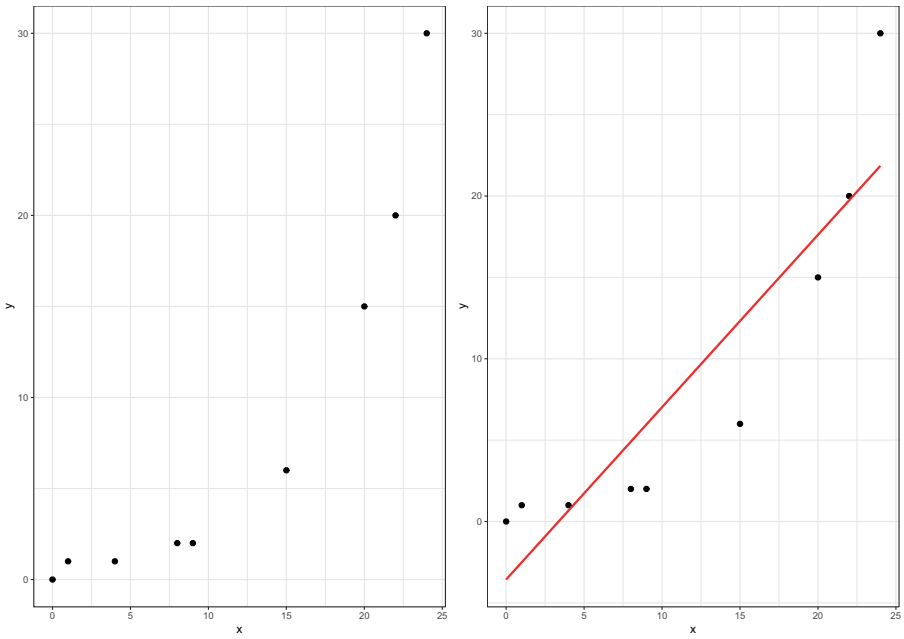
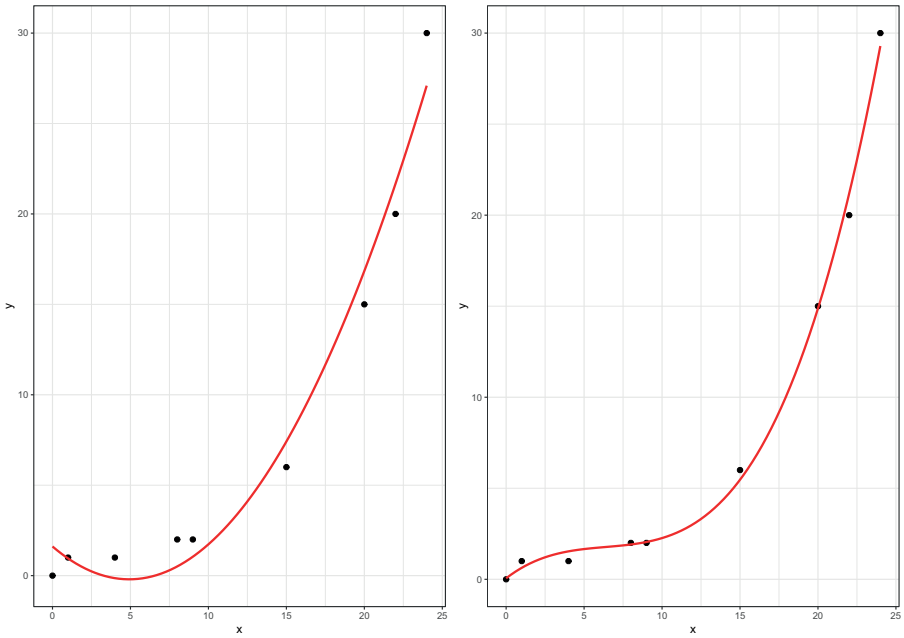


Figura 5. Modelo polinómico de orden 2 y 3 (parecen buenos ajustes)



La situación visualizada no presenta dudas en la interpretación. Hay pocos datos y el efecto es claramente visible. Cuando estemos delante de problemas con millones de datos donde la representación gráfica no sea posible, debemos garantizar que los modelos no están en situación de *overfitting*, lo que implicaría que el modelo no será capaz de generalizar.

Figura 6. Modelo polinómico de orden 4 y 5 (parecen ajustes muy forzados, casi *overfitting*)

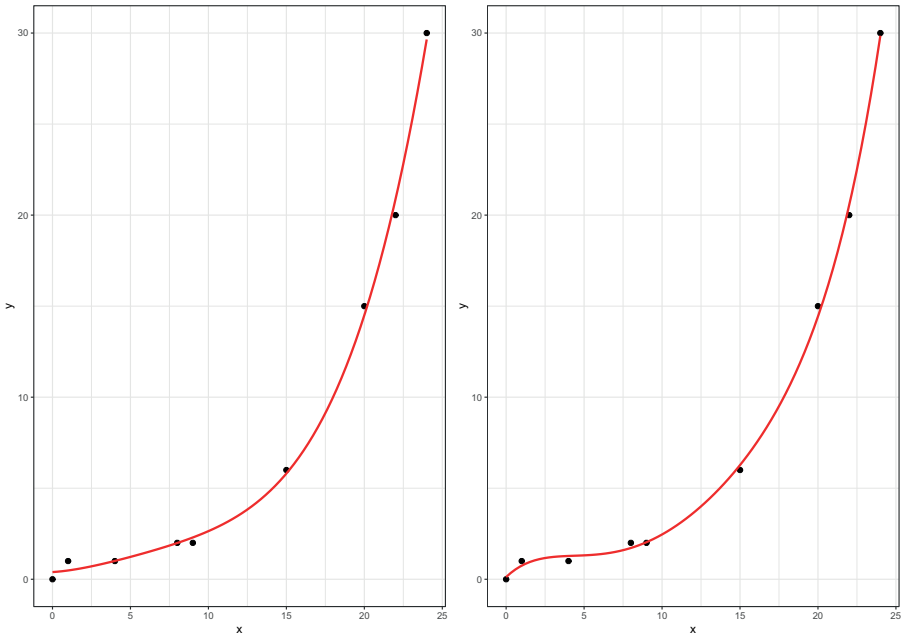
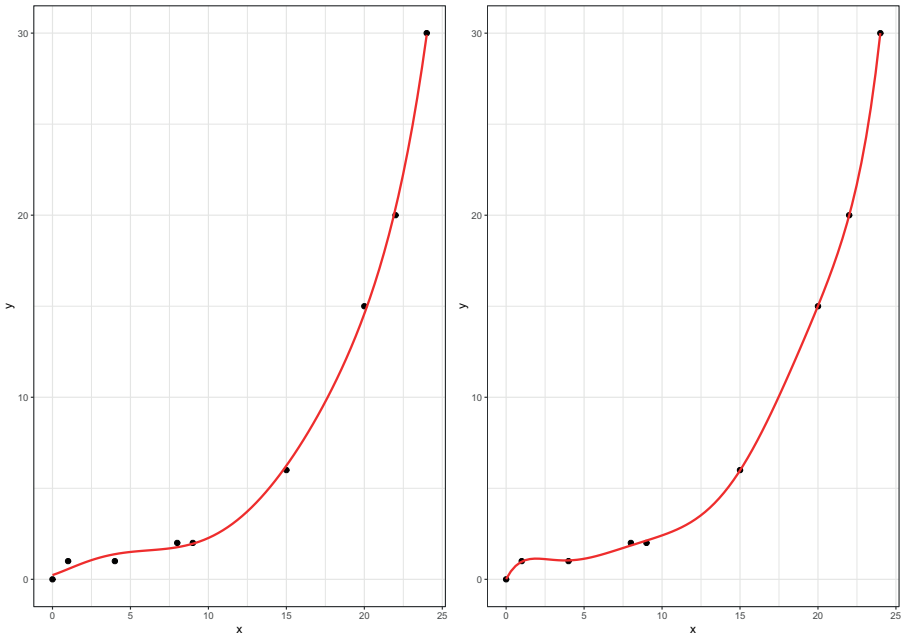


Figura 7. Modelo polinómico de orden 6 y 7 (*overfitting*)



¿Cómo podemos evitar estos problemas? Con métodos que utilicen dos conjuntos de datos: uno para el entrenamiento y otro para el test. Muchas de las metodologías de ML se desarrollan pensando en cómo evitar estos problemas. En regresión y *deep learning* se diseñan soluciones vía regularización para evitar el *overfitting*.

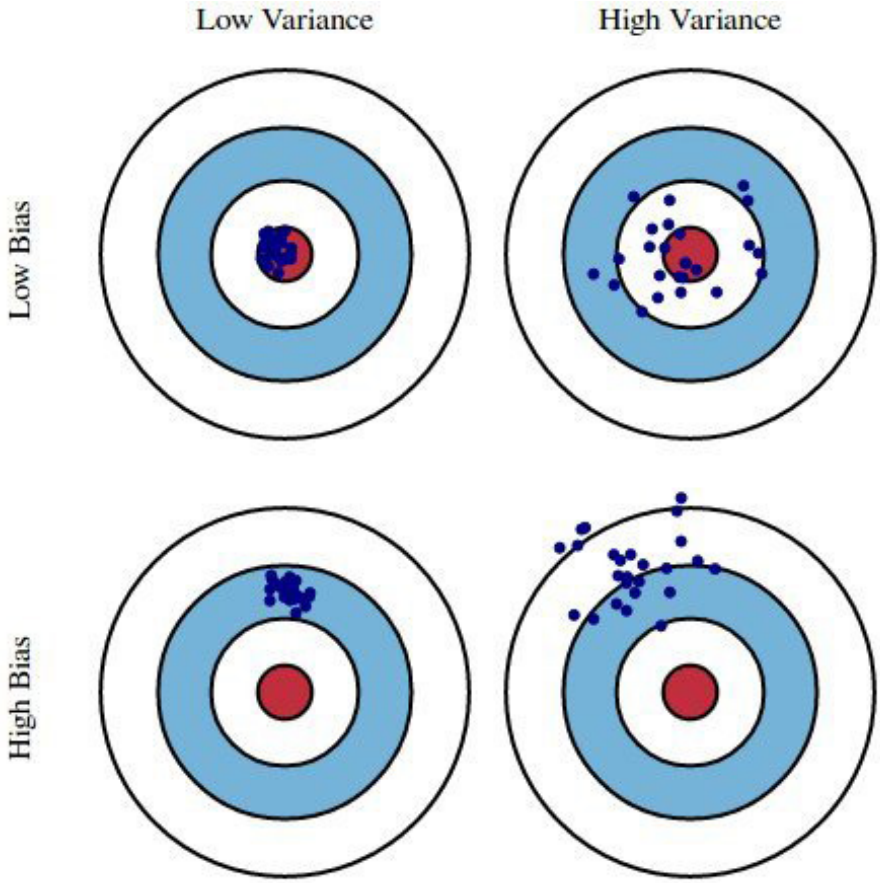
Gráficamente podemos entender el modelo como un sistema que intenta alcanzar un objetivo (diana). En esta diana se dan dos tipos de problemáticas:

- Sesgo (*bias*)
- Varianza

¿Por qué aparecen estas problemáticas? (ver figura 8)

Al analizar la predicción de un modelo, es importante comprender los dos tipos de errores que podemos cometer (sesgo y varianza). Existe una compensación entre la capacidad de un modelo para minimizar el sesgo y la varianza. Entender adecuadamente estos dos mecanismos de generación de errores nos ayuda a construir modelos precisos y a evitar *overfitting*. El concepto de sesgo se visualiza como la diferencia entre la predicción promedio de nuestro modelo y el valor correcto que estamos tratando de predecir (ver de nuevo la figura 8). El concepto de varianza como representación de la variabilidad de la predicción nos indica la dispersión de nuestros datos. Si tenemos una variación alta, el modelo no generaliza correctamente.

Figura 8. Representación visual del efecto del sesgo y la varianza en los sistemas de clasificación



Observad que, en el aprendizaje supervisado, el ajuste excesivo ocurre cuando nuestro modelo captura el ruido junto con el patrón subyacente en los datos. Si sobreentrenamos a nuestro modelo, tendremos bajo sesgo y varianza alta.
Fuente: https://www.kdnuggets.com/2016/08/bias-variance-tradeoff-overview.html

1.5. *Training* y *testing*

La única forma de garantizar que un modelo generaliza de manera aceptable es si el modelo se pone a prueba con datos no vistos. La separación de nuestros datos en conjuntos de *training* (entrenamiento) y *testing* (pruebas) es una parte importante de la evaluación de modelos en ML.

