Módulo 4: Introducción al Machine Learning con R

Parte III: Aprendizaje con R

Autor: Raquel Dormido Canto

Actualizado Enero 2023

Contenido

4. APRENDIZAJE

4.1. Introducción

4.1.1. ¿Qué es el Machine Learning?

Machine Learning (en castellano, Aprendizaje Automático) es una disciplina científica del ámbito de la Inteligencia Artificial que crea sistemas que aprenden automáticamente. El Machine Learning trata pues de desarrollar técnicas para que las máquinas puedan aprender y tomar decisiones por sí mismas. En este contexto, cuando decimos *aprendizaje* queremos decir sistemas que aprenden de datos. La máquina que realmente aprende es un algoritmo que, a través de conjuntos de datos, deducen información sobre las propiedades de esos datos y la información extraída le permite hacer predicciones sobre comportamientos futuros. Esto es posible porque extrae patrones de los datos y esos patrones permiten a la máquina generalizar. El Machine Learning engloba distintas técnicas (clasificación, clustering, regresión,…) y múltiples implementaciones. En este módulo estudiaremos algunas de ellas. El término *automático*, también en este contexto, implica que estos sistemas se mejoran de forma autónoma con el tiempo, sin intervención humana. Existen distintas definiciones sobre el Machine Learning o Aprendizaje Automático.

Dos de las definiciones más populares que dan una idea clara de lo que pretende el Machine Learning son:

- El Aprendizaje Automático es el proceso que le da a las computadoras la habilidad de aprender sin ser explícitamente programadas. – A.L Samuel (1959)

- Se dice que un programa de ordenador es capaz de aprender de la **experiencia** (E) con respecto a una **tarea** (T) y alguna medida de **rendimiento** (P), si su desempeño en T, medido por P, mejora con la experiencia E. – T.M Mitchell (1998)

La segunda definición nos da la idea general del funcionamiento de la mayoría de los algoritmos de aprendizaje: Crear un programa que pueda aprender a realizar una tarea, y mejorar su rendimiento en la misma mediante entrenamiento (experiencia).

Supongamos, por ejemplo, que nuestro programa tiene como tarea (T) el predecir el tráfico en un cruce de calles. Podríamos realizar esa predicción utilizando algún algoritmo de Machine Learning utilizando datos de tráfico pasados (experiencia E). Si el programa tiene un buen aprendizaje será capaz de predecir tráfico en el futuro (este comportamiento podrá ser medido por P).

Pensemos en otro ejemplo. Una empresa de seguros que quiere saber qué clientes están próximos a darse de baja de sus servicios (tarea T), con el objetivo de poder hacer acciones comerciales que

eviten que se vayan a la competencia. ¿Cómo puede hacerlo? La empresa tiene muchos datos de sus clientes: planes contratados, antigüedad, llamadas mensuales al servicio de atención al cliente, últimos cambios de planes contratados… pero seguramente los utiliza solo para facturar y para hacer estadísticas. ¿Qué más puede hacer con esos datos?

Podemos usar todos esos datos para predecir, utilizando algoritmos de Machine Learning, cuándo un cliente se va a dar de baja y gestionar la mejor acción que lo evite. El Machine Learning permitiría a la empresa pasar de ser reactivos a ser proactivos. Los datos históricos del conjunto de los clientes (experiencia, E), debidamente organizados y tratados se pueden explotar para predecir comportamientos futuros, favorecer aquellos que mejoran los objetivos de negocio y evitar aquellos que son perjudiciales. Los algoritmos de Machine Learning permiten detectar patrones de comportamiento a partir de las variables que les proporcionamos y podrán descubrir, en el caso del ejemplo, cuáles son las que le han llevado a darse de baja como cliente. Un enlace en que podemos encontrar explicado de forma sencilla qué es el Machine Learning y cómo funciona es el siguiente: http://blogthinkbig.com/machine-learning-que-como-cuando-donde/

4.1.2. Ámbitos de Aplicación del Machine Learning

El impacto del Machine Learning es enorme, siendo una disciplina en la que se puede enmarcar una larguísima lista de aplicaciones. Realmente podríamos pensar en tantas aplicaciones como imaginemos, pudiéndose adaptar a tantas situaciones como datos con los que contemos. En esta sección nombraremos algunos de sus ámbitos de aplicación.

Muchas actividades actualmente ya se están aprovechando del Machine Learning. Hay sectores que ya llevan tiempo sacando partido a estas tecnologías, como por ejemplo el sector de las compras online (pensemos por ejemplo en cómo se deciden instantáneamente los productos recomendados para cada cliente al final de un proceso de compra), o el sector del online advertising (dónde poner un anuncio para que tenga más visibilidad en función del usuario que visita la web).

Hay otros sectores en los que el Machine Learning resulta clave para la toma de decisiones. En el ámbito quirúrgico, por ejemplo, resulta de gran utilidad a la hora de decidir si es conveniente llevar a cabo una operación partiendo de la tasa de éxito y de las características personales de pacientes anteriores. En el campo de los negocios sus aplicaciones son de lo más diversas permitiendo, por ejemplo, desde establecer en qué fechas es mejor subir o bajar los precios de acuerdo con la demanda, hasta estimar si el ritmo de las ventas está siendo óptimo en un momento dado.

El campo de aplicación práctica del Machine Learning depende de la imaginación y de los datos que estén disponibles en cada caso. Estos son algunos ejemplos en los que se aplica el Machine Learning:

- Predecir de fallos en equipos tecnológicos.
- Reconocimiento facial, de voz o de objetos.
- Buscadores. Para mejorar los resultados y sugerencias de búsqueda.
- Genética. Por ejemplo, en la clasificación de secuencias de ADN.
- Predicción y pronósticos. De clima, tráfico o para evitar fallos tecnológicos en equipos.
- Seleccionar clientes potenciales basándose en comportamientos en las redes sociales, interacciones en la web…
- Saber cuál es el mejor momento para publicar tuits o actualizaciones de Facebook.
- Hacer prediagnósticos médicos basados en síntomas del paciente.
- Cambiar el comportamiento de una app móvil para adaptarse a las costumbres y necesidades de cada usuario.
- Detectar intrusiones en una red de comunicaciones de datos.
- Decidir cuál es la mejor hora para llamar a un cliente.
- Análisis de imágenes de alta calidad.
- Análisis de datos económicos. Para operar en el mercado de valores o evitar el fraude en transacciones.
- Análisis de comportamiento de consumo y productividad. Para la identificación de clientes potenciales, prever qué empleados pueden ser más rentables, adaptar servicios a las necesidades del usuario…

El Machine Learning, por lo tanto, resulta efectivo en problemas de naturaleza compleja en los que la aplicación de algoritmos ayuda a la obtención de soluciones, con el consecuente ahorro de tiempo que sus métodos implican.

La tecnología está ahí y los datos también y muchas empresas están muy interesadas en su uso. De hecho, es interesante que accedáis, si no lo habéis hecho ya a la plataforma Kaggle. Esta plataforma aloja competiciones de análisis de datos y modelado predictivo donde compañías e investigadores aportan sus datos mientras que estadistas e ingenieros de datos de todo el mundo compiten por crear los mejores modelos de predicción o clasificación.

La tecnología de Machine Learning está ahí. Los datos también. Ahora sólo queda el empezar a aprenderla y manejarla.

4.1.3. Objetivos de este Módulo

En esta parte del módulo 6 del máster pretendemos mostrar una introducción a distintas técnicas de Machine Learning utilizando R. Mostraremos, en primer lugar, unas definiciones de la terminología habitual y conceptos básicos del Machine Learning. Trataremos también sus dos categorías más

habituales: aprendizaje supervisado y no supervisado. A continuación, definiremos distintas técnicas que se pueden aplicar a los datos utilizando Machine Learning: regresión, clasificación y clustering. Por último, trataremos cada una de estas técnicas usando R. En cada una de las técnicas mostramos ejemplos de cómo funcionan algunos de los algoritmos más representativos, ya sea con código, o haciendo uso de alguna librería de R. En la medida de lo posible no nos centraremos en el formalismo teórico del algoritmo, tratando de no hacer uso de ecuaciones. Usaremos las herramientas de análisis numérico y gráfico que nos proporciona R. La meta que nos proponemos es explicar algunos algoritmos de Machine Learning basándonos en ejemplos para poder saber cómo aplicarlos a otros datos.

4.2. Terminología del Machine Learning

En esta sección recopilamos algunos conceptos y terminología básica de Machine Learning.

Conjunto de datos (dataset): Es la materia prima con la que trabajan los sistemas de Aprendizaje Automático. Es el histórico de datos que se usa para entrenar al sistema que detecta los patrones. El conjunto de datos se compone de *instancias* (también llamadas objetos o registros), y las instancias de *características* o propiedades.

Instancia, objeto o registro: Es cada uno de los datos de los que se dispone para hacer un análisis, esto es, uno de los elementos a procesar (por ejemplo, a clasificar). Puede ser un documento, una foto, un vídeo, una fila en una base de datos…. Por ejemplo, si se quiere predecir el comportamiento de los clientes de una compañía de seguros, cada instancia correspondería a un asegurado. Cada instancia, a su vez, está compuesta de *características* que la describen, como la antigüedad del cliente en la compañía, el gasto diario en llamadas, etc. En una hoja de cálculo, las instancias serían las filas; las características, las columnas.

Característica o atributo (feature): Son las características o rasgos distintos que se pueden utilizar para describir cada objeto (cada instancia) del conjunto de datos. En el caso de una cartera de clientes, estaríamos hablando del número de compras de cada cliente, antigüedad, si es seguidor en redes sociales, si se ha dado de alta en la newsletter,…En una hoja de cálculo, serían las columnas.

Vector de características: Vector *n*-dimensional con características numéricas que representan un elemento. En la Figura 4.2-1 se muestran las características correspondientes a tres objetos distintos.

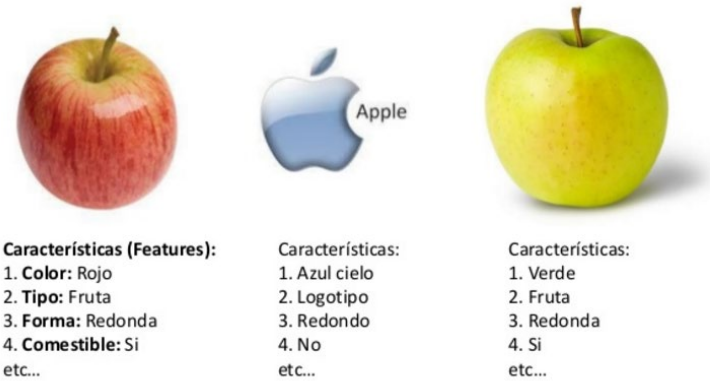


Figura 4.2-1: Vectores de características.

Extracción de características: Preparación del vector de características. Transforma los datos de un espacio con alta dimensionalidad a un espacio con menos dimensiones.

Objetivo (objective): Es el atributo o característica que queremos predecir, el objetivo de la predicción o clasificación. Por ejemplo, la probabilidad de reingreso de un paciente tras una intervención quirúrgica.

Preparación de datos: Se trata del proceso previo a la creación del modelo de predicción en el que se hace un análisis, limpieza y estructuración de los campos de los datos. Es una fase importante y costosa del proceso de predicción. El objetivo es eliminar los campos que no sirven para hacer la predicción y organizarlos adecuadamente para que el modelo no reciba información que no le es útil y que podría provocar predicciones de poca calidad o confianza.

Aprendizaje (learning): Es el proceso en el que se detectan los patrones de un conjunto de datos. Una vez identificados los patrones, se pueden hacer predicciones con nuevos datos que se incorporen al sistema.

Conjunto de entrenamiento (**training**): es el conjunto de datos destinado a descubrir relaciones predictivas. Por ejemplo, los datos históricos de las compras de libros en una web online se pueden usar para analizar el comportamiento de los clientes en sus procesos de compra (títulos visitados, categorías, historial de compras…), agruparlos en patrones de comportamiento y hacer recomendaciones de compra a los clientes nuevos que siguen los patrones ya conocidos o aprendidos.

Modelo o hipótesis: Tras entrenar al sistema (es decir, tras detectar los patrones en los datos), se crea un modelo que servirá para hacer las predicciones. Podemos asimilar un modelo a un filtro en

el que entran datos nuevos y cuya salida es la clasificación de ese dato según los patrones que se han detectado en el entrenamiento. Por ejemplo, si se entrena un modelo con datos históricos de clientes para detectar el riesgo de baja de una tarjeta de crédito, el modelo clasificará a los nuevos clientes en función de su comportamiento para predecir el riesgo de baja.

Confianza: Es la probabilidad de acierto que calcula el sistema para cada una de las predicciones.

4.3. Categorías de Algoritmos de Machine Learning

El Machine Learning se ocupa de resolver una larga serie de problemas y tareas. Aunque las dos más conocidas son el Aprendizaje Supervisado y el No Supervisado que definimos en esta sección.

4.3.1. Aprendizaje Supervisado

El **aprendizaje supervisado** es una técnica que trata de predecir un comportamiento a partir de unos ejemplos anteriores denominados datos de entrenamiento. Los datos de entrenamiento (normalmente vectores) están compuestos de características y etiquetas. El objetivo de los algoritmos supervisados consiste en construir un estimador (deducir una función) capaz de predecir la etiqueta de cualquier objeto de entrada válida en función de su conjunto de características después de haber visto una serie de ejemplos, los datos de entrenamiento. Para ello, tiene que generalizar a partir de los datos situaciones no vistas previamente.

Para entender estos conceptos pensemos en los siguientes ejemplos de aprendizaje supervisado:

- Dados dos conjuntos de imágenes etiquetadas como perro y gato, determinar una función que aplicada a una nueva imagen clasifique si es un perro o un gato.
- Si contamos con cierta información sobre un conjunto de casas, y los precios correspondientes de cada una, podríamos *aprender* alguna regla de correspondencia entre casas y sus precios, que nos permita predecir el precio cuando tengamos información de nuevas casas.
- Dada una imagen multicolor de un objeto capturada por un telescopio, determinar si el objeto es una estrella o una galaxia.
- Dada una lista de películas que ha visto una persona y la puntuación que da a cada una, determinar una lista de películas que le gustaría ver.

Todas estas tareas tienen en común que hay una o más cantidades desconocidas sobre un objeto que tienen que ser determinadas a partir de otras cantidades observadas.

El aprendizaje supervisado trata dos tipos de problemas: regresión y clasificación.

- **Regresión**: Trata de predecir una etiqueta que es continua. El valor predicho por la salida del estimador (o función) es un valor real. Respondería a preguntas del tipo: ¿cuánto tiempo? ¿cuántas unidades? ¿cuánto pesa? ¿qué edad?...O el problema de predecir el precio de una casa, ya que requiere un resultado continuo.
- **Clasificación**: Trata de predecir una etiqueta que es discreta o categórica. Responde a preguntas del tipo ¿es el tumor canceroso? O por ejemplo si se quiere determinar si un email es spam o no, hay dos respuestas: spam y no spam. Por lo tanto, es una tarea de clasificación. El problema de predecir el precio de una casa en términos de clasificación sería determinar si una casa tiene precio "alto, "medio" o "bajo".

Formalmente, el problema de aprendizaje supervisado se puede definir de la manera siguiente:

X es el conjunto de objetos e *Y* el conjunto de etiquetas. Cada objeto está representado por un vector de *n* dimensiones (*n* características), $x^{(i)}$ denota al objeto *i* y x_i la característica *i* del objeto.