Analicemos los conjuntos de datos:

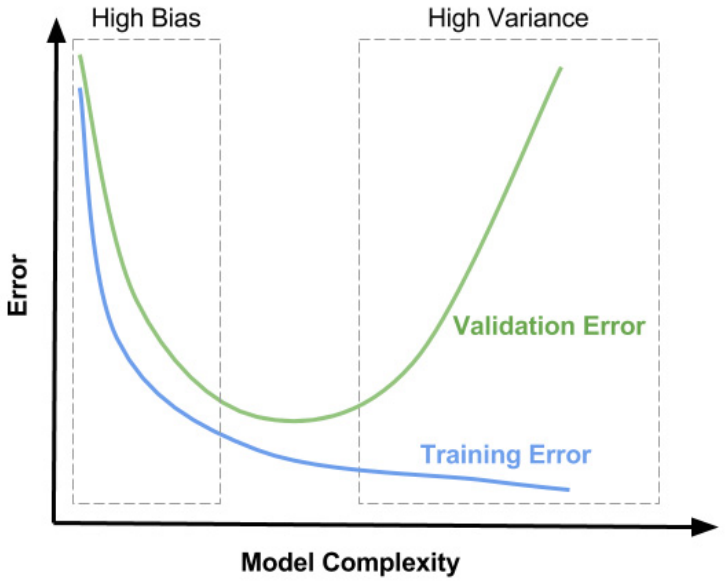
- ***Training***: Corresponde a la muestra de datos utilizada para ajustar el modelo. Puede representar entre un 50 y un 80 % del total de los datos disponibles. En este conjunto de datos se realiza la estimación del modelo, pero no se evalúa el modelo con este conjunto de datos.

- ***Testing***: Corresponde al subconjunto de datos utilizado para proporcionar una evaluación imparcial de nuestro modelo. La evaluación garantiza que las medidas de rendimiento se obtienen con observaciones que el modelo no ha visto nunca.

En ocasiones se utiliza un tercer *data set*, denominado Validation. Se utiliza para reajustar el modelo seleccionado. Este reajuste del modelo será el que posteriormente se utilizará con los datos de test.

Consideraciones importantes: el conjunto de los datos de test proporciona la mejor estrategia para evaluar el modelo. Solo se utiliza una vez que el modelo está completamente ajustado. El conjunto de test es generalmente el que se usa para evaluar los modelos en una competición (por ejemplo, las que se desarrollan en Kaggle). El error que se comete al ajustar el modelo con el conjunto de datos test corresponde a un error de generalización (ver figura 9).

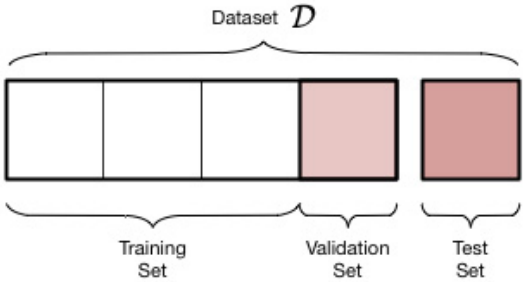
Figura 9. Efecto de los errores sobre el conjunto de datos de *training* y de *testing* y el efecto en el sesgo y la varianza



Fuente: ver AA

La división del conjunto de datos se puede visualizar en la siguiente representación (ver figura 10).

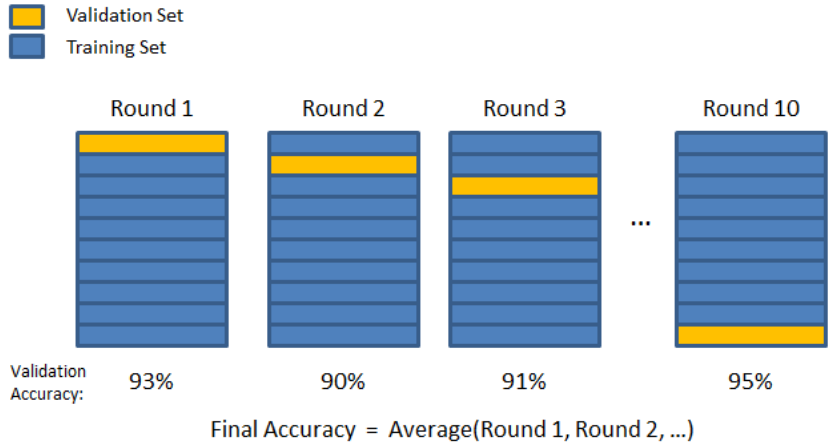
Figura 10. Partición del conjunto de datos en *training*, *validation* y *testing*



Fuente: ver AA

En ocasiones, para evitar perder parte de los datos que quizá podrían ayudar en la etapa de *training* y que se utilizan para el *testing*, conviene definir una estrategia de actuación diferente. Esta estrategia se conoce como Cross-Validation. La idea es hacer k pliegues en el conjunto de datos de modo que $(k-1)$ se utilizan para el *training* y uno para el *testing* (ver figura 11). El resultado que se ofrece en estas situaciones es el promedio de los valores de rendimiento de cada uno de los resultados sobre el conjunto test.

Figura 11. K fold cross-validation



Fuente: https://bit.ly/2IlpQOq

1.6. *Evaluation*

La mayoría de los modelos de clasificación utilizan una matriz de confusión. De los resultados obtenidos en la matriz de confusión se pueden extraer los siguientes indicadores:

$$\text{Sens} = \text{True Positive Rate} = \frac{\text{TP}}{\text{P}} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

$$\text{Spec} = \text{True Negative Rate} = \frac{\text{TN}}{\text{N}} = \frac{\text{TN}}{\text{TN} + \text{FP}}$$

$$\text{Prev} = \frac{\text{P}}{\text{Total Obs}} = \frac{\text{TP} + \text{FN}}{\text{Total Obs}}$$

Figura 12. Confusión Matrix

		Actual	
		False (0)	**True** (1)
Predicted	**False** (0)	True Negative (**TN**)	False Negative (**FN**)
	True (1)	False Positive (**FP**)	True Positive (**TP**)

Podemos obtener los valores de sensibilidad, especificidad, prevalance y otros asociados a la matriz de confusión muy fácilmente utilizando la función confusionMatrix de la librería caret.

Otras de las medidas que podemos obtener son las siguientes:

- Accuracy: ¿en total, cuántas veces el clasificador es ok? (TP+TN)/total.
- Misclassification Rate: ¿en total, cuántas veces el clasificador es ko? (FP+FN)/total.
- True Positive Rate: cuando en realidad es yes, ¿cuántas veces predecimos el yes? TP/actual yes. (Atención: también se denomina Sensitivity o Recall).
- False Positive Rate: cuando en realidad es no, ¿cuántas veces predecimos el yes? FP/actual no.
- True Negative Rate: cuando en realidad es no, ¿cuántas veces predecimos el no? TN/actual no. (Atención: también se denomina Specificity = 1 - FPR).
- Precision: cuando predice el yes, ¿cuántas veces es ok? TP/predicted yes.
- Prevalence: ¿cuántas veces el yes en realidad ocurre? Actual yes/total.
- Null Error Rate: ¿cuántas veces no predeciremos correctamente si utilizamos como valor que predecir el valor de la clase más numerosa?
- F Score: es una media harmónica de True Positive Rate (recall) y Precision.
- ROC Curve: es un gráfico habitualmente usado para representar el rendimiento de un clasificador sobre todos los umbrales posibles. Se genera al trazar la True Positive Rate (eje y) contra la False Positive Rate (eje x) a medida que varía el umbral para asignar observaciones a una clase dada.

2. Métodos de regresión y de clasificación

2.1. Introducción

Como hemos visto, los métodos de aprendizaje supervisado se pueden dividir en métodos de regresión y en métodos de clasificación. La principal diferencia entre las dos aproximaciones es el tipo de previsión que realizamos: si la predicción es una etiqueta, entonces tenemos un problema de clasificación. Si la predicción corresponde a un valor numérico, entonces estamos ante un problema de regresión.

A continuación vamos a detallar los métodos de regresión estándar y los más modernos asociados a la regresión penalizada (con diversas aplicaciones en el entorno de grandes conjuntos de datos). Finalizaremos con dos tipos de métodos que se aplican en una tipología de modelos muy habituales en noticias y medios de comunicación (con un gran impacto en aplicaciones reales), como son los métodos de redes neuronales y los de *deep learning*.

2.2. Métodos de regresión

Los métodos de regresión tienen como principal objetivo predecir una variable respuesta (que habitualmente escribimos como Y) a partir de un conjunto de variables explicativas (que representamos con X_1,X_2,...,X_k).

Ejemplos que podemos describir:

- Predecir el número de transistores integrados en un chip para el año 2020.
- Predecir el consumo de un vehículo a partir de su potencia.
- Predecir los precios de venta de viviendas a partir de los metros cuadrados y el número de habitaciones.
- Predecir el mejor precio de venta para coches de segunda mano.
- Predecir la probabilidad de fallo de un sistema de combustión en una lanzadera espacial.

Todos estos ejemplos tienen en común que, a partir de un conjunto de variables explicativas, estas nos permiten obtener una estimación sobre la variable objeto de estudio o de interés (*target* variable).

2.2.1. Objetivos de los modelos de regresión

Podemos resumir de una forma concisa los objetivos de los modelos de regresión.

Definición: Los modelos de regresión tienen como objetivo caracterizar la relación estadística de la variable respuesta Y con las explicativas X.

A partir de este modelo se pueden resolver dos problemas típicos:

1) Problemas de predicción: cómo a partir del conocimiento de la variable explicativa X predecimos la variable objetivo o respuesta Y.

2) Problemas de inferencia: cómo cambia el valor previsto de Y si cambiamos algunas de las columnas de X.

El modelo de regresión lineal plantea una relación lineal entre las variables explicativas y la respuesta:

$$Y = \beta_0 + \beta_1 X_1 + ... + \beta_n X_n + \epsilon$$

Si observamos el modelo, aparece una parte determinista descrita por la recta (ecuación matemática de un hiperplano) y otra no determinista ϵ, que corresponde al error.

2.2.2. Hipótesis del modelo de regresión y etapas de análisis

La hipótesis básica del modelo de regresión es que existe una relación lineal entre dos variables: una variable Y, que llamaremos variable respuesta, y otra variable X, que llamaremos variable independiente o explicativa. Ambas variables se relacionan a través de la ecuación de una recta.

$$Y = \beta_0 + \beta_1 X_1 + \epsilon$$

En su formulación más sencilla, el modelo relaciona la variable **Y** con una función lineal de la variable **X** más un término de error ϵ. Suponemos que cada

error se distribuye de forma idéntica e independiente como una distribución normal con media cero y con una desviación σ^2. Podemos escribirlo como:

$$\epsilon_i DII \sim Norm(0,\sigma^2)$$

La ecuación de la recta tienes los parámetros β_0, que corresponde a la ordenada en el origen, y el parámetro β_1, que se interpreta como la pendiente de la recta. Otro parámetro que queda asociado con el modelo es la desviación del error σ.

Desde una perspectiva del análisis estadístico, las etapas que deberíamos seguir para modelar nuestros datos como un modelo de regresión lineal simple serían las siguientes.

Paso uno: análisis exploratorio de los datos

La idea de este apartado es contrastar que tiene sentido aplicar un modelo lineal entre ambas variables. Para verificar la idoneidad del modelo lineal, podemos realizar alguno de estos pasos (o todos):

- Verificación de la distribución de los datos asociados a la variable respuesta y a la variable independiente.
- Diagrama entre la variable respuesta y la variable explicativa.
- Cálculo del coeficiente de correlación entre la variable X e Y.

Se pueden obtener medidas numéricas del grado de asociación:

- El test de rango de Spearman es una medida no paramétrica de la asociación monotónica entre dos variables numéricas.

- El test de correlación de rango de Kendall es otra medida no paramétrica de la asociación, basada en la concordancia o discordancia de los pares *x*,*y*.

```
with(mtcars,cor(mpg,hp))
with(mtcars,cor.test(mpg,hp,method="spearman"))
with(mtcars,cor.test(mpg,hp,method="kendall"))
```

Valores del p-value inferiores a 0.05 resuelven el contraste en la dirección de que hay evidencias para afirmar que existe correlación lineal entre las variables.

Paso dos: ajuste del modelo lineal

El ajuste del modelo que implica la estimación de los parámetros asociados al modelo. Esta etapa se puede realizar con la instrucción base de R *lm* o también con la función más general *glm*.

El resultado de estas dos funciones es la lista estimada de los parámetros del modelo.

Como R permite especificar fórmulas estadísticas, la propuesta de sintaxis es muy simple y coherente:

$$\text{variable respuesta} \sim \text{variables explicativas o predictoras}$$

La interpretación que debemos hacer de $\sim$ es: *es modelado como una función de*. Por tanto, podemos especificar en el caso del fichero mtcars:

```
mod1 <- lm(mpg ~ hp, data=mtcars)
```

que leemos como: estimar un modelo de regresión lineal (*lm*) donde la variable respuesta es *mpg* (consumo) y la variable explicativa es *hp*.

Formalmente, estamos modelando la siguiente situación:

$$Y_{mpg} = \beta_0 + \beta_1 X_{hp} + \epsilon$$

El uso de la notación $X_1 + X_2$ se interpreta dentro de la fórmula como añadir otro predictor. Por ejemplo:

```
mod1 <- lm(mpg ~ hp + disp, data=mtcars)
```

indica que estamos modelando la variable consumo, como una función lineal de la potencia y del volumen del motor. Matemáticamente, se expresaría:

$$Y_{mpg} = \beta_0 + \beta_1 X_{hp} + \beta_2 X_{disp} + \epsilon$$

- + para añadir variables $+wt$
- – para eliminar variables $-wt$
- : para añadir interacción entre dos variables $wt : hp$
- $*$ para incluir en el modelo las variables especificadas y la interacción entre ellas $wt * hp$

- | para condicionar $wt|am$
- ^ para incluir interacciones hasta el exponente indicado $(hp + wt + disp)\hat{}3$
- I para incluir la variable como se indica entre paréntesis $I(hp + wt)$
- 1 para hacer referencia a la ordenada en el origen –1 indica regresión a través del origen (0,0)
- . para indicar el conjunto de todas las variables

Por ejemplo, estos tres modelos darán el mismo resultado porque hacen referencia al mismo tipo de variables regresoras:

```
mod1 <- lm(mpg~(hp+wt+disp)^3,data=mtcars)
mod1 <- lm(mpg~(hp*wt*disp),data=mtcars)
mod1 <- lm(mpg~hp+wt+disp+hp:wt+hp:disp+wt:disp+hp:wt:disp,data=mtcars)
```

es decir, modelan

$$Y_{mpg} = \beta_0+\beta_1 X_{hp}+\beta_2 X_{wt}+\beta_3 X_{disp}+\beta_4 X_{hp}X_{wt}+\beta_5 X_{hp}X_{disp}+\beta_4 X_{wt}X_{disp}+\beta_4 X_{hp}X_{wt}X_{disp}+\epsilon$$

Paso tres: resolución de cuestiones de inferencia

En este apartado nos podemos preguntar si los parámetros de la ecuación de la recta son estadísticamente significativos. Este contraste de hipótesis se puede resolver de manera sencilla observando los valores p value.

Para el parámetro correspondiente a la ordenada en el origen, si el contraste de hipótesis nos da evidencias de que no puede ser cero, la interpretación es que el modelo debe contener este parámetro. Para el parámetro de la pendiente se realiza el mismo contraste de hipótesis, esperando que tengamos evidencias de que no es cierta la hipótesis nula y de que, por tanto, sí que hay pendiente en el modelo.

No es habitual calcular la significación para la desviación. Sí que desde una perspectiva estadística se calcula si los coeficientes de la recta en su totalidad son significativos. Este contraste se realiza con la información facilitada por el estadístico F.

La información que habitualmente obtenemos de la etapa de estimación permite realizar inferencia sobre los parámetros del modelo.

```
Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept)  5.255e+01  9.596e+00   5.476 1.25e-05 ***
```

```
hp          -1.426e-01  8.282e-02  -1.722   0.0980 .
wt          -7.511e+00  3.722e+00  -2.018   0.0549 .
disp        -6.416e-02  9.324e-02  -0.688   0.4980
hp:wt        2.614e-02  2.936e-02   0.890   0.3822
hp:disp      3.297e-04  4.180e-04   0.789   0.4379
wt:disp      1.237e-02  2.598e-02   0.476   0.6382
hp:wt:disp  -6.295e-05  1.203e-04  -0.523   0.6055
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 2.269 on 24 degrees of freedom
Multiple R-squared:  0.8903,Adjusted R-squared:  0.8583
F-statistic: 27.82 on 7 and 24 DF,  p-value: 4.908e-10
```

En esta salida observamos los valores correspondientes a la estimación de los parámetros de la recta (hiperplano). La interpretación línea de (intercept):

$$\hat{\beta}_0 = 52{,}55; ; S_{\hat{\beta}_0} = 9{,}5; ; \frac{\hat{\beta}_0}{S_{\hat{\beta}_0}} = 5{,}47$$

De forma directa se visualiza la última columna para verificar si el parámetro es estadísticamente significativo. Observad que en este ejemplo únicamente la ordenada en el origen es significativa. Los parámetros asociados a las variables *hp* y *wt* están ligeramente por encima del valor habitual de corte (0.05).

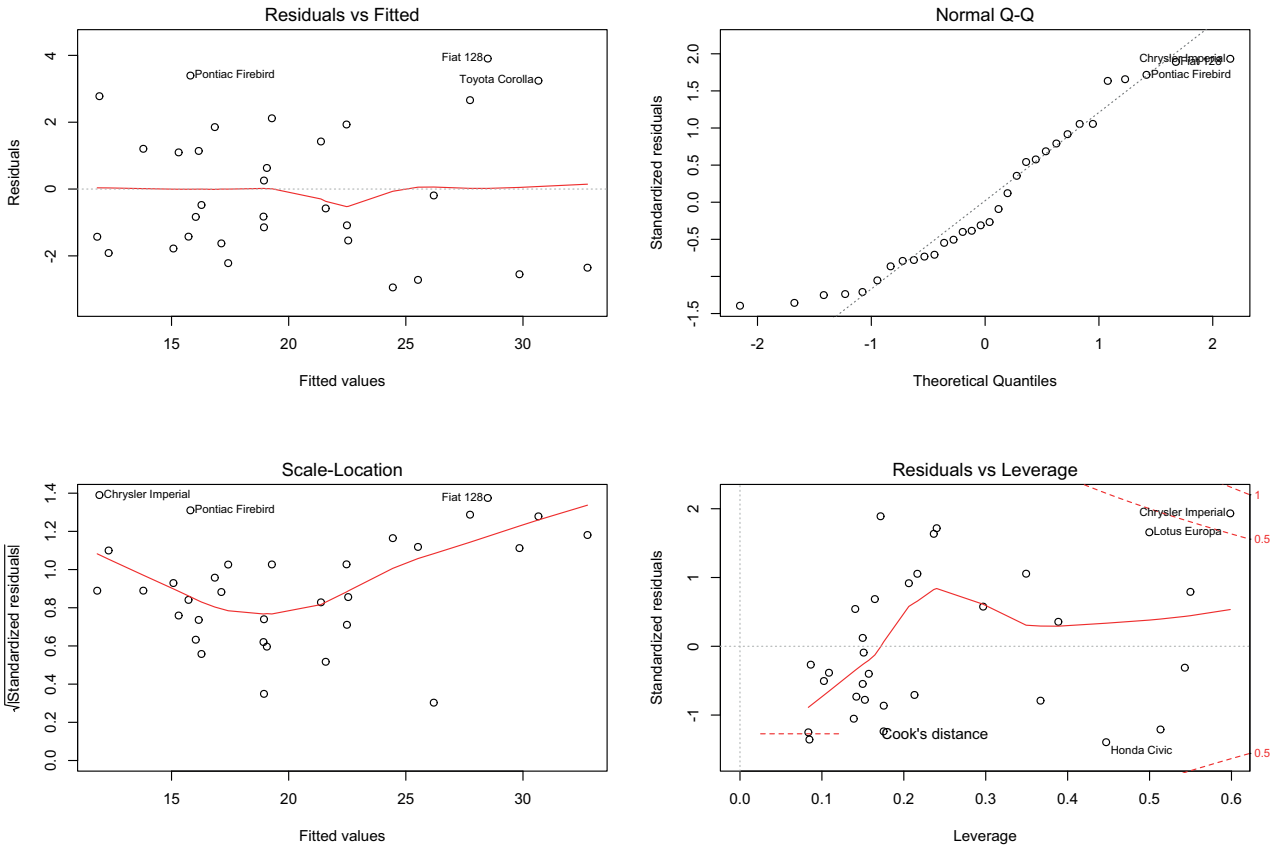
Otros valores que obtenemos son la estimación de la desviación del error: $\hat{\sigma_\epsilon}$, que nos da 2.269. Tenemos también la medida de R^2, de R^2_{adj} y del estadístico F, que se asocia al efecto colectivo de todas las variables predictoras sobre la variable de respuesta.

Paso cuatro: verificación de los residuos

Esta verificación se puede realizar de una forma visual o a través de diferentes medidas estadísticas. La manera más directa y simple es representar un diagrama donde en el eje de las ordenadas aparece la variable del error y en el eje de las abscisas situamos la estimación de la variable respuesta. Con este gráfico podemos observar si los errores se distribuyen de forma independiente e idéntica alrededor del cero, si la varianza es constante, si no existe correlación entre los errores y si la distribución de los errores sigue una distribución normal. En la figura 13 se observa el resultado de visualizar los residuos del modelo ajustado.

```
mod1 <- lm(mpg~(hp*wt*disp),data=mtcars)
plot(mod1)
```

Figura 13. Gráficos asociados a la etapa de análisis de los residuos



¿Podemos visualmente identificar y garantizar que las hipótesis del modelo se cumplen?

Paso cinco: evaluación del modelo

Si tenemos intención de ajustar diferentes modelos, puede ser interesante obtener medidas de la bondad del ajuste del modelo:

- Criterio de información de Akaike
- Criterio de información bayesiano
- Coeficiente Cp (es equivalente al AIC en el caso de regresión lineal gaussiana)
- Coeficiente R cuadrado y coeficiente R cuadrado ajustado
- Coeficiente VIF (*variance inflation factor*)

Criterio de información de Akaike:

$$AIC(M) = -2log\mathcal{L}(M) + 2p(M)$$

donde $\mathcal{L}(M)$ representa la función de verosimilitud del modelo M evaluada en el valor MLE (*maximum likelihood estimator*), y $p(M)$ representa el número de predictores del modelo (grados de libertad del modelo).

Criterio de información bayesiano:

$$BIC(M) = -2log\mathcal{L}(M) + p(M)logn$$

En el caso de tener un modelo con k parámetros: entonces $p(M) = k$, que es una notación más común. Podemos interpretar AIC y BIC como estimadores de la calidad relativa de los modelos evaluados para un conjunto de datos. Ambas medidas tratan de evitar el *overfitting* introduciendo un término de penalización sobre el número de parámetros. El término de penalización es mayor en BIC que en AIC.

El coeficiente R^2 se utiliza para hacer una interpretación del porcentaje de variación explicado por el modelo, pero no es muy recomendable como medida para evaluar modelos. Es una medida totalmente errónea para usar en la selección de modelos. ¿Por qué? Una mejor alternativa es el coeficiente R^2_{adj}.

El coeficiente VIF se usa para identificar multicolinealidad. En realidad, podemos interpretar el VIF como un índice que mide cuánto aumenta la varianza de un coeficiente de regresión estimado debido a la colinealidad.

$$VIF_j = \frac{1}{1 - R^2_j}$$

Paso seis: explotación del modelo

La explotación del modelo se puede dividir en dos apartados diferenciados. El primero correspondería a la predicción de la variable respuesta de nuevas observaciones. En esta situación nos preguntamos dado un valor de x cuál va ser el valor respuesta. Obviamente, al estar modelando estadísticamente deberemos cuantificar el error asociado a esta predicción.

Al realizar predicciones debemos ser cautos al preguntarnos por la respuesta de la variable y cuándo la variable x toma diferentes valores. Si la variable x toma valores dentro del rango de valores observados, decimos que la predicción corresponde a una interpolación. Este tipo de previsiones tiene al menos la garantía de que disponemos de datos dentro del rango preguntado. Si el valor de la variable x está fuera del rango de valores de la variable, entonces decimos que estamos ante una predicción extrapolativa. En este caso, aunque obtengamos un valor de predicción, hay que ser cautelosos porque no hay datos para sustentar que el modelo es de aplicación también en esta situación.

Otro posible uso del modelo es interpretar lo que sucede por cada incremento unitario que la variable explicativa tenga. Es decir, cómo va a variar la variable respuesta por cada incremento o decremento de la variable explicativa.

2.2.3. Ejemplo de regresión simple

Vamos a desarrollar un ejemplo de análisis estadístico para resolver el problema de qué variables inciden en la variable de consumo: mpg (millas por galón) mediante un modelo de regresión lineal simple y a continuación por medio de uno múltiple.