Tenemos además una función de dependencia entre los objetos y las etiquetas.

y: $X \rightarrow Y$, de forma que $y^{(i)} = y(x^{(i)})$, $i = 1,\ldots,l$ son valores conocidos de la función *y*

Cada pareja $(x^{(i)}, y^{(i)})$ representa un objeto, $x^{(i)}$, junto a su etiqueta, $y^{(i)}$. Esta pareja se le conoce como *training sample* o ejemplo de entrenamiento. Para el aprendizaje supervisado contamos con un **conjunto de entrenamiento**:

$$\left\{\left(x^{(i)}, y^{(i)}\right),\ldots,\left(x^{(m)}, y^{(m)}\right)\right\} \subset X$$

El conjunto de entrenamiento no es nada más que un grupo de objetos con sus respectivas etiquetas.

El problema que plantea el aprendizaje supervisado es encontrar una función h: $X \rightarrow Y$, denominada normalmente *hipótesis*, que permita aproximar el conjunto de objetos al de etiquetas. Para encontrar esta función *h* se utiliza un algoritmo de aprendizaje y el conjunto de datos de entrenamiento para entrenar al algoritmo. La función *h* permitirá predecir el valor desconocido $y^{(n)}$ para un nuevo objeto $x^{(n)}$.

En la Figura 4.3-1 se muestra un diagrama esquemático del aprendizaje supervisado.

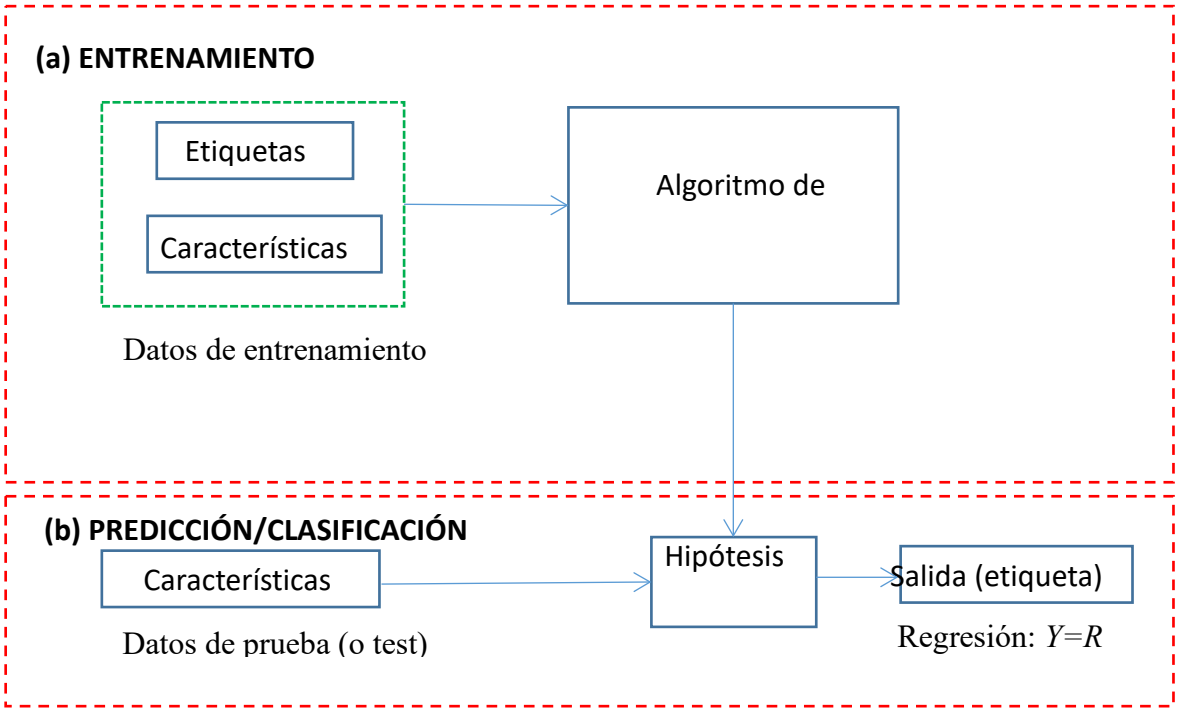


Figura 4.3-1: Diagrama aprendizaje supervisado.

Como se puede ver en el diagrama, el aprendizaje supervisado tiene como meta *aprender una hipótesis h mediante el entrenamiento de un algoritmo*. Consta de dos pasos esenciales:

(a) **Entrenamiento**, donde usamos el conjunto de datos de entrenamiento y un algoritmo que nos permita crear una hipótesis *h* o modelo.

(b) **Predicción o Clasificación**, donde utilizamos la hipótesis obtenida en el paso anterior para realizar nuevas predicciones con nuevos objetos (datos de prueba).

Todo este proceso se llama aprendizaje supervisado porque el proceso del algoritmo de aprendizaje con el conjunto de entrenamiento se puede pensar como un profesor que supervisa el proceso de aprendizaje. Sabemos las respuestas correctas y el algoritmo iterativamente hace predicciones sobre los datos de entrenamiento y es corregido por "el profesor". Se supervisa el entrenamiento del algoritmo, corrigiendo los parámetros de este según los resultados que obtengamos, de forma iterativa. El aprendizaje termina cuando el algoritmo logra una precisión aceptable.

Por ejemplo, si lo que queremos es predecir lo mejor posible el precio de mercado *p* de una casa podríamos tratarlo como un problema de clasificación supervisada considerando:

- Las características de una casa podrían ser su tamaño, el número de cuartos, los años de antigüedad, los baños, la fecha de la última reforma, etc. Por ejemplo, mediante (500,4) podríamos estar representando una casa de 500 metros cuadrados y 4 cuartos.
- En este caso el precio de la casa será la salida (etiqueta) que queremos estimar.
- El conjunto de entrenamiento no es más que un conjunto de casas con sus respectivos precios.
- Para una casa nueva, a partir de $x^{(m)}$, $x^{(m)}$ = (sus metros construidos, número de cuartos) el valor de $h(x^{(m)})$ debe aproximar lo mejor posible el precio de mercado p.

La Figura 4.3-2 muestra el concepto de aprendizaje supervisado para la clasificación de secuencias de ADN.

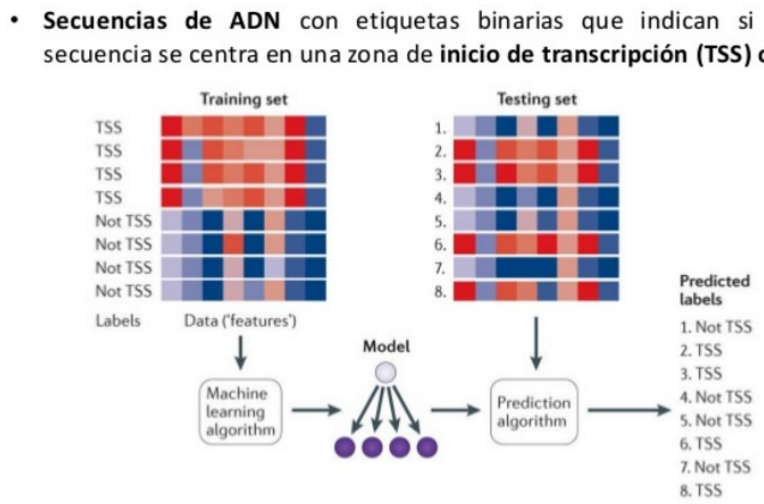


Figura 4.3-2: Aprendizaje supervisado para clasificación de secuencias de ADN.

Como resumen: en el aprendizaje supervisado los datos incluyen características y etiquetas. El objetivo de este tipo de aprendizaje es construir un estimador con capacidad predictiva sobre la etiqueta (denominada también variable objetivo) en función de un conjunto de características. Los tipos de problemas que trata son de clasificación (si la etiqueta es discreta) o de regresión (si es continuo el valor de la etiqueta).

4.3.2. Aprendizaje No Supervisado

El **aprendizaje no supervisado** trata de encontrar relaciones de similitud entre objetos, agrupando los objetos (datos) por similitud o encontrando relaciones entre ellos no evidentes "a simple vista". En este tipo de aprendizaje disponemos sólo de los datos de entrada (X) pero no de sus

correspondientes etiquetas. No se conoce, como en el aprendizaje supervisado, las etiquetas correctas del conjunto de datos de entrenamiento.

Este tipo de aprendizaje se denomina no supervisado porque al contrario del supervisado no hay "respuestas correctas" y no hay "profesor". Los algoritmos por sí solos descubren la estructura presente en los datos.

Los problemas de aprendizaje no supervisado se suelen agrupan en distintos tipos:

- **Clustering**: en un problema de clustering queremos identificar grupos en los datos. Por ejemplo, podemos pensar en agrupar compradores por su comportamiento en las compras.
- **Asociación**: En los problemas de asociación tratamos de encontrar reglas que describen gran parte de los datos. Por ejemplo, la gente que compra X también tiende a comprar Y.
- **Reducción de dimensionalidad**: Son algoritmos que pretenden proyectar los datos en un espacio de dimensión inferior. Se puede utilizar el aprendizaje no supervisado para determinar las combinaciones de medidas que mejor representan la estructura de los datos.

En el aprendizaje no supervisado se trata, por lo tanto, de encontrar patrones en el conjunto de objetos $x^{(i)}$, $i = 1,\ldots,n$. Como hemos comentado, en el aprendizaje no supervisado no se suministran categorías o etiquetas inicialmente a la máquina. No es una tarea de clasificación o predicción sino de organización para descubrir patrones en la estructura de los datos. Examinando la estructura hallada por la máquina es como se extraen los conceptos, abstracciones, etc. que explican el significado de los datos.

Por ello, los principales usos del aprendizaje no supervisado los podemos resumir en estos dos:

- Encontrar estructuras en los datos: ¿Cuáles son similares?
- Presentación descriptiva de los datos (no predictiva como en el aprendizaje supervisado).

Una aplicación bastante popular de este tipo de aprendizaje es la de crear grupos de población, por ejemplo, para segmentar compradores en diferentes grupos según algún criterio específico.

4.4. Técnicas de Machine Learning

Las técnicas que se pueden aplicar al procesado de datos son muchas y muy diferentes: para *generar predicciones, estimaciones o clasificaciones*. Desde técnicas de regresión logística hasta redes neuronales artificiales pasando por redes bayesianas, máquinas de vectores de soporte o árboles de decisión. Hay infinidad de métodos, fomentándose la cooperación entre personas con experiencia

en diferentes campos para obtener el mejor modelo posible. Varias de estas técnicas se encuadran dentro de lo que es el Machine Learning o Aprendizaje Automático.

En esta sección resumimos las más importantes: regresión, clasificación y clustering, indicando las características diferenciadoras entre ellas. Cada una de estas técnicas está implementada en multitud de algoritmos, y en este módulo veremos algunos de ellos. Conviene tener en cuenta que el proceso de resolver una tarea utilizando Machine Learning rara vez es un proceso en cascada y, en multitud de ocasiones, tendremos que retroceder varios pasos para probar diferentes estrategias sobre el conjunto de datos con diferentes algoritmos de Machine Learning.

4.4.1. Regresión

La **regresión** es una de las técnicas centrales del aprendizaje supervisado que trata de predecir valores reales (precios de casas, número de llamadas, total de ventas, etc.) a partir de los datos.

Un **modelo de regresión** permite describir cómo influye una variable *X* sobre otra variable *Y*. *X* es la variable independiente e *Y* la dependiente o respuesta. El objetivo es obtener estimaciones razonables de *Y* para distintos valores de *X* a partir de una muestra de *n* pares: $(x_1,y_1),....(x_n, y_n)$. Estos *n* pares son los datos de entrenamiento.

La regresión lineal es principalmente de dos tipos: regresión lineal simple y regresión lineal múltiple.

Sea cual sea el tipo de regresión que utilicemos, lo que nos debe quedar claro cuando hablamos de regresión es lo siguiente:

- La regresión es una medida de la relación entre una variable dependiente (por ejemplo, un coste) y los valores de una serie de variables independientes (por ejemplo, tiempo y Kg de material utilizado).
- La regresión consiste en estimar la relación entre estas variables.
- Regresión significa predecir la salida o resultado usando los datos de entrenamiento.

Por supuesto, la idea de ajustar los datos experimentales a una curva, no se limita a una recta. También se puede ajustar a un polinomio o realizar una regresión curvilínea.

En la sección 4.6 veremos el funcionamiento en R de algún algoritmo popular de regresión como es la regresión lineal o la regresión multilineal.

4.4.2. Clasificación

La **clasificación** es una técnica del aprendizaje supervisado que trata de predecir una clase (o valor categórico) a partir de algunas variables de entrada. En clasificación el número de posibles respuestas es finito ($|Y|<\infty$) y la finalidad es, por tanto, asignar una categoría de entre las definidas a nuestros datos. La Figura 4.4-1 muestra un esquema del modelo general para los métodos de clasificación.

En este caso tenemos una colección de registros (conjunto de entrenamiento o tabla de aprendizaje en la Figura 4.4-1). Cada registro tiene un conjunto de variables (atributos). Uno de los atributos será el objetivo de la clasificación (en este caso Fraude). El algoritmo de aprendizaje debe generar un modelo (una función) para predecir la clase a la que pertenecería otro registro. Esta asignación a una clase se debe hacer con la mayor precisión posible. El conjunto de prueba (tabla de testing) se utiliza para determinar la precisión del modelo. Por lo general el conjunto de datos se divide en dos conjuntos, el de entrenamiento y el de prueba.

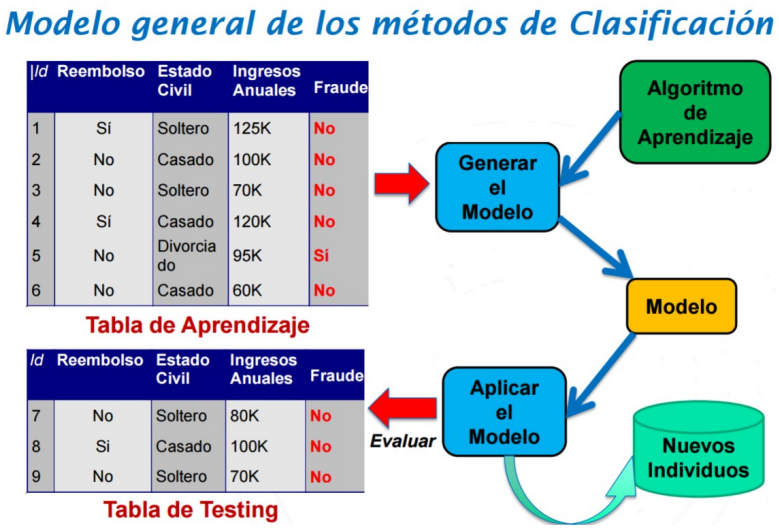


Figura 4.4-1: Modelo general de clasificación.

Ya hemos comentado en la sección anterior la tarea de clasificación que supone, por ejemplo, el determinar si un email es spam o no. Las dos categorías o clases en este caso serían: spam y no spam. Otro ejemplo de clasificación sería el predecir si el riesgo de un préstamo es “alto” o “bajo” en función de la edad, los ingresos, el capital pedido,… En la Tabla 4.4-1 se muestran otros posibles ejemplos de clasificación, pero la lista de aplicaciones y usos es interminable.