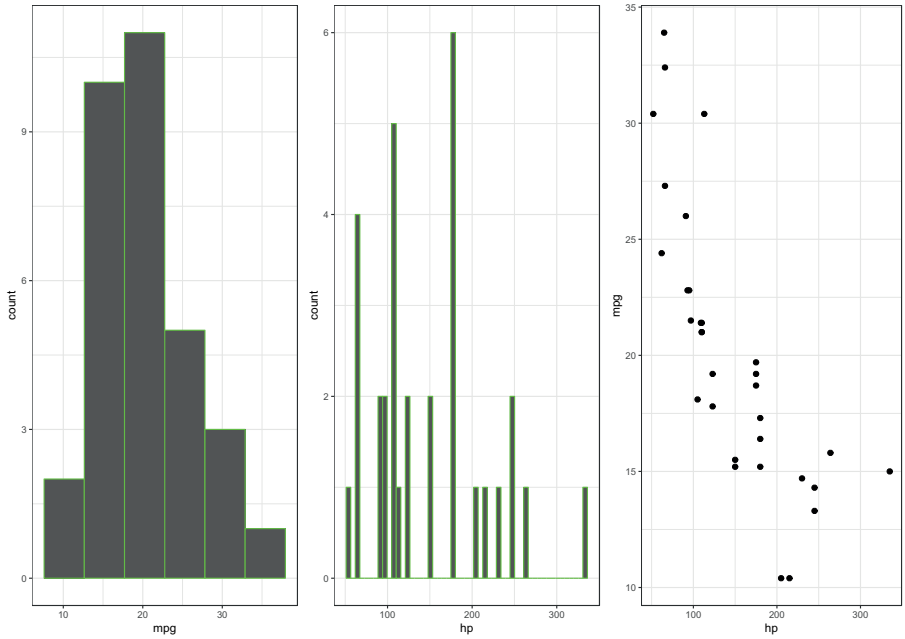
EDA sobre el problema

La parte descriptiva corresponde en este caso a dos variables: consumo como variable respuesta (mpg) y potencia como variable explicativa (hp).

```
data(mtcars)

# Descriptive Statistics
ggplot(mtcars,aes(x=mpg))+ geom_histogram()
ggplot(mtcars,aes(x=hp))+ geom_histogram()
ggplot(mtcars,aes(x=hp,y=mpg))+geom_point()
```

Figura 14. Representación de las distribuciones asociadas a la variable y = mpg a x = hp y *scatter plot* de ambas variables



Estimación de los parámetros del modelo

El procedimiento para describir un modelo en R es relativamente simple: nombre de la variable respuesta seguido de tilde ~ (ALT + 126) y nombre de la o las variables explicativas tal como se ha comentado anteriormente.

```
mod1 <- lm(mpg ~ hp, data=mtcars)
summary(mod1)
```

Los coeficientes estimados:

	Estimate	Std. Error	t value	Pr(> |t|)
(Intercept)	30.0989	1.6339	18.42	0.0000
hp	-0.0682	0.0101	-6.74	0.0000

Los valores estimados adicionales son:

```
Residual standard error: 3.863 on 30 degrees of freedom
Multiple R-squared:  0.6024,Adjusted R-squared:  0.5892
F-statistic: 45.46 on 1 and 30 DF,  p-value: 1.788e-07
```

Inferencia parámetros del modelo

Como todos los parámetros estimados tienen valores p-values más pequeños del 5 %, podemos asegurar que son estadísticamente significativos.

En caso de que algún parámetro no estuviera por debajo del 5 %, deberíamos descartarlo y proceder a reestimar otro modelo sin la variable asociada a este parámetro.

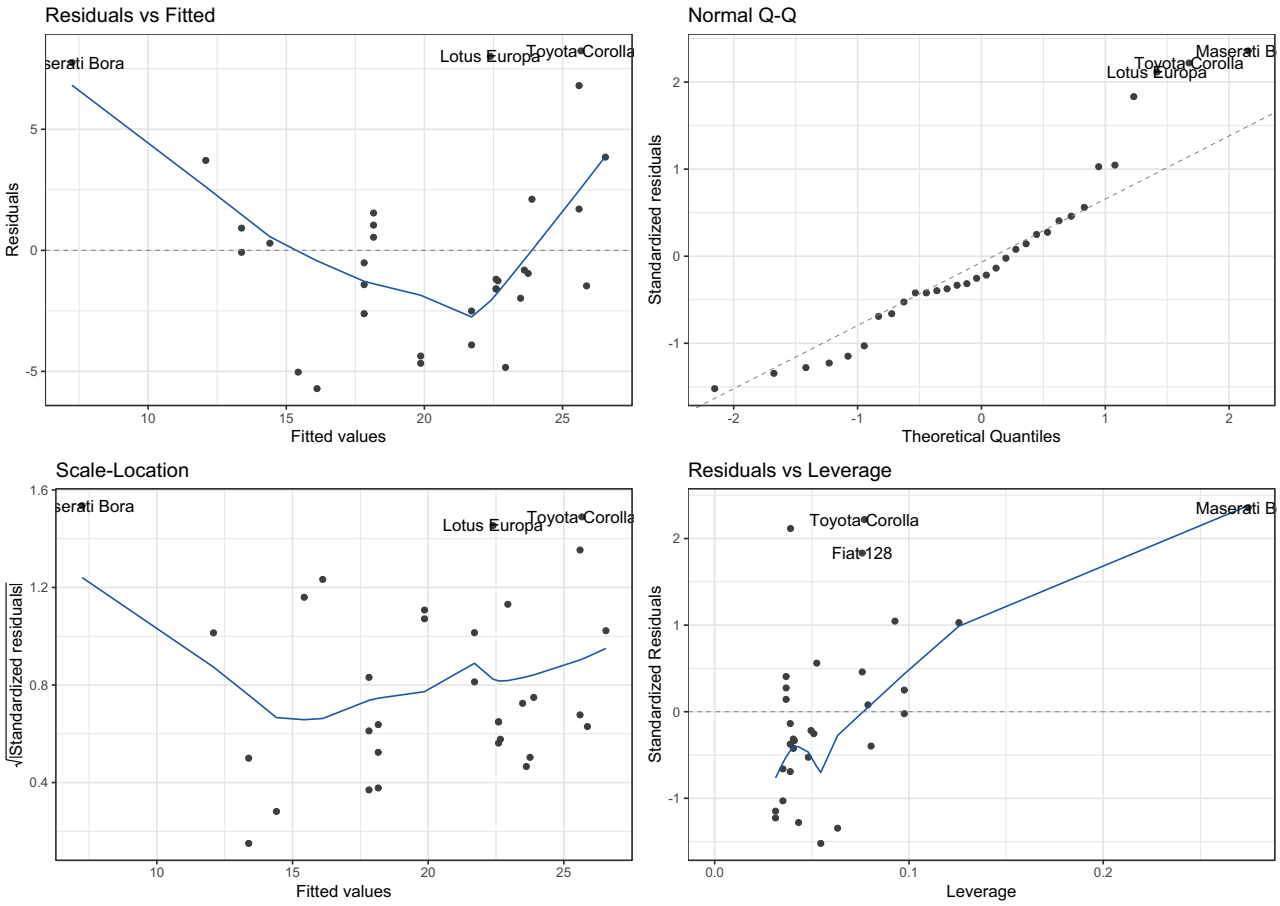
Análisis de los residuos

Si todas las condiciones siguientes se verifican con la observación de los gráficos de la figura 15:

- linealidad: la relación entre X e Y es lineal;
- homoscedasticidad: la varianza de los residuos es la misma para cualquier valor dado de X (o de $\hat{Y}$);
- independencia: todas las observaciones (errores) son independientes entre ellas;
- normalidad: Y se distribuye normalmente para cualquier valor dado de X,

entonces podemos afirmar que el modelo es estadísticamente correcto.

Figura 15. Representación de las visualizaciones del análisis de los residuos para el modelo estimado



Podemos ver que el modelo no cumple con las suposiciones que debería. Solución: cambiar de modelo, mejorarlo, etc.

En el supuesto de que consideráramos la fase de verificación correcta, podríamos realizar la fase de explotación e interpretación del modelo.

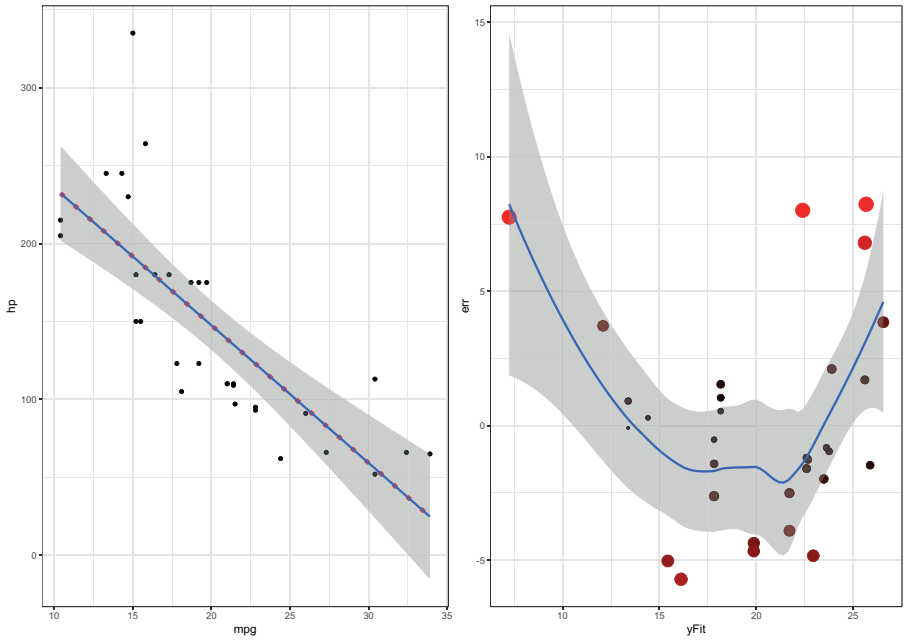
Enlace de interés

Para complementar la interpretación de los residuos y observar la creación de un gráfico paso a paso, podéis consultar: https://bit.ly/2TU6O2O

Explotación del modelo

La figura 16 visualiza dos gráficos. El primero corresponde a la estimación de la recta, así como una zona sombreada que señala el error en la predicción. Cuanto más nos separamos del centro de los datos, más error se comete. El segundo gráfico está de nuevo asociado a los errores.

Figura 16. Representación de la recta estimada entre mpg y hp, y gráfico de los errores



2.2.4. Estimación automática de modelos

En el caso de tener muchas variables disponibles, es posible utilizar alguno de los métodos que permite seleccionar el mejor modelo a partir de alguno de los criterios que anteriormente se han descrito.

Imaginad que hemos de buscar el mejor modelo para la variable *mpg* del fichero mtcars.

La librería leaps permite automatizar el proceso.

```
library(leaps)

mtFM = regsubsets(mpg~., data = mtcars)
mtFMsum = summary(mtFM)
names(mtFMsum)
# Obtenemos el número de variable óptimo utilizando Cp
which.min(mtFMsumry$cp)
# Con el valor de 3 variables obtenemos el mejor valor

# Obtenemos el número de variable óptimo utilizando BIC
which.min(reg.summary$bic)

# Visualizamos el modelo
coef(regfit.full, 3)
```

La pregunta que nos formularmos es: ¿Es este el mejor modelo que podemos encontrar?