TAREAS DE CLASIFICACIÓN	SALIDAS
Identificación de lenguaje en documentos de texto	en, es, de, fr, ja, zh, ar, ru...
Categorizar nuevos artículos	De negocio, tecnológicos, deportes,...
Análisis de sentimientos en los compradores	Negativo, neutral, positivo
Reconocimiento facial	Misma persona/diferente persona
Reconocimiento de voz	Misma persona/diferente persona
Fuentes astronómicas	Clase o tipo

Tabla 4.4-1: Ejemplos de clasificaciones.

Aunque tanto la regresión como la clasificación son métodos supervisados ya hemos ido nombrando las diferencias de ambas técnicas. La Tabla 4.4-2 recoge las principales. En la Figura 4.4-2 se muestra una representación sencilla de posibles representaciones gráficas tras aplicar ambas técnicas a un conjunto de datos.

CLASIFICACIÓN	REGRESIÓN
Clasificar significa **agrupar** los resultados de salida en una clase.	Regresión significa **predecir el valor de la salida** utilizando los datos de entrenamiento.
Ejemplo: Usar clasificación para **predecir** el tipo de un tumor (maligno o benigno) a partir de los datos de entrenamiento.	Ejemplo: Usar regresión para **predecir** el precio de una vivienda a partir de los datos de entrenamiento.
Si la variable es **discreta/categórica** estamos ante un problema de clasificación.	Si nuestro objetivo es un **nº real/continuo** estamos ante un problema de regresión

Tabla 4.4-2: Clasificación vs. regresión.

En la sección 4.7 veremos el funcionamiento en R de algún algoritmo popular de clasificación como es el KNN (K-Nearest Neighbors).

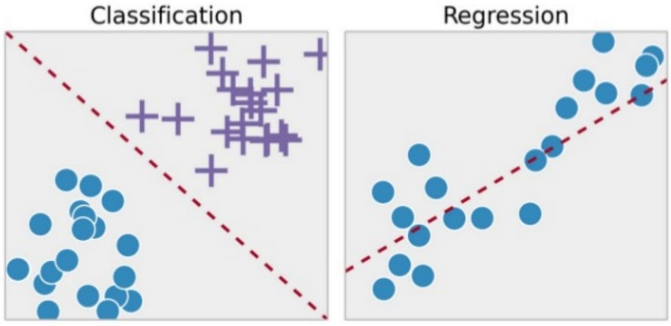


Figura 4.4-2: Clasificación vs. regresión.

4.4.3. Clustering

El **clustering** es una técnica del aprendizaje no supervisado que trata de agrupar los datos en grupos significativos. Los datos pertenecientes al mismo grupo (llamado *cluster*) son más similares entre sí. El clustering particiona los datos en grupos sin categorías/clases disponibles, de acuerdo con alguna medida de similitud predefinida, como por ejemplo la distancia euclídea (ver Figura 4.4-3). Sólo requiere instancias, pero no etiquetas.

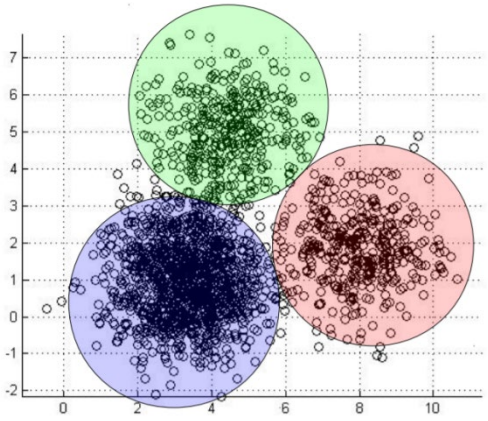


Figura 4.4-3: Clustering.

El objetivo de todo algoritmo de clustering es encontrar grupos tales que los datos en un cluster sean similares entre sí, y diferentes de los datos en otros clusters (ver Figura 4.4-4)

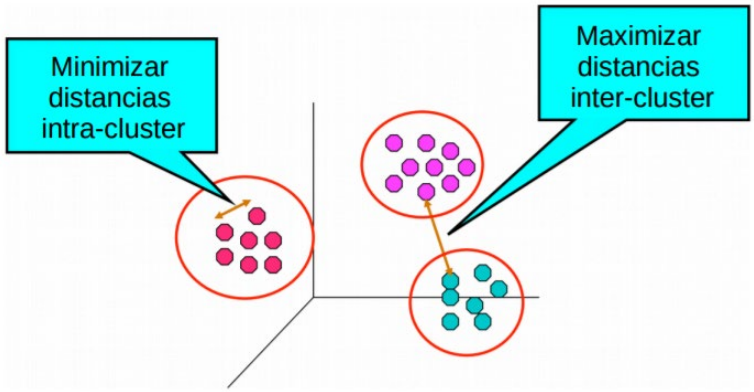


Figura 4.4-4: Objetivos del proceso de clustering.

Entre las aplicaciones más comunes de los procesos de clustering podemos nombrar estas:

- Construcción de perfiles de compradores para análisis de mercado.

- Agrupar documentos relacionados para explorarlos más rápido
- Reducir el tamaño de conjuntos de datos grandes (extracción de características)

En la sección 4.8 veremos el funcionamiento en R de algunos algoritmos populares de clustering como el K-means (K-medias) y el jerárquico.

4.5. Análisis de Componentes Principales

Sea cual sea la técnica de aprendizaje que utilicemos, cuando los datos provienen de aplicaciones reales requieren una etapa de preprocesamiento para limpiarlos, transformarlos o reducir su dimensionalidad. Podemos encontrarnos, por ejemplo, con datos incompletos (datos en los que faltan valores de ciertos atributos) o con datos ruidosos.

En la etapa de preprocesado podemos realizar distintas acciones. En el caso de datos ausentes podríamos: ignorar el dato, rellenar el valor ausente manualmente, usar una constante global para todos los valores ausentes, usar la media de los valores del atributo independientemente de la clase, usar la media de los valores del atributo de la misma clase,…

Si los datos ruidosos suponen un problema y no son un objetivo en la extracción de información, entonces se puede utilizar, por ejemplo, alguna de las técnicas de clustering para eliminar estos datos. También podríamos pensar en reducir el nivel de ruido de las tuplas ruidosas mediante técnicas de regresión: el valor ruidoso de la tupla se calcula mediante regresión de los vecinos.

Si lo que pretendemos en la tarea de preprocesado de los datos es reducir su volumen deberemos utilizar alguna metodología de selección de características (*feature selection*) o reducir su dimensionalidad utilizando alguna técnica como el Análisis de Componentes Principales o PCA (*Principal Component Analysis*).

El objetivo principal de esta sección es mostrar cómo funciona en R una de las técnicas más comunes en la reducción de datos: el PCA.

El PCA es una de las técnicas de reducción de dimensionalidad más utilizadas en análisis de datos. Permitirá reducir el número de características minimizando la pérdida de información. Reducir la dimensionalidad de un conjunto de datos puede ser muy útil. Pensemos, por ejemplo, en la visualización de datos cuando estos tienen más de tres dimensiones. En este caso podría ser interesante dibujar las distintas características dos a dos. Pero cuando hay muchas características puede ser más interesante, por ejemplo, hacer una proyección de los datos usando algún tipo de transformación lineal o no lineal, que nos permita resumir en un gráfico de dos dimensiones las

características más importantes de los mismos en cuanto a variabilidad. Por otro lado, el reducir la dimensión puede a veces implicar una reducción significativa del tiempo de cálculo de algunos algoritmos.

El objetivo del PCA (Principal Component Analysis) es buscar una representación alternativa y reducida de las tuplas originales formadas por *n* atributos. Se buscan los *k* vectores ortogonales (*k* < *n*) que mejor representan los *n* atributos originales. Por lo tanto, los atributos originales son proyectados en una representación de dimensionalidad más reducida. A diferencia de otros métodos en la técnica de PCA no hay descarte de características.

La técnica funciona construyendo automáticamente un conjunto de combinaciones lineales de las variables del conjunto de datos, con la máxima variabilidad posible. Matemáticamente, cada componente principal viene determinada por un vector propio de la matriz de covarianzas o correlaciones de los datos.

Como el PCA no es invariante de escala, es recomendable estandarizar las variables antes de aplicarlo.

EJEMPLO: Análisis de Componentes Principales

Todo el código de este ejemplo está disponible en el Script **PCA.R**

Vamos a utilizar en este ejemplo los datos `iris`. Los datos recogen, para 150 flores, medidas de cuatro variables continuas que corresponden a medidas de la longitud y anchura de pétalos y sépalos; y una variable categórica que describe la especie de flor (setosa, versicolor y virginica).

Para ver la estructura de nuestros datos podemos escribir lo siguiente:

```
data(iris)
head(iris)
```

```
  Sepal.Length Sepal.Width Petal.Length Petal.Width Species
1          5.1         3.5          1.4         0.2  setosa
2          4.9         3.0          1.4         0.2  setosa
3          4.7         3.2          1.3         0.2  setosa
4          4.6         3.1          1.5         0.2  setosa
5          5.0         3.6          1.4         0.2  setosa
6          5.4         3.9          1.7         0.4  setosa
```

Podemos explorar nuestros datos utilizando el comando `pairs()` que permite mostrar los gráficos de dispersión de las variables (ver Figura 4.5-1) ejecutando la instrucción:

```
pairs(iris[,1:4])
```

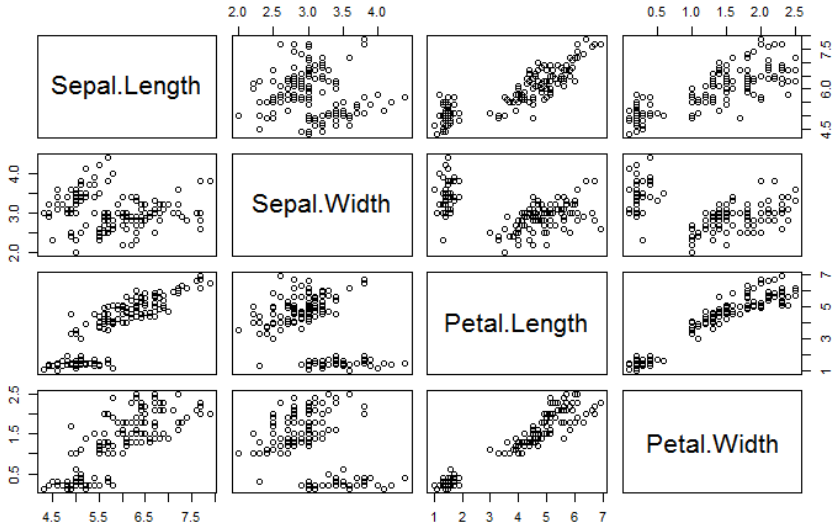


Figura 4.5-1: Gráficos de dispersión de los datos `iris`.

Para dibujar los datos etiquetados por especie (ver Figura 4.5-2) ejecutamos:

```
#Sustituyendo etiquetas
iris.especies<-c(rep("s",50),rep("c",50),rep("v",50))
#Diagramas de dispersión incluyendo etiquetas
pairs(iris[,1:4],panel=function(x,y,...) text(x,y,iris.especies))
```

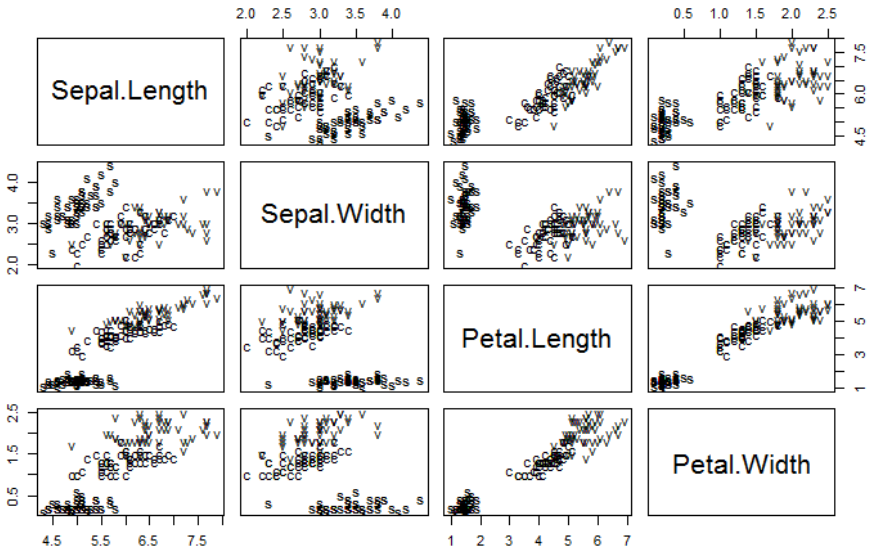


Figura 4.5-2: Gráficos de dispersión de los datos `iris` etiquetados por especies.

En este caso como las etiquetas de las especies son bastante largas (setosa, virginica y versicolor), las hemos sustituido por letras “s” para setosa, “v” para virginica y “c” para versicolor. Analizando la Figura 4.5-2 nos podemos dar cuenta de si las flores de una misma especie aparecen agrupadas con

flores de la misma especie. Las flores que aparezcan entre dos grupos corresponderán a flores difíciles de clasificar claramente en uno de los grupos.

Dibujar las variables 2 a 2 puede ser interesante. Pero veamos cómo aplicar PCA en este caso en el que trabajamos con 4 variables.

Vamos a aplicar PCA a las cuatro variables continuas y utilizaremos la variable categórica para visualizar después las componentes principales.

Tanto la asimetría en los datos como la distinta magnitud de las variables tiene influencia en el resultado del cálculo de las componentes principales. Por ello, suele ser buena práctica aplicar alguna transformación a los datos para centrar y escalar las variables antes de aplicar PCA. Entre las distintas transformaciones posibles en este caso vamos a aplicar una transformación `log` (tal y como se sugiere en Venables, W. N., Brian D. R. Modern applied statistics with S-PLUS. Springer-verlag). A continuación utilizamos la función `prcomp` para calcular el Análisis de Componentes principales, fijando a `TRUE` sus argumentos `center` y `scale` (para centrar y estandarizar las variables antes de aplicar PCA). Para realizar estas acciones tenemos que ejecutar el código siguiente:

```
# transformada log a las variables continuas
log.ir <- log(iris[, 1:4])
# ir.spacies contiene la variable especies
ir.species <- iris[, 5]

# applicando PCA - scale. = TRUE por defecto está a FALSE
ir.pca <- prcomp(log.ir, center = TRUE,scale = TRUE)
```

Vamos a analizar los resultados obtenidos. Podemos ver qué atributos devuelve el análisis utilizando la siguiente instrucción:

```
> attributes(ir.pca)
$names
[1] "sdev"     "rotation" "center"   "scale"    "x"

$class
[1] "prcomp"
```

Si aplicamos el método `print` sobre el objeto `ir.pca` que nos ha devuelto la función `prcomp` nos devuelve la desviación estándar de cada una de las cuatro componentes principales y su rotación, que son los coeficientes de las combinaciones lineales de las variables continuas. Esto lo obtenemos con la instrucción:

```
> print(ir.pca)
Standard deviations:
[1] 1.7124583 0.9523797 0.3647029 0.1656840
```

```
Rotation:
                    PC1         PC2        PC3         PC4
Sepal.Length  0.5038236 -0.45499872  0.7088547  0.19147575
Sepal.Width  -0.3023682 -0.88914419 -0.3311628 -0.09125405
Petal.Length  0.5767881 -0.03378802 -0.2192793 -0.78618732
Petal.Width   0.5674952 -0.03545628 -0.5829003  0.58044745
```

El valor de las desviaciones estándar lo podemos obtener también de la estructura:

```
ir.pca$sdev
[1] 1.7124583 0.9523797 0.3647029 0.1656840
```

¿Cómo determinamos el número de componentes necesarias para representar fidedignamente un conjunto de datos (por ejemplo, si el conjunto tiene dimensión 10, deberemos usar 2, 3, ¿cuántas componentes?).

El método summary describe la importancia de las componentes principales. Al ejecutar:

```
> summary(ir.pca)
Importance of components:
                          PC1    PC2     PC3     PC4
Standard deviation     1.7125 0.9524 0.36470 0.16568
Proportion of Variance 0.7331 0.2268 0.03325 0.00686
Cumulative Proportion  0.7331 0.9599 0.99314 1.00000
```

la primera fila nos da la desviación estándar asociada a cada componente principal (esto mismo se obtiene con `ir.pca$sdev`). La segunda fila muestra la proporción de la varianza en los datos explicados por cada una de las componentes y la tercera fila describe la suma acumulada de la varianza (lo que nos da una idea de la importancia relativa de cada componente).

Como vemos, con 1 componente se recoge aproximadamente el 73% de la variabilidad de los datos. Con 2 componentes se explica el 95% y con 3 ya se explica el 99%

La estructura `ir.pca$x` contiene las coordenadas de los puntos. Luego si queremos dibujar las dos primeras componentes (ver Figura 4.5-3) debemos ejecutar:

```
plot(ir.pca$x[,1:2])
```

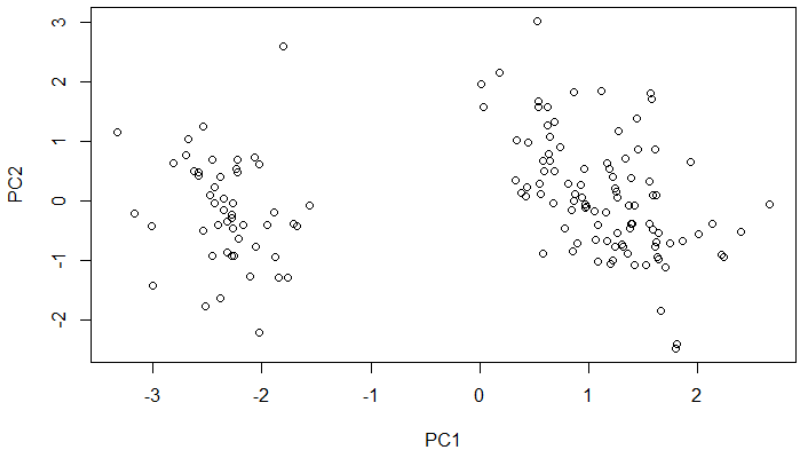


Figura 4.5-3: Gráfico de las dos primeras componentes principales.

Si queremos dibujar los puntos de distinto color para cada especie de flor (ver Figura 4.5-4) tendremos que ejecutar:

```
text(ir.pca$x[ir.species=="setosa",1:2],col=2,"s")
text(ir.pca$x[ir.species=="versicolor",1:2],col=3,"v")
text(ir.pca$x[ir.species=="virginica",1:2],col=4,"c")
```

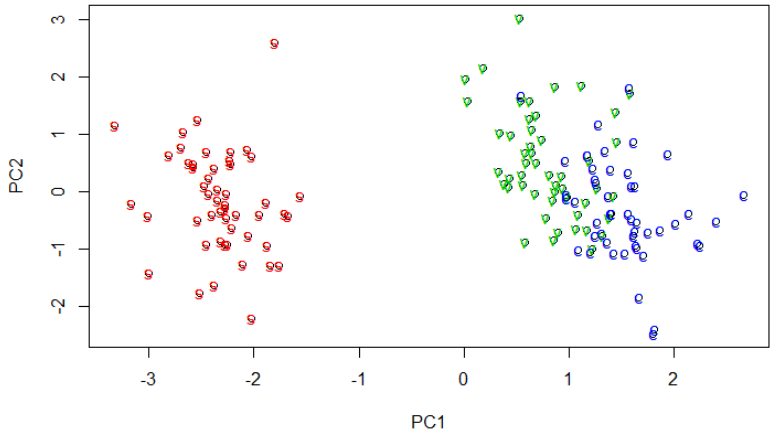


Figura 4.5-4: Gráfico de las dos primeras componentes principales por especies (setosa en rojo, versicolor en verde y virginica en azul).

Como se puede apreciar en el gráfico, hemos conseguido representar un conjunto de dimensión 4 mediante un conjunto de dimensión 2. Puede concluirse que hay 3 grupos, uno bien separado y otros dos más juntos.

La técnica del PCA la podemos emplear para cualquier conjunto de datos del que queramos hacernos una idea de la estructura. También puede utilizarse para codificar información, en el sentido de sustituir la codificación original de ciertos datos por otra que ocupe menos espacio.

Así por ejemplo, si se dispone de imágenes de fotos (en blanco y negro, representadas por niveles de gris), puede utilizarse la técnica de componentes principales para reducir la dimensión de las imágenes.

4.6. Aprendizaje supervisado 1: regresión

Como ya vimos en la sección 4.4.1, un **modelo de regresión** permite describir cómo influye una variable *X* sobre otra variable *Y*. *X* es la variable independiente e *Y* la dependiente o respuesta. El objetivo es obtener estimaciones razonables de *Y* para distintos valores de *X* a partir de una muestra de *n* pares: $(x_1,y_1),\ldots.(x_n, y_n)$. Estos *n* pares son los datos de entrenamiento. Vamos a ver cómo trata R la regresión lineal simple y la múltiple.

4.6.1. Regresión Lineal Simple

4.6.1.1 CÁLCULO DEL MODELO Y PREDICCIONES

La **regresión lineal simple** consiste en describir la relación entre las dos variables mediante una recta. En este caso, la función que proponemos para modelar la relación es:

$$f(x) = a + b \times x$$

donde *a* y *b* son constantes desconocidas. El problema consiste en ajustar la recta que represente los datos de la mejor manera (ver Figura 4.6-1)

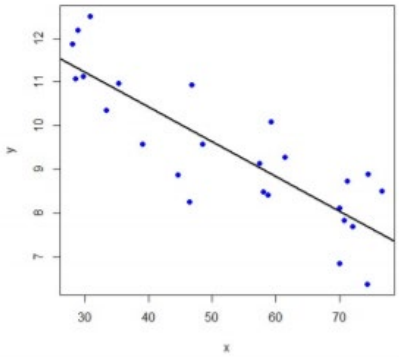


Figura 4.6-1: Ajuste por regresión lineal.

Para *a* (ordenada en el origen o intercept) y *b* (pendiente o slope) fijos, los valores ajustados o predichos ($\hat{y}_i$) y los residuos (e_i) están definidos por:

$$\hat{y}_i = a + b \times x_i$$
$$e_i = y_i - \hat{y}_i$$

El objetivo es hallar los mejores coeficientes *a* y *b* que representan la relación lineal entre las variables. A esos valores, denominados estimadores, los vamos a denotar con $\hat{a}$ y $\hat{b}$, respectivamente.

Una vez hallada la recta, es decir, calculados $\hat{a}$ y $\hat{b}$, tenemos que los valores ajustados en cada punto (ver Figura 4.6-2) son:

$$\hat{y}_i = \hat{a} + \hat{b} \times x_i$$

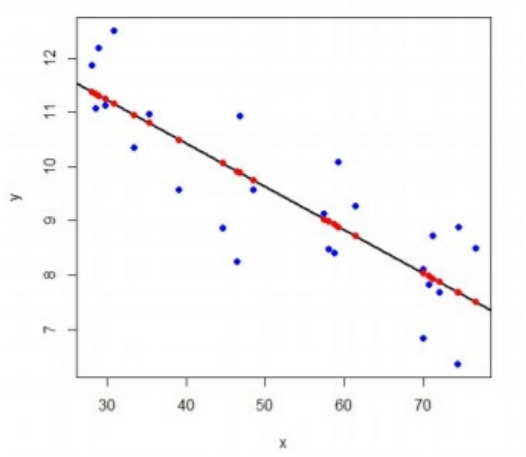


Figura 4.6-2: Valores ajustados sobre la recta de cada punto.

Gráficamente, lo que se resuelve para calcular $\hat{a}$ y $\hat{b}$ es la minimización de las distancias entre los valores observados y los valores predichos (ver Figura 4.5-3).

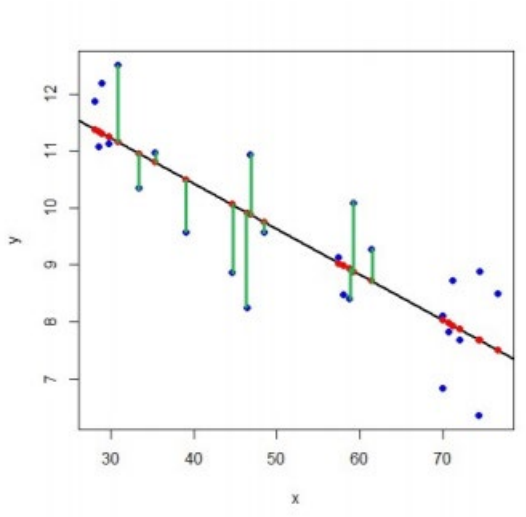


Figura 4.6-3: Minimización de distancias entre valores observados y predichos.

Para predecir cualquier valor para cualquier x_i una vez obtenida la recta $\hat{y}_i = \hat{a} + \hat{b} \times x_i$ tenemos que sustituir los valores que hemos determinado de $\hat{a}$, $\hat{b}$ y el valor de la nueva variable regresora x_i que queremos predecir en la recta calculada y calcular el valor predicho $\hat{y}_i$. Veamos cómo funciona todo esto utilizando R en el siguiente ejemplo.

EJEMPLO: Regresión lineal simple

Todo el código referente a regresión lo podemos encontrar en el fichero **Ejemplo regresion lineal simple.R.**

Supongamos que disponemos de unos datos correspondientes a tres variables medidas en 25 individuos: su edad, peso y grasas en sangre. Estos datos se recogen en el fichero **EdadPesoGrasas.txt** .

Importamos el fichero y vemos cuales son los nombres de sus variables:

```
EdadPesoGrasas <- read.csv("D:/DATOS/EdadPesoGrasas.txt", sep="")
names(EdadPesoGrasas)
[1] "peso"   "edad"   "grasas"
```

Con el fin de conocer las relaciones existentes entre cada par de variables en nuestros datos, podemos representar una *matriz de diagramas de dispersión*. Para ello se utiliza el siguiente comando en R:

```
pairs(EdadPesoGrasas)
```

La representación la mostramos en la Figura 4.6-4. Podemos apreciar que existe una relación lineal bastante clara entre la `edad` y las `grasas`, pero no entre los otros dos pares de variables. Por otra parte, el fichero contiene un dato atípico. Por esto nos planteamos el calcular un modelo de regresión lineal simple entre las variables `edad` y `grasas`.

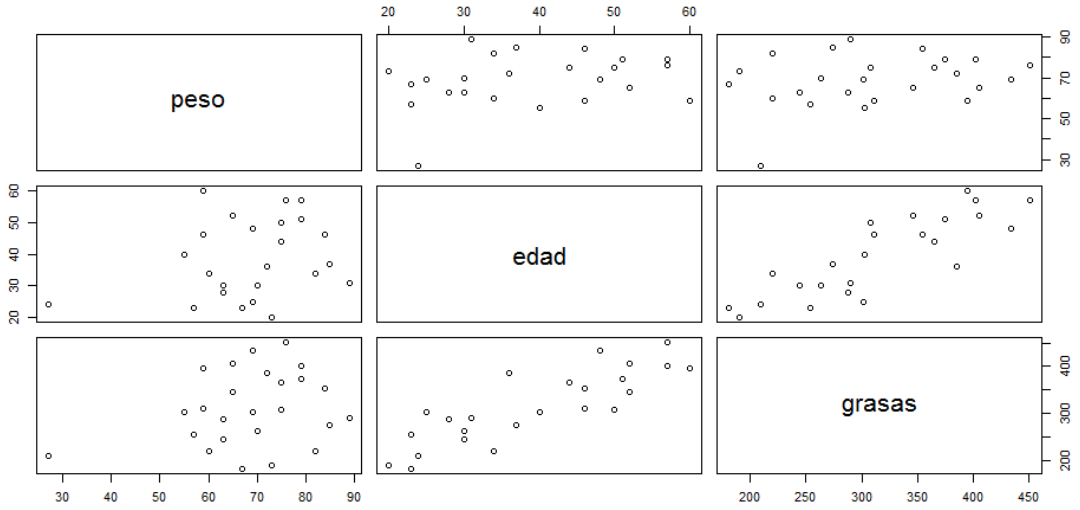


Figura 4.6-4: Diagramas de dispersión.

Para cuantificar el grado de relación lineal, calculamos la *matriz de coeficientes de correlación* de la forma siguiente:

```
> cor(EdadPesoGrasas)
             peso      edad    grasas
peso    1.0000000 0.2400133 0.2652935
edad    0.2400133 1.0000000 0.8373534
grasas 0.2652935 0.8373534 1.0000000
```

Cálculo y representación de la recta

En R el comando básico para calcular la estimación de los parámetros $\hat{a}$ y $\hat{b}$, entre otra información, es `lm` (*linear models*). El primer argumento de este comando es una fórmula `y ~ x` en la que se especifica cuál es la variable respuesta o dependiente (`y`), y cuál es la variable regresora o independiente (`x`). El segundo argumento, llamado `data` especifica cuál es el fichero en el que se encuentran las variables.

En nuestro caso, para calcular el modelo de regresión lineal simple entre las variables `grasas` y `edad` y el resumen ejecutamos el código siguiente:

```
> regresion <- lm(grasas ~ edad, data = EdadPesoGrasas)
> summary(regresion)

Call:
lm(formula = grasas ~ edad, data = EdadPesoGrasas)

Residuals:
    Min      1Q  Median      3Q     Max
-63.478 -26.816  -3.854  28.315  90.881
```

```
Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept) 102.5751    29.6376   3.461  0.00212 **
edad          5.3207     0.7243   7.346 1.79e-07 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 43.46 on 23 degrees of freedom
Multiple R-squared:  0.7012,  Adjusted R-squared:  0.6882
F-statistic: 53.96 on 1 and 23 DF,  p-value: 1.794e-07
```

El resultado de la regresión lineal lo hemos guardado en un objeto llamado `regresion`. Este objeto es una lista que contiene toda la información relevante sobre el análisis. Mediante el comando `summary` obtenemos un resumen de los principales resultados.

Los parámetros de la ecuación de la recta de mínimos cuadrados que relaciona la cantidad de grasas en la sangre en función del peso vienen dados por la columna `Estimate` de la tabla `Coefficients` de la salida del `summary` anterior.