La propuesta óptima para leaps:

```
(Intercept)          wt        qsec          am
   9.617781   -3.916504    1.225886    2.935837
```

Las librerías MASS con el método stepAIC y la librería caret también:

```
library(caret)

# Train the model
step <- train(mpg ~., data = mtcars,
                    method = "lmStepAIC",
                    trace = FALSE
)
# Model accuracy
step$results
# Final model coefficients
step$finalModel
# Summary of the model
summary(step$finalModel)
```

Observad que en este caso hay un modelo equivalente para los dos procedimientos:

```
> summary(step$finalModel)

Call:
lm(formula = .outcome ~ wt + qsec + am, data = dat)

Residuals:
    Min      1Q  Median      3Q     Max
-3.4811 -1.5555 -0.7257  1.4110  4.6610

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)   9.6178     6.9596   1.382 0.177915
wt           -3.9165     0.7112  -5.507 6.95e-06 ***
qsec          1.2259     0.2887   4.247 0.000216 ***
am            2.9358     1.4109   2.081 0.046716 *
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

```
Residual standard error: 2.459 on 28 degrees of freedom
Multiple R-squared:  0.8497,Adjusted R-squared:  0.8336
F-statistic: 52.75 on 3 and 28 DF,  p-value: 1.21e-11
```

¿Es este modelo estadísticamente correcto?

¿Qué sucede si se comparan con los resultados aplicando la estrategia de dos conjuntos de datos (*training* y *testing*) para tener medidas de estimación del modelo mucho más ajustadas? Utilizad la librería caret para seleccionar el mejor modelo utilizando como ayuda la dirección https://bit.ly/2V1ohqr.

2.2.5. Métodos de regresión logística

Si la relación entre las variables no es lineal y tenemos que la variable respuesta toma valores de 0 y 1, es decir, actúa como una variable binaria, la aplicación de una regresión lineal no dará buenos resultados. En estos casos es recomendable utilizar la formulación de R *glm*, que permite incluir otras distribuciones y optar por la regresión logística.

Tanto la regresión lineal descrita anteriormente como la regresión logística son tipos de modelos lineales generalizados que pueden representarse con la misma modelización:

$$Y \mid \mathbf{X} = \mathbf{x} \sim N(\beta_0 + \beta_1 x_1 + \ldots + \beta_{p-1} x_{p-1}, \sigma^2)$$

La regresión logística se define de manera similar a la lineal:

$$\log\left(\frac{p(\mathbf{x})}{1-p(\mathbf{x})}\right) = \beta_0 + \beta_1 x_1 + \ldots + \beta_{p-1} x_{p-1}$$

Podemos observar que el cálculo de la respuesta no es directo. Es decir, la parte izquierda de la ecuación con la variable de respuesta se conoce como *log odds*. En un contexto binario, las probabilidades son la probabilidad de que un evento positivo $Y = 1$ dividido por la probabilidad de un evento negativo $Y = 0$.

$$\frac{p(\mathbf{x})}{1-p(\mathbf{x})} = \frac{P[Y = 1 \mid \mathbf{X} = \mathbf{x}]}{P[Y = 0 \mid \mathbf{X} = \mathbf{x}]}$$

La ecuación de regresión logística garantiza que se calcula un valor entre 0 y 1. La transformación logit inversa lo garantiza, con lo que resulta en una predicción de probabilidad "directa".

$$p(\mathbf{x_i}) = P[Y_i = 1 \mid \mathbf{X_i} = \mathbf{x_i}] = \frac{e^{\beta_0+\beta_1 x_{i1}+\cdots+\beta_{p-1}x_{i(p-1)}}}{1+e^{\beta_0+\beta_1 x_{i1}+\cdots+\beta_{p-1}x_{i(p-1)}}}$$

Observad que es una predicción de probabilidad, no un valor numérico. Este valor de probabilidad debe traducirse en una predicción categórica.

Observación

Este modelo se puede extender al caso de que la variable respuesta Y tome más de dos valores.

En R usaríamos:

```
#logistic regression model
model <- glm(am ~ hp + mpg + disp, data=mtcars, family="binomial")
```

2.3. Regresión penalizada

Existen una serie de métodos para eludir los efectos adversos de alguno de los problemas de la regresión. De entre estos problemas podemos citar, entre otros, el de la multicolinealidad (variables que entre ellas están altamente correlacionadas), el de la selección de características (problema cuando hay muchas variables) o también el de la generalización. Los métodos de regresión penalizada facilitan la construcción de modelos más robustos y parsimoniosos.

¿Cuál es la forma de realizar la penalización? En función de la penalización tenemos tres métodos:

- LASSO
- Ridge
- Elastic Net

2.3.1. LASSO

La regresión LASSO (Least Absolute Shrinkage and Selection Operator) reduce los coeficientes de regresión hacia cero al penalizar el modelo de regresión con un término de penalización denominado norma L_1 (utiliza la suma del valor absoluto de los coeficientes). Esta penalización tiene el efecto de obligar a algunas de las estimaciones de los coeficientes, con una contribución menor al modelo, a ser cero (tiende a eliminarlas).

La mejor selección depende del parámetro libre λ.

El hecho de obtener coeficientes a cero permite simplificar el modelo. Además, facilita la interpretación al reducir el conjunto de predictores.

Como puntos negativos del procedimiento podemos citar que cuando el número de variables es superior al número de observaciones ($p > n$), LASSO selecciona como máximo n variables. En variables agrupadas LASSO no realiza una selección agrupada y tiende a seleccionar una variable de un grupo e ignorar las otras.

2.3.2. Ridge y Elastic Net

La regresión vía Ridge realiza la regularización tipo L_2, es decir, agrega una penalización equivalente al cuadrado de la magnitud de los coeficientes.

La aproximación Elastic Net actúa como una propuesta que combina las aproximaciones de LASSO y Ridge Regression.

Algunas diferencias que podemos identificar entre las tres variantes de regresión penalizada son:

- La aproximación LASSO hace una selección *sparse*, mientras que Ridge no.
- La aproximación Ridge reduce los dos coeficientes cuando identifica variables altamente correlacionadas. LASSO es un tanto indiferente y generalmente elige uno sobre el otro.
- La aproximación Ridge penaliza a los coeficientes β más grandes (por el efecto de la métrica L_2 en el término de penalización). LASSO penaliza de manera más uniforme.

En general, la aproximación vía Elastic Net es un compromiso entre las dos aproximaciones.

2.3.3. Regresión penalizada con R

Se utilizará la librería glmnet.

El procedimiento que seguiremos es verificar todas las aproximaciones.

- LASSO: establecer la α = 1 en la función glmnet.
- Ridge: establecer la α = 0 en la función glmnet.
- Elastic Net: establecer el valor de α óptimo utilizando validación cruzada usando la función cv.glmnet.

Es importante destacar que la librería espera objetos de tipo matricial.

```
library(glmnet)
# Datos X e Y como matrices
trainx=as.matrix(trainset)
testx=as.matrix(testset)

# Ejemplo para modelo logístico
lasso = glmnet(x=trainx,y=trainxLab, family = "binomial",alpha=1)

plot(lasso,xvar="lambda")

#coeficientes
coef(lasso,s=min(lasso$lambda))

#predicción y rendimiento
predlasso=predict(lasso, newx = testx, s = min(lasso$lambda), type = "class")

# de la libraria caret
cconfusionMatrix(predlasso,testset$Class,positive="Positive")
```

La aproximación Ridge es igual cambiando el parámetro α a 0.

```
ridge = glmnet(x=trainx,y=trainxLab, family = "binomial",alpha=0)
plot(ridge,xvar="lambda")
```

La aproximación Elastic Net:

$$\min_{\beta_0,\beta} \frac{1}{N} \sum_{i=1}^{N} w_i l(y_i,\beta_0+\beta^T x_i) + \lambda \left[(1-\alpha)||\beta||_2^2/2 + \alpha||\beta||_1\right]$$

```
elastnet = cv.glmnet(x=trainx,y=trainxLab, family = "binomial", type.measure = "class")
plot(elastnet)

#Coef optimos con mejor lambda
coef(elastnet,s="lambda.min")
```

Enlaces de interés

Para complementar la información, es aconsejable consultar esta referencia:
https://stanford.io/2voWjra
https://stanford.io/2tqLkiE

2.4. Redes neuronales

Las redes neuronales artificiales (ANN) se pueden considerar métodos de ML vagamente inspirados por las redes neuronales biológicas. La idea de una red neuronal no es más que un algoritmo de aprendizaje automático. Su principal interés es la capacidad de aprender a realizar tareas considerando ejemplos y con la ventaja de que no necesitan ser programados con ninguna regla específica. Es decir, los datos y el procedimiento hacen el trabajo de adaptarse al problema propuesto. Las opciones disponibles en R son varias.

Enlaces de interés

Considerad los dos enlaces de ejemplo:
https://bit.ly/2lZ1PPD
https://bit.ly/2VQ4Ou3

2.5. *Deep learning*

¿Por qué *deep learning*? *Deep learning* es la metodología de clasificación más exitosa en la actualidad. Quizá se reparte las victorias con el método de Extreme Gradient Boosting. Para verificar estos datos, recomendamos visitar el siguiente enlace: https://www.kaggle.com/.

Las razones que podemos resaltar que benefician a los métodos de *deep learning* son los siguientes:

- Ofrecen rendimientos muy superiores a la mayoría de los métodos existentes.
- Simplicidad en abordar problemas extremadamente complejos (lo eran hace unos años).
- Automatiza la parte más crítica en los problemas de *machine learning*: *feature engineering* (proceso basado en el conocimiento del dominio de los datos para crear características que permitan a los algoritmos de aprendizaje incrementar su rendimiento. Proceso en general difícil y costoso).
- Permite adaptarse al problema con opciones de añadir capas adicionales para incrementar la capacidad de representación.
- Todas las etapas involucradas en *deep learning* están simplificadas y empaquetadas en la librería de keras (API de la librería de Google tensorflow).

Abordar problemas de *deep learning* lleva en ocasiones asociado el disponer de máquinas de altas prestaciones. De hecho, podemos disponer de altas prestaciones con una tarjeta gráfica de coste medio. Es imprescindible disponer de una GPU en situaciones donde la arquitectura de la red es considerable (podemos llegar a tener que estimar más de 14 millones de parámetros). Ejemplos actuales de GPU del mercado:

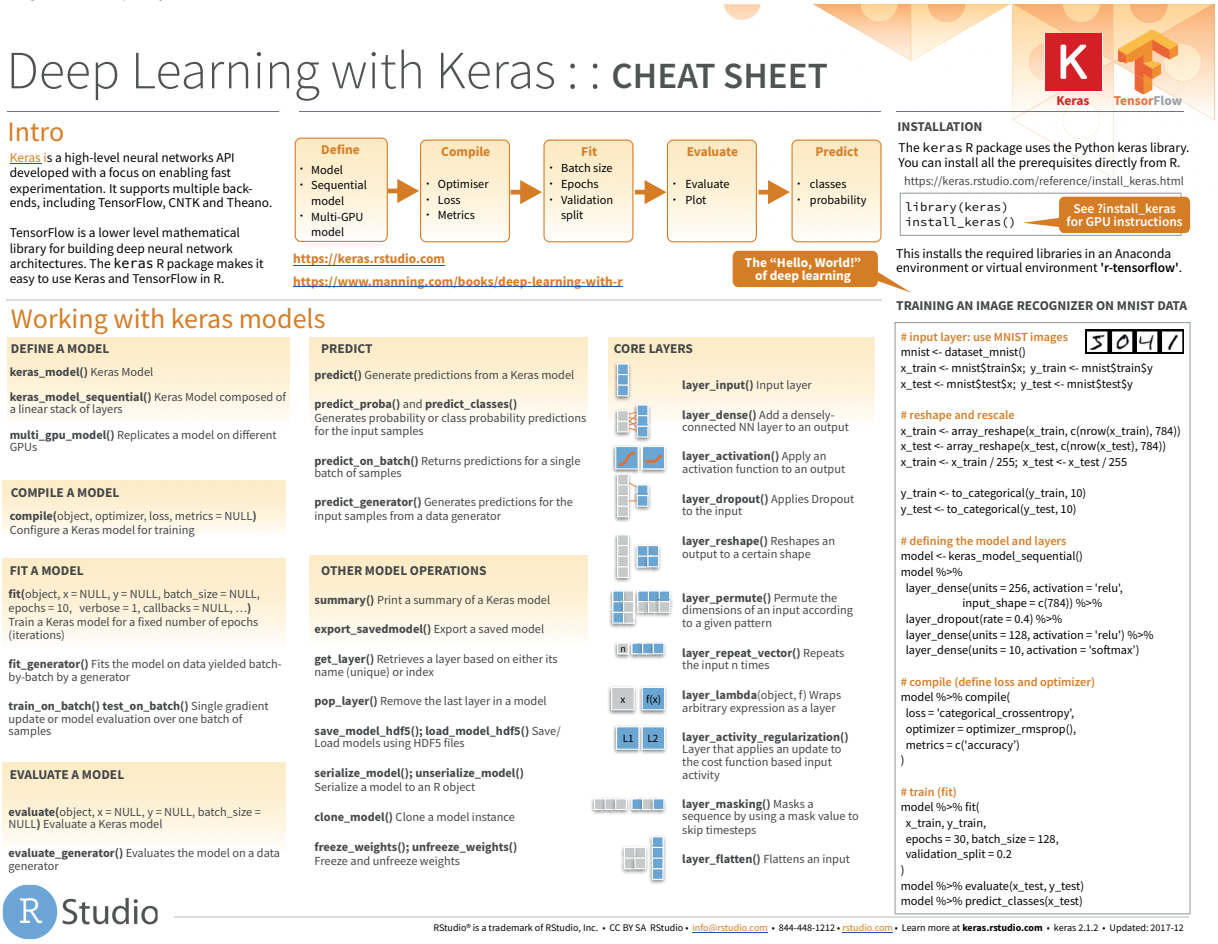
- NVIDIA GTX-1080: número de *cores* > 2000
- NVIDIA TITAN Xp: número de *cores* > 3000 (3840 NVIDIA CUDA cores)
- NVIDIA GEFORCE GTX TITAN Z: número de *cores* > 5000 (5760 NVIDIA CUDA Cores)

Es importante para sacar el máximo rendimiento a los problemas disponer de una GPU. Si un procesador normal puede tener de 4 a 16 núcleos de procesamiento en una GPU, el número se dispara. Será cuestión de nuestro presupuesto disponer de un modelo u otro.

2.5.1. Mejoras en los algoritmos

El hecho de disponer de una API como keras permite absoluta flexibilidad y alternativas para cubrir la mayoría de los escenarios con los que nos podamos encontrar.

Figura 17. Ejemplo de uso de la librería keras de RStudio



Fuente: https://www.rstudio.com/resources/cheatsheets/

Existían originalmente tres puntos críticos con las redes neuronales:

- la funciones de activación,
- los esquemas de inicialización de los pesos de la red y
- los algoritmos de optimización global.

Hasta que no se consiguieron operativas de funcionamiento correcto con un número considerable de capas, los métodos de *deep learning* no empezaron a ocupar el espacio que actualmente ocupan. Muchos avances han facilitado este salto considerable. Se pueden investigar tesis doctorales de hace diez años, en diversos campos, como el de procesado digital de la imagen, donde actualmente cualquier usuario en treinta minutos utilizando alguna arquitectura de *deep learning* y con la API de keras mejorar los resultados ofrecidos.

¿Hace gracia?

De hecho, en la actualidad, las mejores soluciones *deep learning* son capaces de superar a expertos en diferentes disciplinas. Google testeó la fiabilidad para diagnosticar melanomas y la contrastó con un equipo experto de dermatólogos. El resultado es que los algoritmos ofrecen mejores resultados. Experimentos parecidos se han repetido y el resultado es evidente. Estos algoritmos superan la capacidad de diagnosticar correctamente desde una imagen.

2.5.2. Tipologías de redes neuronales

Existen diferentes arquitecturas de redes neuronales. A continuación detallamos las más utilizadas:

- Convolutional Neural Networks (redes neuronales convolucionales)
- Long Short-Term Memory
- Recurrent neural networks
- Deep belief networks
- Deep stacking networks

Enlace de interés

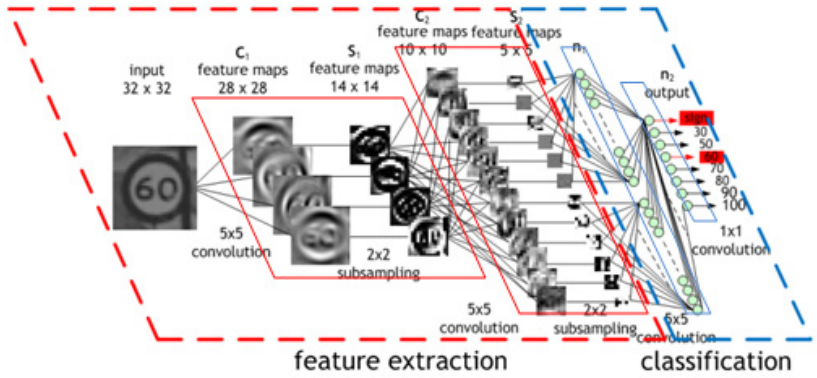
Es recomendable leer el artículo de IBM: https://ibm.co/2N9DS4j

2.5.3. Convolutional Neural Networks

Está inspirada biológicamente en la arquitectura neuronal del cortex visual. La arquitectura es particularmente útil en aplicaciones para procesar imágenes.

Una arquitectura típica es la mostrada en la figura 18.

Figura 18. Ejemplo de arquitectura de red neuronal convolutiva de Nvidia



Fuente: https://developer.nvidia.com/discover/convolutional-neural-network

Con la librería keras la definición de la arquitectura de la red hace uso de los métodos:

```
layer_conv_2d(...)
layer_max_pooling_2d(...)
```

Este procedimiento se puede encadenar para configurar una arquitectura más sofisticada. En general, las salidas de estas dos capas son un tensor 3D. Es decir, una estructura matricial (tensor) con tres dimensiones (altura, anchura y canal).

2.5.4. Long Short-Term Memory

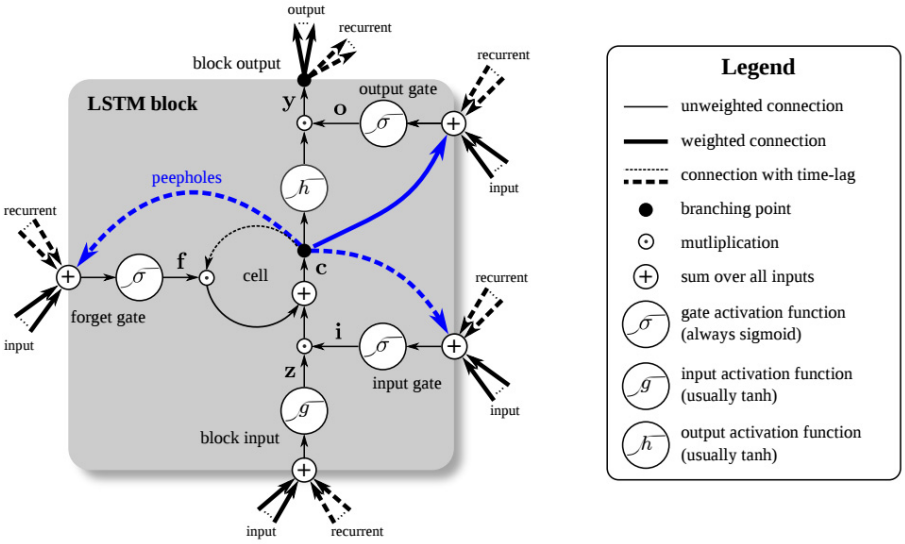
Este tipo de redes son indicadas para resolver problemas orientados a clasificar, procesar y hacer predicciones basadas en datos donde existe la variable tiempo (datos tipo series temporales).

Los problemas de predicción de series temporales incluyen una amplia gama de situaciones reales:

- cotizaciones en el mercado de valores (cualquier serie de ámbito económétrico),
- procedimiento de predicción de la siguiente palabra que escribir (en app tipo móvil),
- finalizaciones de composiciones musicales,
- identificación de las palabras clave de un documento (*keywords*) y
- procedimientos de traducción entre idiomas.

En figura 19 se aprecia una de las unidades básicas de la arquitectura de la red LSTM.

Figura 19. Ejemplo de arquitectura de red neuronal Long Short-Term Memory



Fuente: https://developer.nvidia.com/discover/lstm

Con la librería keras la definición de la arquitectura de la red hace uso de los métodos:

```
layer_simple_rnn(...)
```

2.5.5. Api Keras

Según la información facilitada por Rstudio, el *package* keras pone a disposición de los usarios de R una interfaz con keras, la API de *deep learning* centrada en el rápido prototipaje de modelos *deep learning*.

Keras ofrece las siguientes características:

- Permite ejecutar el mismo código en CPU o GPU de forma indistinta.
- Dispone de una API amigable que permite la rápida experimentación de los modelos en desarrollo.
- Da soporte a las redes de convolución (*convolutional networks*), con multitud de aplicaciones en el campo de la imagen por computadora, y da soporte a las redes recurrentes (*recurrent networks*), con una gran variedad de ejemplos en el procesamiento de secuencias o cualquier combinación de ambas.
- Da soporte a la creación de cualquier arquitectura arbitraria que el investigador quiera poner a prueba.
- Puede correr por encima de una gran variedad de *back-ends*, como tensorflow (desarrollado por Google Brain), cntk (desarrollado por Microsoft Research) o Theano (desarrollada por el Montreal Institute for Learning Algorithms, MILA, Universidad de Montreal).

Enlace de interés

Podemos descargarnos un resumen de los métodos asociados a keras en este enlace: https://bit.ly/2tnOxiS

2.5.6. Instalación

Para empezar a utilizar keras, es necesario instalar el *package* de keras:

```
install.packages("keras")
```

Por defecto, la interfaz de keras R utiliza tensorflow. Para instalar ambos módulos, es decir, la librería keras y el *back-end* de tensorflow, hacemos:

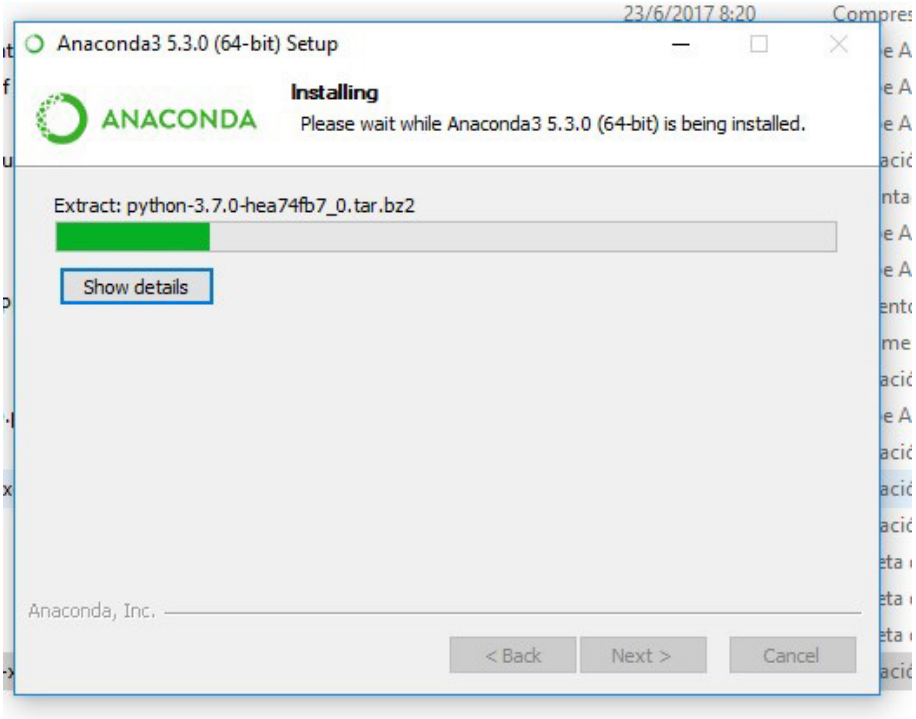
```
library(keras)
install_keras()
```

Debemos tener presente que si nuestro sistema no dispone de Python instalado, la instalación dará un error como el que se muestra a continuación:

```
Error: Keras installation failed (no conda binary found)

Install Anaconda for Python 3.x (https://www.anaconda.com/download/#windows)
before installing Keras.
```

Figura 20. Instalación de Anaconda para garantizar el correcto uso de keras



Una vez cargadas las dos librerías, tendremos la instalación base de CPU por defecto de keras y tensorflow. Si tenemos alguna GPU tipo Nvidia, como la Titan XP (NVIDIA CUDA Cores = 3840) o bien la GeForce GTX 1080 (NVIDIA CUDA Cores = 2560), se indicará en la carga de la librería.

```
install_keras(tensorflow = "gpu")
```

Si no disponemos de ninguna tarjeta GPU, es muy recomendable adquirir una por un precio razonable. Muchas arquitecturas *deep learning* necesitan optimizar más de 10^7 parámetros y este tipo de procesos requieren suficiente potencia para la computación distribuida.

2.5.7. MNIST Example

Vamos a resolver un problema que desde la perspectiva de un humano representa un ejercicio trivial, como es el reconocimiento de dígitos. La resolución para un algoritmo es una tarea mayúscula. De hecho, como ya hemos comentado en la introducción, el crecimiento en la popularidad de estos métodos de *deep learning* se debe al tipo de problemas que es capaz de resolver y en la eficacia de los resultados obtenidos.

Para poder realizar el ejercicio correctamente, vamos a necesitar el conjunto de datos y la librería keras (ver la figura 21 con una muestra del fichero que tratamos).

Figura 21. Diagrama con una muestra de los dígitos disponibles en el conjunto de datos MNIST

Fuente: Josef Steppan - Own work, CC BY-SA 4.0, https://bit.ly/2oblf3f

Carga de los datos

Para obtener los datos asociados al problema MNIST se puede acceder a:

http://yann.lecun.com/exdb/mnist/

Se puede observar que hay disponible un conjunto de datos correspondientes a dígitos escritos a mano. Están divididos de la siguiente forma:

- 60.000 muestras en el fichero de entrenamiento (*training*)
- 10.000 muestras en el fichero de test (*testing*)
- 60.000 etiquetas asociadas al fichero de entrenamiento (*training*)
- 10.000 etiquetas asociadas al fichero de test (*testing*)

Una opción válida y mucho mas rápida es utilizar los datos de MNIST ya precargados dentro de la librería keras.

```
mnist <- dataset_mnist()
```

La carga del fichero puede tardar unos segundos en función de la máquina disponible.