Por lo tanto, en este ejemplo la ecuación de la recta de ajuste por mínimos cuadrados es: $y = 102.575 + 5.321\,x$

La gráfica de la Figura 4.6-5 representa la nube de puntos (comando `plot`) y añade la recta de mínimos cuadrados (comando `abline` aplicado al objeto generado por `lm`). Para dibujarlas se han ejecutado los comandos siguientes:

```
plot(EdadPesoGrasas$edad, EdadPesoGrasas$grasas, xlab = "Edad", ylab =
"Grasas")
abline(regresion)
```

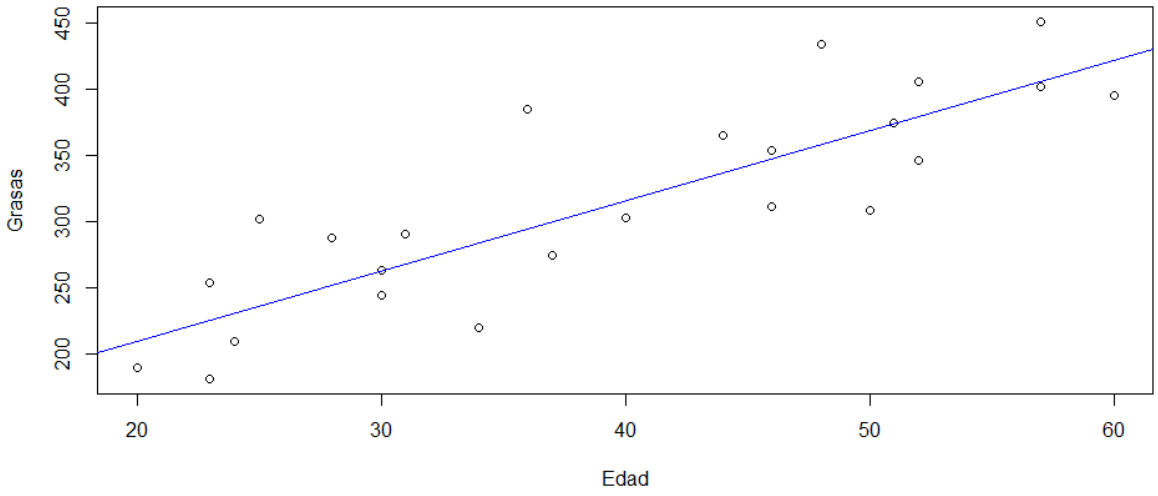


Figura 4.6-5: Recta calculada mediante regresión lineal.

El **coeficiente de correlación al cuadrado** mide la bondad del ajuste de la recta a los datos. A partir de la salida obtenida con la función `summary`, vemos que su valor en este caso es Multiple R-squared: 0.701. Si este coeficiente es igual a 1 significa que los puntos están exactamente sobre la recta. Cuánto más cerca esté de 1, más próximos estarán los puntos de la recta.

Cálculo de predicciones

Supongamos que queremos utilizar la recta calculada para predecir la cantidad de grasas para individuos de edades 30, 31, 32,..., 50. Basta crear un data frame que contenga las nuevas variables regresoras y usar el comando `predict`:

```
nuevas.edades <- data.frame(edad = seq(30, 50))
predict(regresion, nuevas.edades)
```

```
       1        2        3        4        5        6        7        8        9       10       11       12
262.1954 267.5161 272.8368 278.1575 283.4781 288.7988 294.1195 299.4402 304.7608 310.0815 315.4022 320.7229
      13       14       15       16       17       18       19       20       21
326.0435 331.3642 336.6849 342.0056 347.3263 352.6469 357.9676 363.2883 368.6090
```

Por ejemplo, para un individuo de 30 años, predecimos una cantidad de grasas de 262.2

4.6.1.2 INFERENCIA EN EL MODELO DE REGRESIÓN LINEAL SIMPLE

En esta sección vamos a ver algunos conceptos para que ver qué tipo de validaciones se pueden realizar en esta clase de modelos. No profundizamos en la parte teórica, simplemente queremos que sea una muestra de lo que R permite hacer.

Una vez que tenemos el modelo ajustado procedemos con su diagnóstico, analizando para ello los distintos valores que nos ha devuelto la llamada a la función `summary(regresion)`.

Expresado matemáticamente, lo que suponemos ahora es que los datos proceden de un modelo de regresión simple de la forma:

$$\hat{y}_i = \hat{a} + \hat{b} \times x_i + \varepsilon_i, i = 1..n$$

donde $\varepsilon_i, i = 1..n$ son errores aleatorios a los que se supone ciertas propiedades como, por ejemplo, la independencia.

Bajo este modelo, los **errores típicos** de los estimadores de los parámetros $\hat{a}$ y $\hat{b}$ se encuentran en la columna `Std Error` de la salida de `summary (regresion)` que veíamos antes. En el ejemplo, sus valores son 29.638 y 0.724 respectivamente.

Una posible duda podría ser: ¿Son los coeficientes $\hat{a}$ y $\hat{b}$ significativos? Podemos responder a esta pregunta analizando los **p-valores** que devuelve R. La columna `t value` de la salida de `summary(regresion)` contiene el **estadístico t**, es decir, el cociente entre cada estimador y su error típico. Estos cocientes son la base para calcular los p-valores. Los correspondientes p-valores aparecen en la última columna `Pr(>|t|)` de la tabla `Coefficients` de la salida de `summary(regresion)`. En este caso, como sólo tenemos dos coeficientes estimados (*a* y *b*), nos devuelve 2 valores. En otro caso nos devolvería un p-valor por cada uno de los coeficientes involucrados en el problema de estimación. Si el valor del p-valor es muy pequeño (menor de 0.05), el coeficiente estimado se considera significativo, siendo en ese caso el coeficiente distinto de cero. En nuestro ejemplo, los p-valores son muy pequeños (0.00212 y 1.79×10^{-7}) por lo que podemos suponer $a \neq 0$ y $b \neq 0$. Como además habíamos visto que el modelo lineal estaba bastante bien atendiendo al valor del Multiple R-squared, entonces la recta dada por $y = 102.575 + 5.321\,x$ es buena para modelar la relación entre *X* e *Y*. Si por el contrario, hubiésemos tenido un p-valor alto para la ordenada al origen (por ejemplo, de más de 0.20), entonces esto hubiese sugerido probar con el modelo lineal $f(x) = bx$.

El estimador de la **desviación típica de los errores** aparece como `Residual standard error` y su valor en el ejemplo es 43.5

Los **intervalos de confianza** para los coeficientes estimados dan una medida de la precisión de las estimaciones realizadas. Se obtienen con el comando `confint`. El parámetro `level` permite elegir el nivel de confianza (por defecto es 0.95). Dos ejemplos de cálculo de estos intervalos son los siguientes:

```
> confint(regresion)
                2.5 %     97.5 %
(Intercept) 41.265155 163.885130
edad         3.822367   6.818986

> confint(regresion, level = 0.9)
                 5 %       95 %
(Intercept) 51.780153 153.370132
edad         4.079335   6.562018
```

La incertidumbre en una predicción la podemos evaluar calculando lo que se conoce como **bandas de confianza**, que reflejan la incertidumbre en la línea de regresión y la incertidumbre sobre las futuras observaciones. Estas bandas se pueden obtener usando el comando `predict`. Por ejemplo, el siguiente código calcula y representa los dos tipos de intervalos para el rango de edades que va de 20 a 60 años. La Figura 4.6-6 muestra las bandas calculadas en este código.

```
nuevas.edades <- data.frame(edad = seq(20, 60))
# Grafico de dispersion y recta
```

```
plot(EdadPesoGrasas$edad, EdadPesoGrasas$grasas, xlab = "Edad", ylab =
"Grasas")
abline(regresion)

# Intervalos de confianza de la respuesta media: ic es una matriz con
#tres columnas: la primera es la prediccion, las otras dos son los
#extremos del intervalo
ic <- predict(regresion, nuevas.edades, interval = "confidence")
lines(nuevas.edades$edad, ic[, 2], lty = 2)
lines(nuevas.edades$edad, ic[, 3], lty = 2)

# Intervalos de prediccion
ic <- predict(regresion, nuevas.edades, interval = "prediction")
lines(nuevas.edades$edad, ic[, 2], lty = 2, col = "red")
lines(nuevas.edades$edad, ic[, 3], lty = 2, col = "red")
```

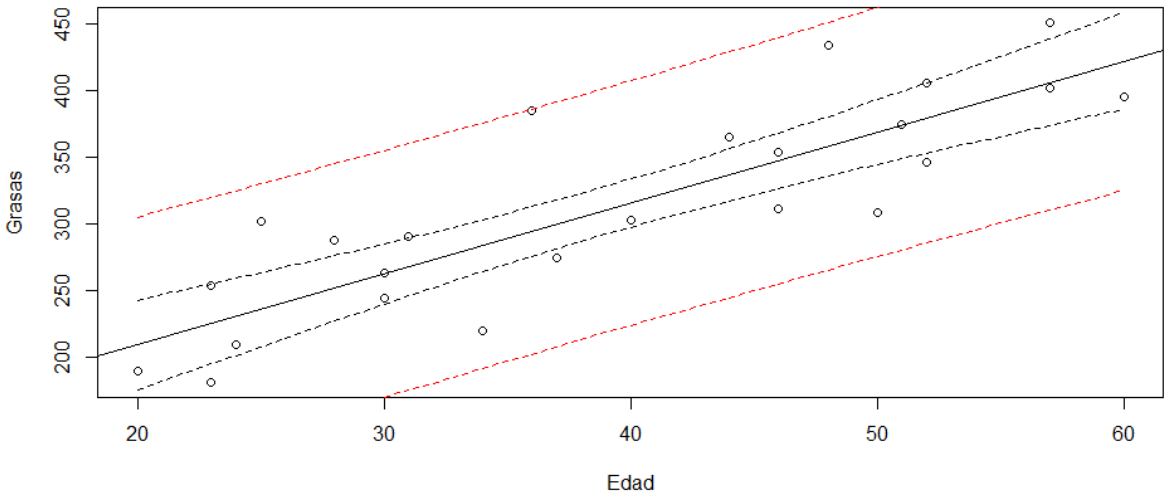


Figura 4.6-6: Intervalos de confianza y predicción (en rojo).

La **tabla de análisis de la varianza** de los errores se obtiene con el comando `anova`:

```
> anova(regresion)
Analysis of Variance Table

Response: grasas
          Df Sum Sq Mean Sq F value    Pr(>F)
edad       1 101933  101933  53.964 1.794e-07 ***
Residuals 23  43444    1889
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

Para finalizar, comentamos que para comprobar si se verifican las hipótesis en el ajuste de un modelo de regresión lineal, se suele utilizar el análisis de sus residuos. Sin entrar en detalles teóricos, lo deseable es que tengan distribución normal para el modelo lineal estimado. Vamos a ver un par de gráficos que suelen aportar información sobre esto.

Los valores predichos $\hat{y}_i$ y los residuos $e_i = y_i - \hat{y}_i$ se pueden obtener con los comandos `residuals` y `fitted` respectivamente. Los residuos estandarizados se obtienen con `rstandard` (los residuos estandarizados nos permiten distinguir residuos grandes porque tienen media cero y varianza próxima a 1).

Un gráfico útil para llevar a cabo el diagnóstico del modelo es la representación de los residuos estandarizados frente a los valores ajustados (ver Figura 4.6-7). En R lo podemos obtener mediante el siguiente código:

```
#Diagnóstico del modelo: residuos estandarizados frente a los valores
#ajustados
residuos <- rstandard(regresion)
valores.ajustados <- fitted(regresion)
plot(valores.ajustados, residuos)
```

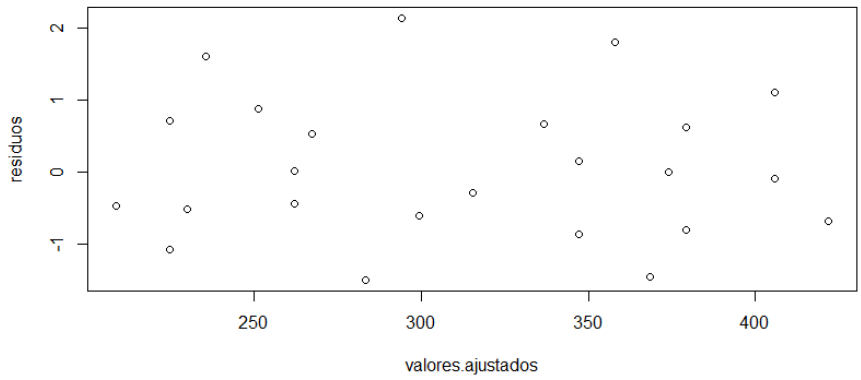


Figura 4.6-7: Residuos estandarizados vs. valores ajustados.

No se observa ningún patrón especial, por lo que tanto la homocedasticidad (el garantizar que la dispersión de datos es constante) como la linealidad, ambas hipótesis de este tipo de modelos, resultan hipótesis razonables.

La hipótesis de normalidad se suele comprobar mediante un QQ plot de los residuos (ver Figura 4.6-8). Esta gráfica sirve para revisar los cuantiles con respecto al comportamiento de los cuantiles en una distribución normal. El siguiente código en R sirve para obtenerlo:

```
# Hipótesis de normalidad
qqnorm(residuos)
qqline(residuos)
```

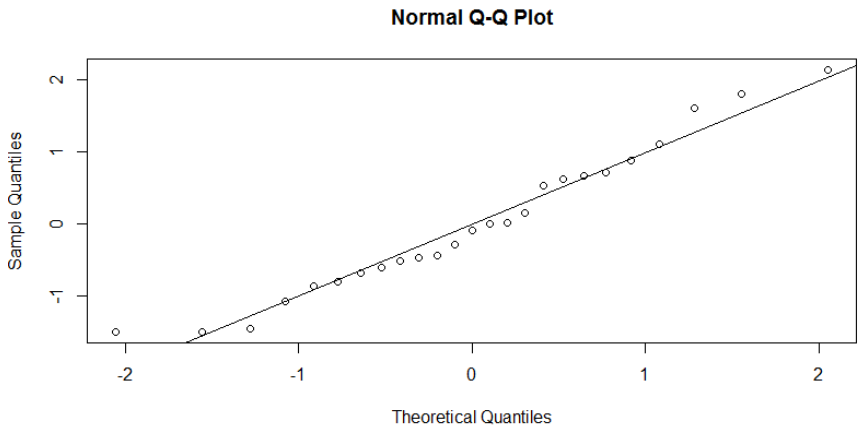


Figura 4.6-8: QQ plot de los residuos.

Dado que los puntos están bastante alineados, la normalidad también parece aceptable.

****

EJERCICIO 1: Regresión lineal

En el fichero **Venecia.txt** tenemos almacenadas 50 observaciones anuales acerca del nivel del mar en Venecia entre los años 1931 y 1981. Estos datos son datos reales publicados en la referencia: Smith R. L, "Extreme value theory based on the r largest annual events", Journal of Hydrology, 86 (1986).

Calcular y analizar un modelo de regresión lineal simple para estos datos entre las variables `año` y `nivel`. Para ello:

a) Dibujar la recta de ajuste del modelo junto con la correspondiente nube de puntos.
b) Obtener un resumen del ajuste.
c) ¿Cuánto vale el coeficiente de correlación al cuadrado en este caso?
d) ¿Cuánto valen los estimadores de todos los parámetros del modelo?
e) Calcula un intervalo de confianza con un nivel del 90%
f) Calcular y representar los intervalos de confianza y de predicción al 95% del nivel medio para los años comprendidos entre 1940 y 1970.
g) Calcular los residuos estandarizados frente a los valores ajustados y verificar la hipótesis de normalidad.

****

4.6.2. Regresión Lineal Múltiple

La regresión lineal simple se puede generalizar a más de una variable para explicar el comportamiento de los datos de salida. A este tipo de modelos se les llama**, modelos de regresión lineal múltiple.** Como su nombre sugiere, este tipo de regresión se caracteriza por múltiples (más de una) variables independientes. En este caso tratamos de ajustar los datos a una recta representada por la ecuación:

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \cdots + \beta_p X_p$$

donde *Y* es la variable dependiente o estimada, X_i las variables independientes (o predictores), β_i los parámetros estimadores que miden la influencia de las variables independientes sobre *Y* y β_0 el término independiente.

En particular modelos del tipo $Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2^2$ también entran en este tipo de modelo (basta con coger $X_1 = 1, X_2 = X$ y $X_3 = X^2$). En realidad, podemos hacer cualquier tipo de ajuste polinomial de este estilo, como veremos a continuación en un ejemplo.

Para la regresión lineal múltiple, R utiliza los mismos comandos que los vistos para la regresión lineal simple. La única diferencia radica en la expresión de la fórmula que especifica el modelo de interés en `lm`. Vamos a ilustrarlo con un ejemplo.

EJEMPLO: Regresión lineal múltiple

Vamos a realizar un ajuste polinomial para los datos históricos de Galileo. En 1609 Galileo demostró matemáticamente que la trayectoria de un cuerpo que cae con una componente de velocidad horizontal es una parábola. Su descubrimiento tuvo su origen en observaciones empíricas que realizó casi un año antes. Para estas observaciones, ideó un experimento en el que una bola empapada de tinta rodaba en un plano inclinado para luego caer desde una altura de 500 punti (1 punti=0.0944 cm). Galileo estudió la distancia horizontal que alcanza la bola en función de la altura desde la que sale. Un diagrama ilustrativo, extraído de Ramsey, Schafer (2002), “The statistical Sleuth” p. 268, se muestra en la Figura 4.6-9.

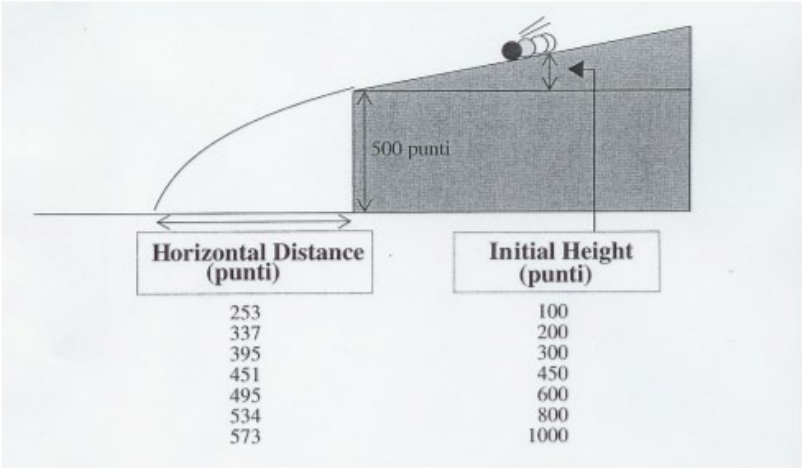


Figura 4.6-9: Diagrama ilustrativo del experimento de Galileo.

Los datos con los que vamos a trabajar en este ejemplo se encuentran en el fichero **galileo.txt**. Así que en primer lugar los importamos:

```
galileo.dat <-
read.table(file="D:/DATOS/galileo.txt",header=F,col.names=c("d",
"h"));galileo.dat #no hay linea de cabecera
attach(galileo.dat)
```

Tenemos dos variables, la distancia horizontal `d` y la altura `h`. La Figura 4.6-10 muestra los datos dibujados y el ajuste a una recta. Como podemos apreciar, un ajuste de este tipo para los datos que tenemos sería realmente malo. El código utilizado ha sido:

```
plot(d,h,xlab = "Distancia (d)",ylab = "Altura (h)")
```

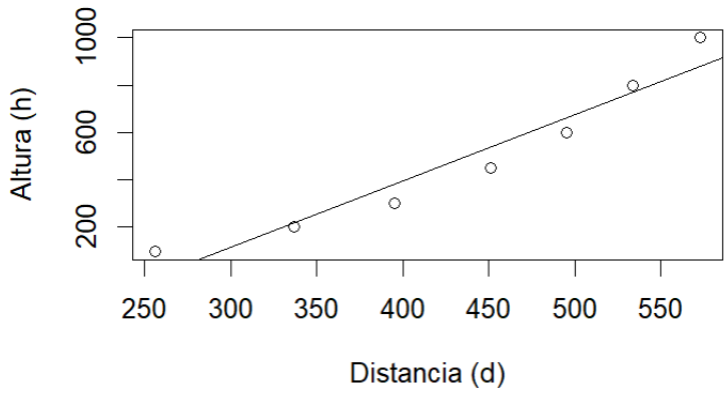


Figura 4.6-10: Datos de galileo.txt y ajuste a una recta.

Podemos probar a ajustar un polinomio de grado 2 o de grado 3, por ejemplo. El modelo sería $d = a_0 + a_1h + a_2h^2$ o $d = a_0 + a_1h + a_2h^2 + a_3h^3$.

```
# Ajuste entre d, h y h^2 de la forma: d=ao+a1*h+a2*h^2;
h2=h^2
galileo.lm=lm(d~h+h2)#expresa que d será un modelo en función de h y
h2
```

Otra forma de hacerlo sin definir previamente el valor de `h2` es:

```
galileo.lm=lm(d~h+I(h^2))
```

En este caso hemos utilizado en la fórmula la notación `I(h^2)` para indicar que queremos incluir en el modelo la potencia de grado 2 de `h`.

Podemos obtener las características del modelo utilizando la llamada:

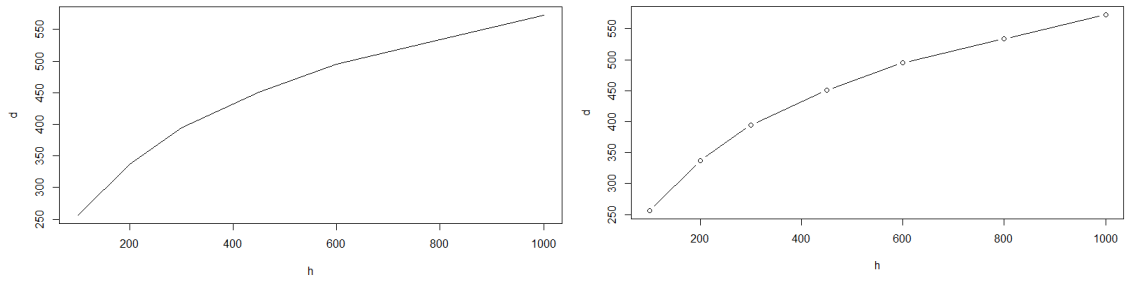
```
summary(galileo.lm)

Call:
lm(formula = d ~ h + h2)

Residuals:
      1       2       3       4       5       6       7
-13.235   8.018  12.996   2.071  -5.724 -12.249   8.123

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept)  2.028e+02  1.580e+01  12.832 0.000213 ***
h            6.983e-01  7.055e-02   9.899 0.000585 ***
h2          -3.362e-04  6.296e-05  -5.340 0.005926 **
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 12.86 on 4 degrees of freedom
Multiple R-squared:  0.9913,   Adjusted R-squared:  0.9869
F-statistic: 227.6 on 2 and 4 DF,  p-value: 7.586e-05
```

La interpretación de la salida es la misma que en el caso de la regresión lineal simple. En este caso, nuestro modelo sería:

$$d = 202.8 + 0.6983h - 3.62 \times 10^{-4} h^2$$

El valor del Multiple R-squared es muy bueno: 0.9913. La última columna, `Pr(>|t|)`, nos permite ver si los coeficientes son significativos. Si es más pequeño de 0.05 se considera significativo (recordamos que, cuando el p-valor es muy pequeño el coeficiente es significativo). Si es mayor que 0.2 quitamos la variable del modelo (pero hay que hacerlo uno a uno, recalculando todos los p-valores en cada paso.)

Podemos realizar una gráfica con puntos conectados por líneas o las líneas y los puntos como se muestra en la Figura 4.6-11



(a) `plot(h,d,type="l")` (b) `plot(h,d,type="b")`

Figura 4.6-11: Puntos de galileo.txt conectados (a) por líneas y (b) por líneas y puntos.

El modelo teórico calculado, proporcionado por la ecuación calculada es: `galileo.lm$fitted`. Podemos dibujarlo (en rojo en la Figura 4.6-12) con el código siguiente:

```
lines(h,galileo.lm$fitted,col="red")
```

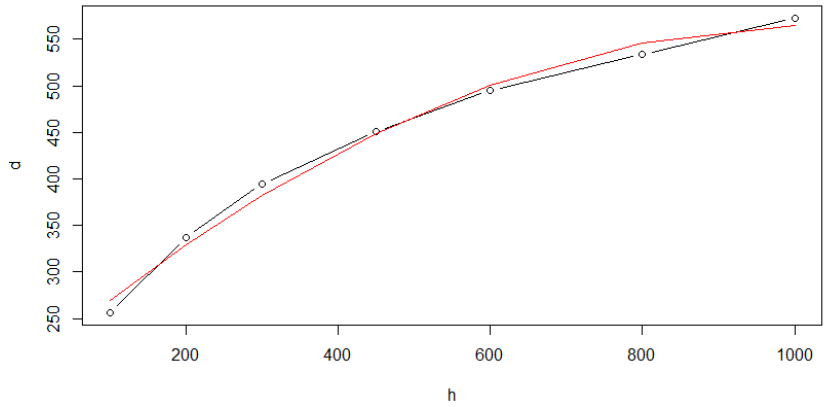


Figura 4.6-12: Modelo teórico de orden 2 calculado (en rojo).

El siguiente código correspondería al ajuste de los datos a un modelo de orden 3 y 4. En la Figura 4.6-13 se muestran los modelos calculados.

```
#modelo de orden 3
h2=h^2
h3=h^3
galileo.lm=lm(d~h+h2+h3)#expresa que d será un modelo en función de h, h2 y h3
lines(h,galileo.lm$fitted,col="blue") # con esto añadimos a la gráfica los valores del modelo ajustado de grado 3.

#################
#modelo de orden 4
h2=h^2
h3=h^3
h4=h^4
```

```
galileo.lm=lm(d~h+h2+h3+h4)#expresa que d será un modelo en función de
h h2 , h3 y h4
lines(h,galileo.lm$fitted,col="green")
summary(galileo.lm)
```

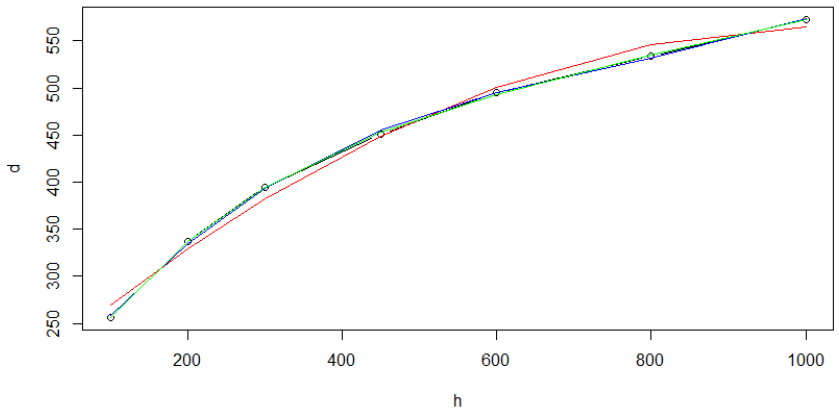


Figura 4.6-13: Modelo teórico de orden 3 (en azul) y de orden 4 (en verde).

****

EJERCICIO 2: Regresión lineal múltiple

En el fichero **granizados.txt** tenemos almacenadas una serie de datos que se pretenden analizar para identificar qué factores son los más influyentes a la hora de consumir granizados. En concreto, los datos corresponden a mediciones realizadas en una familia durante 30 semanas (entre el 18 de marzo de 1953 y el 11 de julio de 1953) del consumo por semana de granizado por cada persona (`y`). Aparte de `y`, otras variables almacenadas en el fichero que se pensaba que podían tener alguna influencia sobre el consumo son:

`p`: precio de una pinta de granizado
`i`: ingresos semanales de la familia
`temp`: temperatura media de la semana
`SEMANA`: número de semana

Teniendo en cuenta esta información queremos realizar lo siguiente:

a) Representar gráficamente el consumo de granizados en función de las semanas.
b) Determinar la matriz de correlación de las variables `y`, `p`, `i` y `temp`. Para calcular la matriz de correlación se puede utilizar la función `cor`.
c) ¿Cuál es la variable que parece tener más influencia en `y`?

d) Realizar un ajuste lineal de `y` sobre `p`, `i` y `temp`. Dibujar la evolución predicha por nuestro modelo. Analizar si todas las variables son igual de significativas en el modelo.
e) ¿Cuánto vale la varianza residual y el R2?
f) Realizar un ajuste lineal de `y` sobre `i` y `temp`. ¿Cuánto vale en este caso la varianza residual y R2?

****

4.7. Aprendizaje supervisado 2: clasificación

La **clasificación** es una forma de **aprendizaje supervisado. Clasificar** es la tarea de predecir el valor de una variable categórica a partir de las variables de entrada (características o predictores). La clasificación trata de construir modelos que separan datos en clases distintas a partir de conjuntos de datos de entrenamiento (datos para los que las clases están pre-etiquetadas y a partir de los cuales aprende el algorimo). Esta sección presenta utilizando R uno de los algoritmos de clasificación más populares: KNN (K-Nearest Neighbors).

KNN pertenece a un tipo de clasificadores denominado lazy learner, "vago". Hay otros algoritmos, como el SVM, que pertenecen a otro tipo denominado eager learner, "impaciente". Los clasificadores tipo eager construyen un modelo (clasificación) antes de recibir ningún dato perteneciente al conjunto de predicción. Por ejemplo SVM construye el modelo con el conjunto de datos de entrenamiento.

En los clasificadores tipo lazy, la construcción del modelo se retrasa hasta que se demanda la predicción. Mientras tanto solo almacena los datos suministrados.

4.7.1. Algoritmo K Vecinos más Próximos (KNN)

La idea del algoritmo KNN (*K-Nearest Neighbors*) usado para clasificación es la siguiente. El nuevo objeto se clasificará en la clase más frecuente de sus K vecinos más próximos. Una representación gráfica se presenta en la Figura 4-7.1. En ella se consideran dos valores de K diferentes. Si k=3 (círculo más pequeño en la Figura 4-7.1), el círculo verde será clasificado como triángulo rojo. Si k=5, entonces el círculo verdes será clasificado como cuadrado azul. De esta manera, es la mayoría de clases en los vecinos la que determinará la clase del objeto que estamos clasificando.

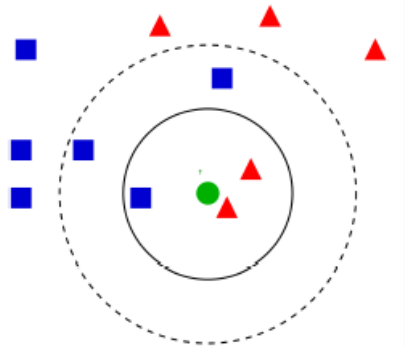


Figura 4-7.1: Clasificación por KKN del círculo verde para k= 3 y k = 5.

Para resolver los casos de empate, se puede añadir alguna regla heurística como puede ser el vecino más próximo. Dicha regla marca que, en caso de empate, será el vecino más próximo el que determine la clase del nuevo objeto.

Si la elección es k = 1, el objeto a clasificar es asignado a la clase del vecino más próximo.

Si k es demasiado pequeño, entonces el resultado es muy sensible a puntos ruidosos. Pero si k es demasiado grande, entonces la vecindad del punto desconocido incluirá muchos puntos de otras clases lejanas. Como regla para la elección de k podemos decir que debe ser grande para disminuir la probabilidad de una mala clasificación, pero pequeño en comparación con el número de puntos.

Hay algoritmos clasificatorios que basan la clasificación de cada nuevo caso en dos tareas. Primero se induce el modelo clasificatorio (*inducción*), y posteriormente, se deduce la clase del nuevo caso (*deducción*). Sin embargo, en KNN ambas tareas están unidas (*transducción*).

Podemos resumir el algoritmo de la siguiente manera:

Datos: D={(x_1, c_1), …, (x_n, c_n) } datos.
Objetivo: X =(x_1, …, x_n) nuevo dato a clasificar
Algoritmo:

- Para todo objeto ya clasificado (x_i, c_i)
  Calcular distancia d_i = (x_i, X)
- Ordenar d_i (i = 1,…, N) en orden ascendente
- Quedarnos con los k casos D_X^k ya clasificados más cercanos a X
- Asignar a X la clase más frecuente en D_X^k

EJEMPLO: Algoritmo KNN

El código de este ejemplo lo podemos encontrar en el script **practClasificacion.R**

Vamos a utilizar en este ejemplo el conjunto de datos `iris` que ya vimos en el ejemplo de PCA. Iremos realizando este ejemplo paso a paso.

1) Cargar datos de UC Irvine Machine Learning Repository: Podemos cargarlos del sitio web `http://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data` mediante la instrucción:

```
iris <- read.csv(url("http://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data"), header = FALSE)
```

Esta instrucción importa los datos y los salva en el dataframe `iris`. `header` lo ponemos a `FALSE` porque los datos cargados no tienen cabecera. Si queremos añadir cabecera a los datos basta con ejecutar:

```
names(iris) <- c("Sepal.Length", "Sepal.Width", "Petal.Length", "Petal.Width", "Species")
```

2) Inspeccionando los datos: Una vez cargados los datos en RStudio nos debemos hacer una idea de nuestros datos. Aunque sobre este conjunto de datos ya trabajamos en parte la sección dedicada a PCA, inspeccionaremos aquí parte de su estructura y características. Para visualizar los datos en la consola (aunque ocupa mucha parte de ella) basta con ejecutar:

```
iris
```

Podemos inspeccionar parcialmente la estructura de los datos utilizando las instrucciones:

```
head(iris)
  Sepal.Length Sepal.Width Petal.Length Petal.Width     Species
1          5.1         3.5          1.4         0.2 Iris-setosa
2          4.9         3.0          1.4         0.2 Iris-setosa
3          4.7         3.2          1.3         0.2 Iris-setosa
4          4.6         3.1          1.5         0.2 Iris-setosa
5          5.0         3.6          1.4         0.2 Iris-setosa
6          5.4         3.9          1.7         0.4 Iris-setosa

str(iris)
'data.frame':  150 obs. of  5 variables:
 $ Sepal.Length: num  5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...
 $ Sepal.Width : num  3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 ...
 $ Petal.Length: num  1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 ...
 $ Petal.Width : num  0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1 ...
 $ Species     : Factor w/ 3 levels "Iris-setosa",..: 1 1 1 1 1 1 1 1 1 1 ...
```

Analizando la información devuelta por estas instrucciones nos damos cuenta de que los datos son150 observaciones de 5 variables. Además el comando `str` permite distinguir el tipo de datos `num` y los tres niveles del atributo `Species`. Dicho de otra forma, el conjunto de datos contiene características sobre flores y una clasificación de ellas por especies. La variable especie en este caso es nuestra variable objetivo, es decir, esta variable (`Species`) determinará los resultados de la clasificación basándonos en las otras cuatro variables numéricas (ancho y largo de sépalos y pétalos).

Un vistazo al atributo `Species` nos muestra la división de las especies de flores en 50-50-50. Esto lo podemos hacer ejecutando la instrucción:

```
table(iris$Species)
```

```
    Iris-setosa Iris-versicolor  Iris-virginica
             50              50              50
```

Si queremos ver los resultados de la división de flores porcentualmente debemos ejecutar:

```
round(prop.table(table(iris$Species)) * 100, digits = 1)
```

```
    Iris-setosa Iris-versicolor  Iris-virginica
           33.3            33.3            33.3
```

`round` redondea los valores del primer argumento, `prop.table(table(iris$Species))*100`, al número especificado de dígitos (segundo parámetro), que es 1 dígito en este caso después del punto decimal.

Para hacernos una idea de los datos podemos dibujar los diagramas de dispersión (scatter plots). Este diagrama nos da una idea de la influencia entre variables y sus correlaciones.

Ejecutamos en primer lugar:

```
plot(iris$Sepal.Length,iris$Sepal.Width,col=factor(iris$Species))
```

Con lo que se obtiene el diagrama de la Figura 4.7-2.

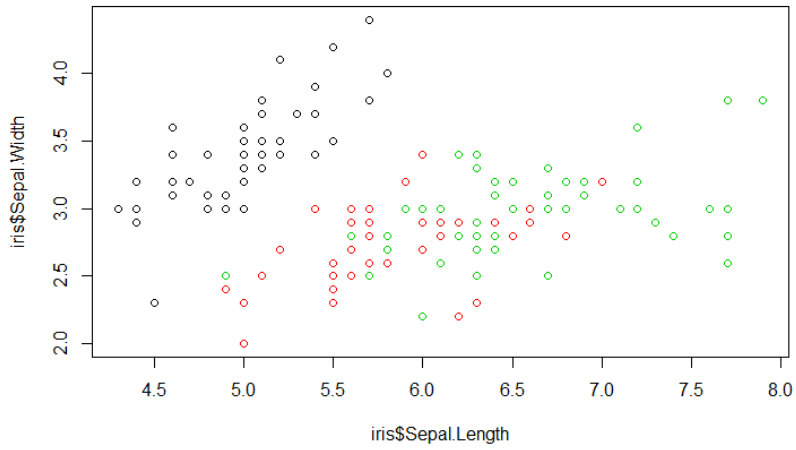


Figura 4-7.2: Sépalos: Setosa en negro, virginica en verde y versicolor en rojo.

Vemos que hay una alta correlación entre la longitud y la anchura del sépalo en las flores de especie Setosa (en negro). Para las flores Virginica (en verde) y Versicolor (en rojo) la correlación es menor.

Análogamente podemos dibujar (ver Figura 4-7.3) el diagrama de dispersión entre la longitud y anchura del pétalo ejecutando la instrucción:

```
plot(iris$Petal.Length,iris$Petal.Width,col=factor(iris$Species))
```

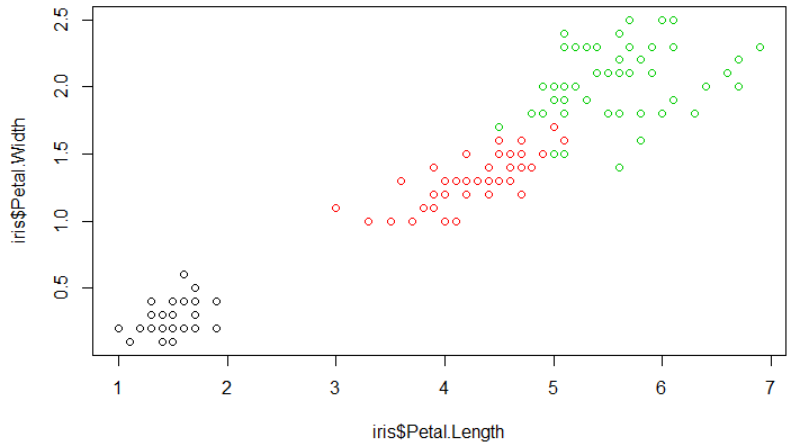


Figura 4-7.3: Pétalos: Setosa en negro, virginica en verde y versicolor en rojo.

Este diagrama nos informa de la correlación entre la longitud y anchura del pétalo para cualquiera de las especies del conjunto de datos `iris` (setosa en negro, virginica en verde, y versicolor en rojo).

Podemos saber más sobre los datos utilizando la función `summary()`. Esta función nos da el valor mínimo, el primer cuartil, mediana, media, tercer cuartil y máximo valor del conjunto de datos `iris` para los datos de tipo numérico.

```
summary(iris)
```

```
 Sepal.Length    Sepal.Width     Petal.Length    Petal.Width          Species
Min.   :4.300   Min.   :2.000   Min.   :1.000   Min.   :0.100   Iris-setosa    :50
1st Qu.:5.100   1st Qu.:2.800   1st Qu.:1.600   1st Qu.:0.300   Iris-versicolor:50
Median :5.800   Median :3.000   Median :4.350   Median :1.300   Iris-virginica :50
Mean   :5.843   Mean   :3.054   Mean   :3.759   Mean   :1.199
3rd Qu.:6.400   3rd Qu.:3.300   3rd Qu.:5.100   3rd Qu.:1.800
Max.   :7.900   Max.   :4.400   Max.   :6.900   Max.   :2.500
```

Para la variable categórica `Species` devuelve el número de datos de cada una. Podemos también refinar el resultado de `summary` con atributos específicos. Por ejemplo:

```
summary(iris[c("Petal.Width", "Sepal.Width")])
  Petal.Width     Sepal.Width
 Min.   :0.100   Min.   :2.000
 1st Qu.:0.300   1st Qu.:2.800
 Median :1.300   Median :3.000
 Mean   :1.199   Mean   :3.054
 3rd Qu.:1.800   3rd Qu.:3.300
 Max.   :2.500   Max.   :4.400
```

En este caso hemos añadido la funcion `c()` a la instrucción `summary`: las columnas `Petal.Width` y `Sepal.Width` se concatenan y se pide un `summary` solo de esas dos columnas del conjunto de datos `iris`.

3) Hacia dónde dirigirnos. Una vez que tenemos estudiados cómo son nuestros datos, tendríamos que pensar qué cosas podríamos aprender de ellos. Qué tipo de algoritmos podemos aplicar a los datos para obtener los resultados que creamos que se pueden obtener. De manera general, cuanto más nos familiaricemos con los datos más fácil será lograr casos de uso para los datos y más fácil será encontrar el algoritmo de Machine Learning adecuado.

En nuestro caso, como adelantábamos antes (y por la sección en la que nos encontramos) vamos a utilizar el conjunto de datos `iris` para clasificación. El atributo `Species` será la variable objetivo de la clasificación o variable que queremos predecir en este ejemplo.

4) Instalar los paquetes adecuados. Muchos de los algoritmos utilizados en Machine Learning no se incorporan por defecto en R. Lo más probable es que necesitemos descargar los paquetes que queramos utilizar. Podemos acceder a https://www.rdocumentation.org/ para realizar una búsqueda, bien introduciendo como término de búsqueda el nombre del algoritmo que queremos utilizar o bien el nombre del paquete.

Para ilustrar el algoritmo KNN en este ejemplo trabajaremos con el paquete `class`.

```
library(class)
```

Si no tenemos el paquete instalado tendremos que instalarlo previamente:

```
install.packages('class')
```

5) Preparar los datos:

Normalización de datos

Es práctica habitual normalizar los datos y transformarlos para que los valores de las distintas variables tengan una escala común.

Normalmente se necesita normalizar los datos cuando los rangos (valores entre máximo y mínimo) de las variables (atributos) son muy diferentes. Esto lo podemos comprobar viendo los resultados que nos devuelve la función `summary`. En el caso de los datos `iris` no sería necesaria la normalización. Si vemos en el resultado de `summary(iris)` el atributo `Sepal.Length` tiene valores entre 4.3 y 7.9 y el `Sepal.Width` entre 2 y 4.4. Por su parte `Petal.Lenght` tiene valores de 1 a 6.9 y `Petal.Width` entre 0.1 y 2.5, lo que se puede considerar aceptable.

De todas maneras, como es práctica habitual normalizar los datos, veamos cómo se hace.

La función de normalización la podemos definir en R de la manera siguiente:

```
normalize <- function(x) {
return ((x - min(x)) / (max(x) - min(x))) }
```

Ejecutando esta función normalizamos los atributos numéricos del conjunto de datos. En lugar de normalizar cada una de las 4 variables (o atributos) individualmente podemos hacerlo de manera conjunta ejecutando:

```
iris_norm <- as.data.frame(lapply(iris[1:4], normalize))
```

En esta instrucción guardamos los resultados de la normalización en el dataframe `iris_norm` utizando `as.data.frame()`. La función `lapply()` devuelve una lista de la misma longitud que el conjunto de datos de entrada. Cada elemento de la lista es el resultado de aplicar el argumento `normalize` al conjunto de entrada. En nuestro caso hemos aplicado el argumento `normalize` a los 4 atributos numéricos del conjunto de datos `iris` (`Sepal.Length, Sepal.Width, Petal.Length, Petal.Width`).

Para ver el efecto de la normalización en el conjunto de datos podemos comparar el resultado del `summary` siguiente, obtenido con los datos normalizados del conjunto de datos `iris` con el obtenido anteriormente para los datos sin normalizar.