```
Downloading data from https://s3.amazonaws.com/img-datasets/mnist.npz
11493376/11490434 [==============================] - 34s 3us/step
```

Podemos apreciar que la estructura del objeto MNIST corresponde a dos listas, una de entrenamiento y otra de test. Cada una de ellas con un tensor de imágenes y un vector de etiquetas. Recordad que un tensor no es más que una estructura dimensional; un vector sería un tensor de 1 dimensión, una matriz sería un tensor de dimensión 2 y así sucesivamente. Al hablar de tensor nos referimos a este tipo de estructura multidimensional.

```
trainImg <- mnist$train$x # En el objeto trainImg imágenes entrenamiento
trainLab <- mnist$train$y # Etiquetas
testImg <- mnist$test$x # En el objeto tesImg imágenes test
testLab <- mnist$test$y # etiquetas
```

Podemos preguntarnos por la distribución de dígitos representados en cada conjunto de datos (*train* y *test*).

```
require(ggplot2)
require(gridExtra)
resTbl <- table(trainLab)
resTblDf <- as.data.frame(resTbl)
gp1 <- ggplot(resTblDf,aes(trainLab,Freq))+geom_bar(stat = "identity")+ theme_bw()
resTbl2 <- table(testLab)
resTblDf2 <- as.data.frame(resTbl2)
gp2 <- ggplot(resTblDf2,aes(testLab,Freq))+geom_bar(stat = "identity")+theme_bw()
grid.arrange(gp1, gp2,ncol=2)
```

Se observa una distribución de los dígitos parecida para cada uno de los dígitos presentes en cada uno de los ficheros.

```
digit <- c()
for(i in 1:10) {
digit[[i]] <- trainImg[i,,]
}
# First 5 digits
#par(mfrow=c(2,5))
for(i in 1:10) {
plot(as.raster(255-digit[[i]],max=255))
}
```

Definición de la arquitectura

Para definir la arquitectura utilizamos los métodos descritos en keras:

```
deepLearningNet <- keras_model_sequential() %>%
  layer_dense(units=512, activation = "relu", input_shape = c(28*28)) %>%
  layer_dense(units=10, activation = "softmax")
```

Especificación de la preparación de la arquitectura de red

```
deepLearningNet %>% compile(
  optimizer = "rmsprop",
  loss = "sparse_categorical_crossentropy",
  metrics = c("accuracy")
)
```

Optimización de la arquitectura de red

Se necesita verificar que los valores introducidos en la red están correctamente escalados y que están dentro del rango 0 y 1. Para hacer esta operación, keras facilita esta transformación.

```
trainImg <- array_reshape(trainImg, c(60000, 28*28))
trainImg <- trainImg /255
testImg <- array_reshape(testImg, c(10000, 28*28))
testImg <- testImg /255

deepLearningNet %>% fit(trainImg, trainLab, epochs = 5, batch_size = 128)
```

Este es el mensaje que se puede obtener si los datos no tienen correctamente las dimensiones adecuadas.

```
 Error in py_call_impl(callable, dots$args, dots$keywords) :
  ValueError: Error when checking input:
   expected dense_1_input to have 2 dimensions,
   but got array with shape (60000, 28, 28)
```

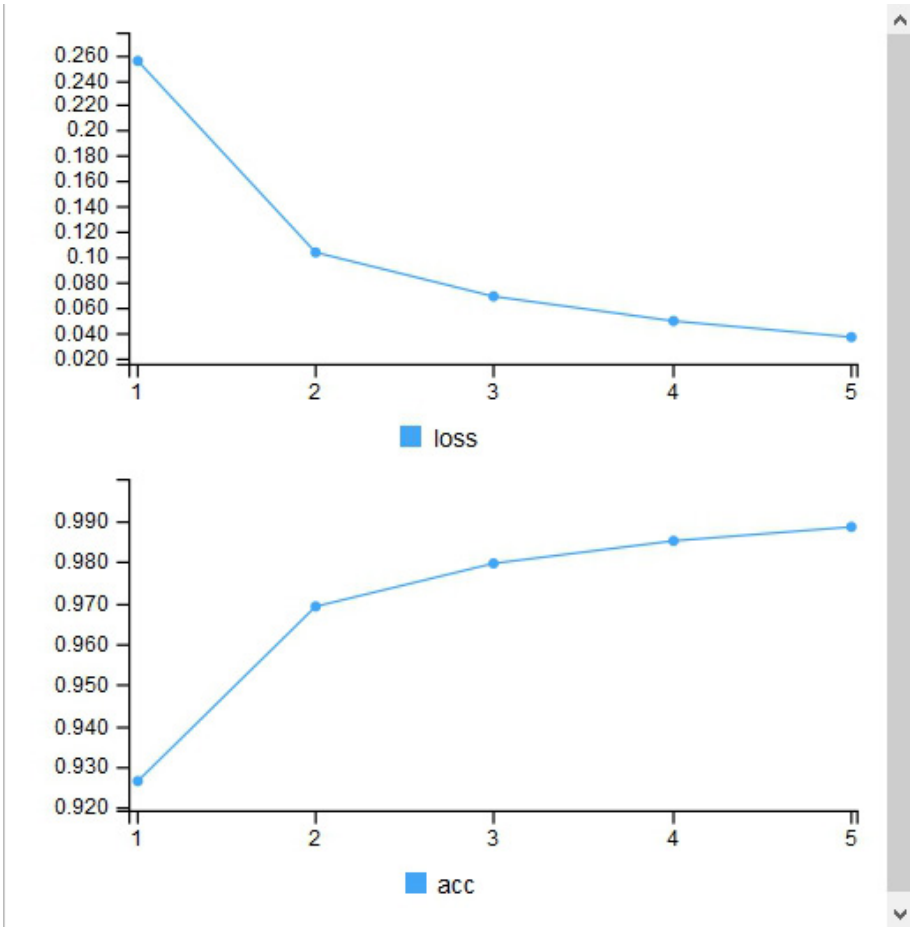
Si todo funciona correctamente, la salida puede ofrecer un resultado como el que a continuación presentamos:

```
2018-10-04 18:36:03.441297: I T:\src\github\tensorflow\tensorflow
\core\platform\cpu_feature_guard.cc:141]
 Your CPU supports instructions that this TensorFlow
  binary was not compiled to use: AVX2
Epoch 1/5
60000/60000 [==============] - 3s 47us/step - loss: 0.2562 - acc: 0.9263
Epoch 2/5
60000/60000 [==============] - 3s 46us/step - loss: 0.1027 - acc: 0.9699
Epoch 3/5
60000/60000 [==============] - 3s 44us/step - loss: 0.0676 - acc: 0.9791
Epoch 4/5
60000/60000 [==============] - 3s 43us/step - loss: 0.0497 - acc: 0.9851
Epoch 5/5
60000/60000 [==============] - 3s 43us/step - loss: 0.0376 - acc: 0.9888
```

Cada iteración muestra el tiempo total de cada proceso, el valor de la función de pérdida y la medida de la precisión.

La evolución se puede visualizar con un gráfico, que puede ayudar a decidir si conviene modificar alguno de los parámetros utilizados para el entrenamiento.

Figura 22. Evolución de la función de pérdida junto con la evolución de la precisión



Validación sobre el conjunto de datos test

Una vez realizado el entrenamiento, podemos evaluar sobre el conjunto de datos de test.

```
rendimiento <- deepLearningNet %>% evaluate(testImg, testLab)
```

Podemos observar que el resultado ofrece una tasa de aciertos del **97.78 %**.

Podemos visualizar también los resultados sobre los primeros 20 dígitos:

```
predClass <- deepLearningNet %>%  predict_classes(testImg[1:20,])
realClass <- testLab[1:20]
# y apreciar los aciertos
predClass
realClass
realClass == predClass
```

Para poder visualizar la arquitectura que se ha definido, podemos utilizar el método *summary* con el nombre de la red previamente definida.

```
summary(deepLearningNet)
```

En la siguiente salida de datos relativa a la arquitectura, vemos el número de capas de nuestra red, así como el número de parámetros que están asociados a esta capa.

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 512)               401920
_________________________________________________________________
dense_2 (Dense)              (None, 10)                5130
=================================================================
Total params: 407,050
Trainable params: 407,050
Non-trainable params: 0
_________________________________________________________________
```

Para disponer de los resultados en formato matriz de confusión y facilitar diversas medidas de utilidad, podemos hacer uso de la librería carat y proceder de la siguiente manera:

```
# Confusion Matrix
  dimt <- length(testLab)
  predClass <- deepLearningNet %>%  predict_classes(testImg[1:dimt,])
  realClass <- testLab

  # use caret to use confusion matrix
  resConfMat1 <- confusionMatrix(as.factor(predClass),as.factor(realClass))
```

Matriz de confusión asociada al modelo propuesto:

	0	1	2	3	4	5	6	7	8	9
0	954	0	11	2	2	9	14	2	10	13
1	0	1108	5	2	5	3	3	10	8	6
2	3	2	922	22	5	6	5	24	13	3
3	1	2	15	898	0	33	1	5	29	12
4	0	0	14	0	901	9	9	6	12	31
5	9	3	3	40	3	775	16	0	43	8
6	10	4	15	2	12	16	906	0	13	0
7	1	1	11	19	2	11	2	943	15	31
8	2	15	32	16	3	23	2	3	814	3
9	0	0	4	9	49	7	0	35	17	902

De esta matriz se derivan los siguientes cálculos:

	x
Accuracy	0.9123000
Kappa	0.9025170
AccuracyLower	0.9065853
AccuracyUpper	0.9177733
AccuracyNull	0.1135000
AccuracyPValue	0.0000000
McnemarPValue	NaN

Y dado que tenemos información asociada a diez posibles clases, podemos observar el comportamiento de cada clase. Este conjunto de valores de rendimiento pueden ayudar a identificar una clase que presente más problemas o que presente un comportamiento excepcional (i.e. siempre clasifica correctamente).

	Senst	Specf	+Pred	-Pred	Prec.	Recall	F1	Prev	DetR	DetPr	BalAc
Cls: 0	0.973	0.993	0.938	0.997	0.938	0.973	0.955	0.098	0.095	0.102	0.983
Cls: 1	0.976	0.995	0.963	0.997	0.963	0.976	0.970	0.114	0.111	0.115	0.986
Cls: 2	0.893	0.991	0.917	0.988	0.917	0.893	0.905	0.103	0.092	0.100	0.942
Cls: 3	0.889	0.989	0.902	0.988	0.902	0.889	0.895	0.101	0.090	0.100	0.939
Cls: 4	0.918	0.991	0.918	0.991	0.918	0.918	0.918	0.098	0.090	0.098	0.954
Cls: 5	0.869	0.986	0.861	0.987	0.861	0.869	0.865	0.089	0.078	0.090	0.928
Cls: 6	0.946	0.992	0.926	0.994	0.926	0.946	0.936	0.096	0.091	0.098	0.969
Cls: 7	0.917	0.990	0.910	0.991	0.910	0.917	0.914	0.103	0.094	0.104	0.953
Cls: 8	0.836	0.989	0.892	0.982	0.892	0.836	0.863	0.097	0.081	0.091	0.912
Cls: 9	0.894	0.987	0.882	0.988	0.882	0.894	0.888	0.101	0.090	0.102	0.940

Observación: la codificación corresponde a:

- Senst ⇒ Sensitivity
- Specf ⇒ Specificity
- +Pred ⇒ Pos Pred Value
- -Pred ⇒ Neg Pred Value
- Prec. ⇒ Precision
- Recall ⇒ Recall
- F1 ⇒ F1
- Prev ⇒ Prevalence
- DetR ⇒ Detection Rate
- DetPr ⇒ Detection Prevalence
- BalAc ⇒ Balanced Accuracy

2.5.8. Ejemplo clasificación de dígitos - 2 (MNIST)

Si repetimos el proceso de una manera más compacta:

```
# Define el Modelo: Network Arquitechture
deepLearningNet1 <- keras_model_sequential() %>%
  layer_dense(units=512, activation = "relu", input_shape = c(28*28)) %>%
  layer_dense(units=10, activation = "softmax")

# Compilation Step
deepLearningNet1 %>% compile(
  optimizer = "rmsprop",
  loss = "categorical_crossentropy",
  metrics = c("accuracy"))


# Preparing Data
trainImg <- array_reshape(trainImg, c(60000, 28*28))
trainImg <- trainImg /255
```

```
testImg <- array_reshape(testImg, c(10000, 28*28))
testImg <- testImg /255

############################
trainLab <- to_categorical(trainLab)
testLab <- to_categorical(testLab)
############################

##################################
# Train Network
deepLearningNet1 %>% fit(trainImg, trainLab, epochs = 5, batch_size = 128)
#
##################################


##################################
# Test
perform <- deepLearningNet1 %>% evaluate(testImg, testLab)
perform

# Si queremos hacer el cálculo de la "Confusion Matrix"
dimt <- length(testLab) # dim(testImg)
predClass <- deepLearningNet1 %>%  predict_classes(testImg[1:10000,])
realClass <- mnist$test$y # etiquetas con la clase real

# uso de la librería caret para calcular la matriz de confusión
resConfMat11 <- confusionMatrix(as.factor(predClass),as.factor(realClass))
```

El proceso de entrenamiento da unos valores ligeramente mejores:

```
Epoch 1/5
60000/60000 [================] - 3s 50us/step - loss: 0.2550 - acc: 0.9265
Epoch 2/5
60000/60000 [================] - 3s 45us/step - loss: 0.1039 - acc: 0.9691
Epoch 3/5
60000/60000 [================] - 3s 44us/step - loss: 0.0692 - acc: 0.9796
Epoch 4/5
60000/60000 [================] - 3s 44us/step - loss: 0.0498 - acc: 0.9851
Epoch 5/5
60000/60000 [================] - 3s 44us/step - loss: 0.0371 - acc: 0.9885
```

Y si realizamos el cálculo sobre el conjunto de datos test:

```
perform
$`loss`
```

```
[1] 0.06512839

$acc
[1] 0.9806
```

¿Qué ha cambiado del modelo?

Aquí mostramos la matriz de confusión:

	0	1	2	3	4	5	6	7	8	9
0	970	0	3	0	1	2	5	0	4	1
1	0	1129	3	0	1	0	3	9	1	9
2	0	1	1008	1	1	0	1	5	3	0
3	1	1	3	1000	1	9	1	2	4	6
4	0	0	2	0	969	2	7	1	4	7
5	1	1	0	2	0	869	2	0	4	2
6	2	2	2	0	4	4	936	0	2	0
7	1	0	6	3	1	0	0	1005	4	6
8	3	1	5	2	1	3	3	2	945	3
9	2	0	0	2	3	3	0	4	3	975

Medidas de rendimiento adicionales:

	x
Accuracy	0.9806000
Kappa	0.9784352
AccuracyLower	0.9777025
AccuracyUpper	0.9832126
AccuracyNull	0.1135000
AccuracyPValue	0.0000000
McnemarPValue	NaN

Y, por último, resultados asociados a cada clase de nuestro problema de clasificación.

	Senst	Specf	+Pred	-Pred	Prec.	Recall	F1	Prev	DetR	DetPr	BalAc
Cls: 0	0.973	0.993	0.938	0.997	0.938	0.973	0.955	0.098	0.095	0.102	0.983
Cls: 1	0.976	0.995	0.963	0.997	0.963	0.976	0.970	0.114	0.111	0.115	0.986
Cls: 2	0.893	0.991	0.917	0.988	0.917	0.893	0.905	0.103	0.092	0.100	0.942
Cls: 3	0.889	0.989	0.902	0.988	0.902	0.889	0.895	0.101	0.090	0.100	0.939
Cls: 4	0.918	0.991	0.918	0.991	0.918	0.918	0.918	0.098	0.090	0.098	0.954
Cls: 5	0.869	0.986	0.861	0.987	0.861	0.869	0.865	0.089	0.078	0.090	0.928
Cls: 6	0.946	0.992	0.926	0.994	0.926	0.946	0.936	0.096	0.091	0.098	0.969
Cls: 7	0.917	0.990	0.910	0.991	0.910	0.917	0.914	0.103	0.094	0.104	0.953
Cls: 8	0.836	0.989	0.892	0.982	0.892	0.836	0.863	0.097	0.081	0.091	0.912
Cls: 9	0.894	0.987	0.882	0.988	0.882	0.894	0.888	0.101	0.090	0.102	0.940

Métodos de regularización

Uno de los principales problemas que debemos resolver al aplicar métodos de *deep learning* es que el rendimiento del sistema sobre los datos de entrenamiento sea igual de bueno que en los datos de test. Este tipo de estrategias se denominan de regularización. Las hemos visto en el caso de los métodos de regresión y a continuación las describimos para el caso de *deep learning*.

La mayoría de los métodos de regularización se basan en añadir alguna restricción adicional al modelo. Añadir un término adicional a la función objetivo suele ser una de estas aproximaciones.

Si nuestra función objetivo J sobre el conjunto de hiperparámetros θ con los datos X y salidas y:

$$J(\theta;\ X,y) = J(\theta;\ X,y) + \alpha\Sigma(\theta)$$

donde el parámetro $\alpha \in [0, \text{inf})$ corresponde al peso que tendrá la contribución de la penalización Σ.

Regularización L^2 (también conocida como *ridge regression*)

En este caso tenemos:

$$J(\theta;\ X,y) = J(\theta;\ X,y) + \frac{\alpha}{2} w^T w$$

El efecto que esta regularización tiene sobre nuestro modelo es que modificamos la tasa de aprendizaje.

Regularización L^1

En este caso la componente que se añade es:

$$\Omega(\sigma) = ||w||_1$$

y por tanto respecto a la función objetivo:

$$J(\theta;\ X,y) = J(\theta;\ X,y) + \alpha||w||_1$$

El efecto de esta regularización es radicalmente diferente a la propuesta anteriormente. Para apreciar esta diferencia, deberíamos diferenciar las funciones para observar que en este caso se hace una modificación que es constante, con-

siguiendo un efecto de *sparsity*. Esto es similar al método LASSO (introducido por Tibshirani 1995) en los modelos de regresión. Este tipo de penalización es utilizada para facilitar las etapas de selección de atributos.

Incremento de datos (*dataset augmentation*)

Para incrementar la capacidad de generalización del modelo, es vital disponer de más datos. Si no disponemos de suficientes datos, existe la posibilidad de aumentar los datos y aplicar una estrategia de generación de nuevos datos (que no han sido observados).

Dropout introduce un método eficiente y muy potente de regularización. Esta técnica permite reducir el sobreajuste de las redes neuronales evitando las adaptaciones complejas con los datos de entrenamiento.

Keras permite esta operación con la capa:

```
layer_dropout(rate = 0.5)
```

Ejemplo de uso de esta capa:

```
model = keras_model_sequential()
model %>%
  layer_dense(input_shape = 4, units = 10, activation = "relu") %>%
  layer_dropout(0.2) %>%
  layer_dense(units = 10, activation = "relu") %>%
  layer_dropout(0.2) %>%
  layer_dense(units = 3, activation = "softmax")
model %>%  compile(loss = "categorical_crossentropy",
    optimizer = "rmsprop",metrics = "accuracy"  )
fit = model %>% fit(as.matrix(xTrain1),yTrain1,batch_size = 5,epochs = 50)
```

3. Caso práctico

3.1. Descripción

La idea de este apartado es, a partir de un juego de datos diferente, aplicar algunas de las distintas herramientas presentadas.

Se van a analizar estos dos grupos de datos: cars y Arrests.

3.1.1. Regresión sobre los datos cars

En la librería caret hay un conjunto de datos llamado cars, con 804 observaciones y un total de 18 variables. Estas variables recogen información de la reventa de vehículos de la empresa GM (General Motors) sobre modelos de 2005. La variable de interés es el precio, Price, del vehículo. ¿Qué modelo podemos construir para que sea capaz de predecir el precio de una forma correcta?

```
# Cargar la librería
library(caret)

# Cargar los datos
data(cars)

# visualizar la estructura de estos
str(cars)
'data.frame': 804 obs. of  18 variables:
 $ Price      : num  22661 21725 29143 30732 33359 ...
 $ Mileage    : int  20105 13457 31655 22479 17590 23635 17381 27558 25049 17319 ...
 $ Cylinder   : int  6 6 4 4 4 4 4 4 4 4 ...
 $ Doors      : int  4 2 2 2 2 2 2 2 2 4 ...
 $ Cruise     : int  1 1 1 1 1 1 1 1 1 1 ...
 $ Sound      : int  0 1 1 0 1 0 1 0 0 0 ...
 $ Leather    : int  0 0 1 0 1 0 1 1 0 1 ...
 $ Buick      : int  1 0 0 0 0 0 0 0 0 0 ...
 $ Cadillac   : int  0 0 0 0 0 0 0 0 0 0 ...
 $ Chevy      : int  0 1 0 0 0 0 0 0 0 0 ...
 $ Pontiac    : int  0 0 0 0 0 0 0 0 0 0 ...
 $ Saab       : int  0 0 1 1 1 1 1 1 1 1 ...
 $ Saturn     : int  0 0 0 0 0 0 0 0 0 0 ...
```

```
 $ convertible: int  0 0 1 1 1 1 1 1 1 0 ...
 $ coupe      : int  0 1 0 0 0 0 0 0 0 0 ...
 $ hatchback  : int  0 0 0 0 0 0 0 0 0 0 ...
 $ sedan      : int  1 0 0 0 0 0 0 0 0 1 ...
 $ wagon      : int  0 0 0 0 0 0 0 0 0 0 ...
```

Tenemos que explicar la variable Price (precio) y debemos buscar cuáles son los mejores regresores, así como el tipo de modelo que aplicar.

Estas son las cuestiones que debemos resolver:

1) Si queremos explicar el precio con una única variable, ¿cuál será?

2) ¿Qué interpretación se realiza al utilizar una única variable explicativa?

3) Extender el modelo para utilizar las variables que se consideren (del conjunto de las variables disponibles) para mejorar la capacidad de predicción.

4) ¿Qué medidas de evaluación son recomendables?

5) Contrastar método de regresión con uno de *deep learning*.

3.1.2. Classificación sobre los datos Arrests

En la librería carData hay un conjunto de datos llamado Arrests, con 5.226 observaciones y un total de 8 variables. Estas variables recogen información de arrestos producidos por la policía de Toronto por posesión de pequeñas cantidades de marihuana. Nuestro objetivo es explicar la variable released.

```
# Cargar la librería
library(carData)

# Cargar los datos
data(Arrests)

# visualizar la estructura de estos
'data.frame': 5226 obs. of  8 variables:
 $ released: Factor w/ 2 levels "No","Yes": 2 1 2 1 2 2 2 2 2 2 ...
 $ colour  : Factor w/ 2 levels "Black","White": 2 1 2 1 1 1 2 2 1 2 ...
 $ year    : int  2002 1999 2000 2000 1999 1998 1999 1998 2000 2001 ...
 $ age     : int  21 17 24 46 27 16 40 34 23 30 ...
 $ sex     : Factor w/ 2 levels "Female","Male": 2 2 2 2 1 1 2 1 2 2 ...
 $ employed: Factor w/ 2 levels "No","Yes": 2 2 2 2 2 2 1 2 2 2 ...
 $ citizen : Factor w/ 2 levels "No","Yes": 2 2 2 2 2 2 2 2 2 2 ...
 $ checks  : int  3 3 3 1 1 0 0 1 4 3 ...
```

¿Qué variables son más determinantes? Preguntas por resolver:

1) ¿Cuál es la variable (única) que mejor explica la variable released?

2) ¿Qué tipo de modelo de regresión resuelve esta pregunta?

3) ¿Todas las observaciones presentan un comportamiento similar y son necesarias para el análisis?

4) Muestra la matriz de confusión para un conjunto de modelos probados. ¿Qué modelo es mejor?

5) ¿Cómo podéis interpretar los resultados si el modelo usado es de *deep learning*?

Resumen

En general, hemos de ser capaces de identificar el paradigma de aprendizaje apropiado. ¿Qué tipo de problema pretendemos resolver? El análisis del problema permitirá decidir la aproximación correcta.

La aplicación de una metodología ML implica ser capaz de analizar algunas de las cuestiones que aquí se exponen a título de resumen.

- Si es una clasificación o si es una regresión.
- Características de la variable *target*. En caso de clasificación, verificar proporciones para identificar posibles sesgos. En el caso de regresión, contrastar la distribución asociada a la variable *target*.
- ¿Qué objetivo persigue el modelo?, ¿ofrecer probabilidades, rangos, etiquetas, etc.?
- ¿Qué debemos hacer con el modelo?, ¿se debe utilizar para interpretar o únicamente interesa que sea útil?
- ¿Qué garantías ofrece el modelo seleccionado?
- Considerar siempre diversos modelos.
- Considerar verificar bajo diferentes conjuntos de *training* y *test*.
- Definir los criterios sobre los que se seleccionará el mejor modelo.
- Sobre el modelo seleccionado ser conocedor de los diferentes parámetros que permiten ajustarlo.

Hay que ser precavidos en los resultados ofrecidos por la solución seleccionada. Nunca obtendremos un *warning* de advertencia y es nuestra responsabilidad garantizar que en entornos de exploración se garantiza que funciona automáticamente bajo las condiciones y suposiciones preestablecidas.

Hay que analizar la capacidad del modelo para no estar en *underfitting* o en *overfitting*. Se debe empezar por modelos simples y añadir complejidad a medida que se asegura que tiene sentido incrementar la complejidad del modelo.