```
summary(iris_norm)
  Sepal.Length     Sepal.Width      Petal.Length     Petal.Width
 Min.   :0.0000   Min.   :0.0000   Min.   :0.0000   Min.   :0.00000
 1st Qu.:0.2222   1st Qu.:0.3333   1st Qu.:0.1017   1st Qu.:0.08333
 Median :0.4167   Median :0.4167   Median :0.5678   Median :0.50000
 Mean   :0.4287   Mean   :0.4392   Mean   :0.4676   Mean   :0.45778
 3rd Qu.:0.5833   3rd Qu.:0.5417   3rd Qu.:0.6949   3rd Qu.:0.70833
 Max.   :1.0000   Max.   :1.0000   Max.   :1.0000   Max.   :1.00000
```

Datos de Entrenamiento (training) y Datos de Prueba (test)

El algoritmo KNN se aplica a un conjunto de datos de entrenamiento y los resultados se validan después con un conjunto de datos de prueba. Por ello, necesitamos dividir el conjunto de datos en dos partes: el conjunto de entrenamiento (training set) y el de prueba (test set). Como su nombre indica, el primer conjunto se utiliza para entrenar el sistema, mientras que el segundo se utilizará para evaluarlo.

En la práctica, la división de los datos en estos dos conjuntos depende de los datos y del problema que se trate. Normalmente se suelen tomar 2/3 de los datos originales como datos de entrenamiento y el 1/3 restante como datos de prueba.

En el caso de nuestros datos `iris`, si observamos los datos podemos ver que si realizamos la división de datos conforme los datos están ordenados en nuestro dataframe, el conjunto de datos de entrenamiento contendría especies de “Setosa” y “Versicolor”, pero no tendríamos ningún dato de la especie “Virginica”. Si hacemos esto, el modelo clasificador que obtengamos clasificaría cualquier flor como “Setosa” o como “Versicolor”, pero ninguna como “Virginica”. De esta manera, el modelo clasificaría a todas las flores como “Setosa” o “Versicolor”, puesto que no sabría que en los datos puede haber el tercer tipo de especie de flor, “Virginica”. Esto es, obtendríamos predicciones incorrectas para el conjunto de datos de prueba.

Por lo tanto, necesitamos asegurarnos de que los tres tipos de especies de flor están presentes en los datos de entrenamiento. Y, lo que, es más la cantidad de flores de las tres especies en los datos de entrenamiento debería estar más o menos en la misma proporción que en los datos originales. Veamos cómo hacer esto.

Lo primero que hacemos para determinar cuáles son los datos de entrenamiento y cuáles los datos de prueba es fijar una semilla. Esto no es más que un número del generador de números aleatorio

de R. La mayor ventaja de fijar una semilla es que podemos obtener la misma secuencia de números aleatorios siempre que proporcionemos la misma semilla en el generador de número aleatorio. En nuestro caso vamos a fijar la siguiente semilla:

```
set.seed(1234)
```

A continuación, queremos asegurarnos de que tenemos la misma proporción de especies en nuestros datos de entrenamiento y de prueba. Para conseguir esto utilizamos la función `sample()`. Esta función toma una muestra de tamaño fijado por el número de filas del conjunto de datos `iris`, esto es 150. Usamos `sample` con reemplazamiento de esta manera:

```
ind <- sample(2, nrow(iris), replace=TRUE, prob=c(0.67, 0.33))
ind
  [1] 1 1 1 1 2 1 1 1 1 1 2 1 1 2 1 2 1 1 1 1 1 1 1 1 2 1 2 2 1 1 1
1 1 1 2 1 1 2 2
 [41] 1 1 1 1 1 1 2 1 1 2 1 1 2 1 1 1 1 2 1 2 2 1 1 1 1 2 1 1 1 1 1 2
1 2 1 1 1 1 1 1
 [81] 2 1 1 1 1 2 1 1 1 2 1 2 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 2 1
2 1 1 2 2 1 1 2
[121] 2 2 2 2 1 1 1 1 1 1 2 1 1 1 2 1 2 1 1 2 1 2 1 1 1 1 2 1 2 1
```

con esto elegimos un vector de 2 elementos de forma que asignamos o 1 o 2 a las 150 filas del conjunto de datos iris. Además, las asignaciones de elementos las hacemos sujetos a probabilidades de 0.67 y 0.33. Esto se debe a que queremos que los datos de entrenamientos sean 2/3 del conjunto de datos original (por eso asignamos "1" con una probabilidad de 0.67 y "2" con una probabilidad de 0.33 en las 150 filas).

Podemos utilizar la variable `ind` así construida para dividir nuestro conjunto de datos en los datos de prueba y de entrenamiento de la manera siguiente:

```
iris.training <- iris[ind==1, 1:4]
iris.test <- iris[ind==2, 1:4]
```

En los datos de prueba y de entrenamiento no hemos incluido la variable objetivo de la clasificación, `Species`. Las únicas variables almacenadas en los datos de prueba y entrenamiento son los cuatro atributos `Sepal.Length`, `Sepal.Width`, `Petal.Length` y `Petal.Width`. Debemos incluir también el atributo `Species` en el algoritmo KNN para que podamos predecir sobre él. Para almacenar la etiqueta de las clases (las especies de flores) tanto en el conjunto de datos de prueba como en los de entrenamiento utilizamos el código siguiente:

```
iris.trainLabels <- iris[ind==1, 5]
iris.testLabels <- iris[ind==2, 5]
```

El código anterior coge la columna `Species` (columna 5 del data frame `iris`) y crea los data frame `iris.trainLabels` e `iris.testLabels` que almacenan las etiquetas (especies de flores) para los datos de entrenamiento y prueba respectivamente.

6) Construcción del Clasificador: Modelo KNN

Una vez que tenemos ya los datos preparados estamos en condiciones de comenzar el "aprendizaje". Una manera de aplicar el algoritmo KNN es utilizar la función `knn()` para entrenar un modelo. Esta función está incluida en el paquete `class` y utiliza la distancia euclídea como medida de distancia para encontrar los k vecinos más próximos al nuevo dato, siendo k un parámetro a fijar por el usuario.

Para construir el clasificador ejecutamos la siguiente instrucción, en la que clasificamos los datos de prueba:

iris_pred <- knn(train = iris.training, test = iris.test, cl = iris.trainLabels, k=3)

Con esta instrucción se almacena en `iris_pred` el resultado de la función `knn()` que tiene como argumento el conjunto de datos de entrenamiento (`iris.training`), el conjunto de datos de prueba (`iris.test`), las etiquetas del conjunto de datos de entrenamiento (`iris.trainlabels`) y la cantidad de vecinos que queremos fijar en el algoritmo (`k`). El resultado de la llamada a `knn()` es un factor que contiene las clases predichas para cada uno de los datos de entrenamiento. En nuestro caso:

```
iris_pred
 [1] Iris-setosa     Iris-setosa     Iris-setosa     Iris-setosa     I
ris-setosa
 [6] Iris-setosa     Iris-setosa     Iris-setosa     Iris-setosa     I
ris-setosa
[11] Iris-setosa     Iris-setosa     Iris-versicolor Iris-versicolor I
ris-versicolor
[16] Iris-versicolor Iris-versicolor Iris-versicolor Iris-versicolor I
ris-versicolor
[21] Iris-versicolor Iris-versicolor Iris-versicolor Iris-versicolor I
ris-virginica
[26] Iris-virginica  Iris-virginica  Iris-virginica  Iris-versicolor I
ris-virginica
[31] Iris-virginica  Iris-virginica  Iris-virginica  Iris-virginica  I
ris-virginica
[36] Iris-virginica  Iris-virginica  Iris-virginica  Iris-virginica  I
ris-virginica
Levels: Iris-setosa Iris-versicolor Iris-virginica
```

Podemos observar que las etiquetas de los datos de prueba no se han utilizado. Las utilizaremos en la evaluación del modelo para ver si predice de manera adecuada.

7) Evaluación del Modelo

Una vez construido el modelo, en Machine Learning es muy importante la evaluación de su comportamiento. Esto es, deberemos comprobar la precisión de nuestro modelo en las predicciones que realiza. En nuestro ejemplo tendríamos que comprobar si los valores predichos en `iris_pred` coinciden (o en qué grado) con `iris.testLabels`

```
> iris_pred
 [1] Iris-setosa     Iris-setosa     Iris-setosa     Iris-setosa     I
ris-setosa
 [6] Iris-setosa     Iris-setosa     Iris-setosa     Iris-setosa     I
ris-setosa
[11] Iris-setosa     Iris-setosa     Iris-versicolor Iris-versicolor I
ris-versicolor
[16] Iris-versicolor Iris-versicolor Iris-versicolor Iris-versicolor I
ris-versicolor
[21] Iris-versicolor Iris-versicolor Iris-versicolor Iris-versicolor I
ris-virginica
[26] Iris-virginica  Iris-virginica  Iris-virginica  Iris-versicolor I
ris-virginica
[31] Iris-virginica  Iris-virginica  Iris-virginica  Iris-virginica  I
ris-virginica
[36] Iris-virginica  Iris-virginica  Iris-virginica  Iris-virginica  I
ris-virginica
Levels: Iris-setosa Iris-versicolor Iris-virginica
> iris.testLabels
 [1] Iris-setosa     Iris-setosa     Iris-setosa     Iris-setosa     I
ris-setosa
 [6] Iris-setosa     Iris-setosa     Iris-setosa     Iris-setosa     I
ris-setosa
[11] Iris-setosa     Iris-setosa     Iris-versicolor Iris-versicolor I
ris-versicolor
[16] Iris-versicolor Iris-versicolor Iris-versicolor Iris-versicolor I
ris-versicolor
[21] Iris-versicolor Iris-versicolor Iris-versicolor Iris-versicolor I
ris-virginica
[26] Iris-virginica  Iris-virginica  Iris-virginica  Iris-virginica  I
ris-virginica
[31] Iris-virginica  Iris-virginica  Iris-virginica  Iris-virginica  I
ris-virginica
[36] Iris-virginica  Iris-virginica  Iris-virginica  Iris-virginica  I
ris-virginica
Levels: Iris-setosa Iris-versicolor Iris-virginica
```

El modelo realiza predicciones con bastante precisión. Salvo la fila 29, en la que la clasificación falla puesto que se ha predicho "Versicolor" mientras que la etiqueta es "Virginica", todas las demás predicciones son correctas. Esto es un primer indicador del funcionamiento correcto del clasificador.

Para asegurarnos del funcionamiento correcto del clasificador podemos utilizar la función `CrossTable()` del paquete `gmodels`. Debemos primero importar el paquete `gmodels`:

```
install.packages("gmodels")
```

Si ya tenemos el paquete instalado basta con hacer:

```
library(gmodels)
```

Una vez instalado debemos calcular la tabla de tabulación cruzada o contingencia (este tipo de tabla se suele utilizar para entender la relación entre dos variables). En nuestro caso queremos entender cómo las clases de nuestro conjunto de datos de prueba, almacenadas en `iris.testLabels` están relacionadas con las del modelo calculado, almacenadas en `iris_pred`. Para ello ejecutamos la siguiente instrucción:

```
CrossTable(x = iris.testLabels, y = iris_pred, prop.chisq=FALSE)
```

```
> CrossTable(x = iris.testLabels, y = iris_pred, prop.chisq=FALSE)

   Cell Contents
|-------------------------|
|                       N |
|           N / Row Total |
|           N / Col Total |
|         N / Table Total |
|-------------------------|

Total Observations in Table:  40

                | iris_pred
iris.testLabels |     Iris-setosa | Iris-versicolor |  Iris-virginica |       Row Total |
----------------|-----------------|-----------------|-----------------|-----------------|
    Iris-setosa |              12 |               0 |               0 |              12 |
                |           1.000 |           0.000 |           0.000 |           0.300 |
                |           1.000 |           0.000 |           0.000 |                 |
                |           0.300 |           0.000 |           0.000 |                 |
----------------|-----------------|-----------------|-----------------|-----------------|
Iris-versicolor |               0 |              12 |               0 |              12 |
                |           0.000 |           1.000 |           0.000 |           0.300 |
                |           0.000 |           0.923 |           0.000 |                 |
                |           0.000 |           0.300 |           0.000 |                 |
----------------|-----------------|-----------------|-----------------|-----------------|
 Iris-virginica |               0 |               1 |              15 |              16 |
                |           0.000 |           0.062 |           0.938 |           0.400 |
                |           0.000 |           0.077 |           1.000 |                 |
                |           0.000 |           0.025 |           0.375 |                 |
----------------|-----------------|-----------------|-----------------|-----------------|
   Column Total |              12 |              13 |              15 |              40 |
                |           0.300 |           0.325 |           0.375 |                 |
----------------|-----------------|-----------------|-----------------|-----------------|
```

De esta tabla podemos inferir el número de predicciones correctas e incorrectas. De las 40 observaciones del conjunto de datos prueba, la que ha sido etiquetada como Versicolor por el modelo y pertenece a la clase de Virginica corresponde al 1 que aparece en la fila de Iris-Virginica y en la columna Iris-versicolor. Las predicciones realizadas en todos los demás casos son correctas. Por ejemplo, se han predicho 12 Iris-setosa correctamente, lo que equivale a un 30%, 12 Iris-versicolor

(otro 30%) y 15 iris-virginica (un 37,5%). Del comportamiento del modelo podemos concluir que es correcto y que no necesitamos mejorarlo.

****

EJERCICIO 3: KNN

Uno de los usos del Machine Learning está relacionado con la detección de tumores oncológicos, de manera que, es posible construir modelos para detectar si un tumor es maligno o no o para predecir el crecimiento anormal de las células.

En el fichero **tumor.csv** se almacenan los datos de 100 pacientes con 10 variables por paciente. 9 de estas variables son numéricas (una de ellas es la variable `id` que se puede eliminar por no aportar información), y se refieren a datos medidos sobre tumores de los pacientes. Las otras 8 variables numéricas son las siguientes: `radius` (radio), `texture` (textura), `perimeter` (perímetro), `area` (área), `smoothness` (grado de suavidad), `compactness` (grado de compactibilidad), `symmetry` (grado de simetría), `fractal dimensión` (dimensión fractal). La otra variable es una variable categórica: `diagnosis_result`. Esta variable puede tomar dos valores M (maligno) o B (benigno), en función del tipo de tumor.

El objetivo de este ejercicio es, siguiendo las pautas desarrolladas en el ejemplo del algoritmo KNN, aplicar dicho algoritmo a los datos y construir un modelo que tenga la variable `diagnosis_result` como variable objetivo, esto es, que determine los resultados de la diagnosis basándose en las 8 variables numéricas mencionadas anteriormente. Para ello resolver las siguientes cuestiones:

a) Cargar los datos y normalizarlos.
b) Crear los conjuntos de datos de entrenamiento y prueba. Dividir para esto los datos en dos partes, de manera que, de las 100 observaciones, los datos del 1 al 65 los tomamos como datos de entrenamiento y los datos del 66 al 100 como datos de prueba.
c) Construir el clasificador.
d) Evaluar el modelo.

****

4.7.2. Otros algoritmos de clasificación

Algunos enlaces de otros algoritmos de clasificación utilizados y su manejo con R los podemos encontrar en:

SVM con R (http://rischanlab.github.io/SVM.html)

Naive Bayes con R (http://rischanlab.github.io/NaiveBayes.html)

Random Forest con R (http://rischanlab.github.io/RandomForest.html)

4.8. Aprendizaje no supervisado: clustering

En los sistemas de **clasificación no supervisados** no disponemos de una batería de ejemplos previamente clasificados. Lo que intentamos es obtener una **agrupación** (**clustering**) a partir de las características de los datos según su similaridad. En esta sección presentaremos dos de los algoritmos de clustering más utilizados: el k-means y el jerárquico aglomerativo.

4.8.1. Algoritmo de k-medias (k-Means)

El algoritmo de k-medias (presentado por MacQueen en 1967) es uno de los algoritmos de clustering más simples y utilizados. Tiene como objetivo la partición de un conjunto de *n* objetos en *k* clusters (*k* <*n*) (ver Figura 4.8-1)

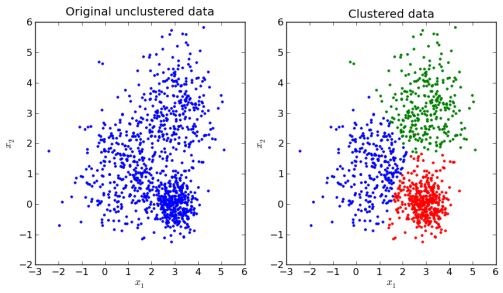


Figura 4.8-1: Resultado de un proceso de clustering utilizando k-means con *k*=3.

Al final del algoritmo, los elementos que pertenecen a un grupo deben ser similares entre ellos, y no similares a los elementos que pertenecen a otro grupo.

Podemos resumir el algoritmo de la siguiente manera:

Datos: {x_1 , …, x_n } datos.

Objetivo: particionar los datos en *k* clusters (para un *k* dado).

Algoritmo:

- Elegir *k* centroides (centros de los grupos) al azar: μ_1 , …, μ_K
- Repetir:

  1) Asignar cada dato al grupo que tiene el centroide más cercano.

  2) Recalcular el centroide de cada cluster.
- Hasta que los centroides no cambien.

Para asignar cada dato al grupo que tiene el centroide más cercano el algoritmo deberá utilizar alguna medida de distancia (para poder calcular la distancia de los datos a los centroides). Un diagrama de flujo de este algoritmo es el que se muestra en la Figura 4.8-2.

Este algoritmo es simple, eficiente y general. Pero tiene una serie de desventajas: hay que especificar el número de clusters *k*, es sensible a outliers y muy sensible a la elección de los centroides iniciales.

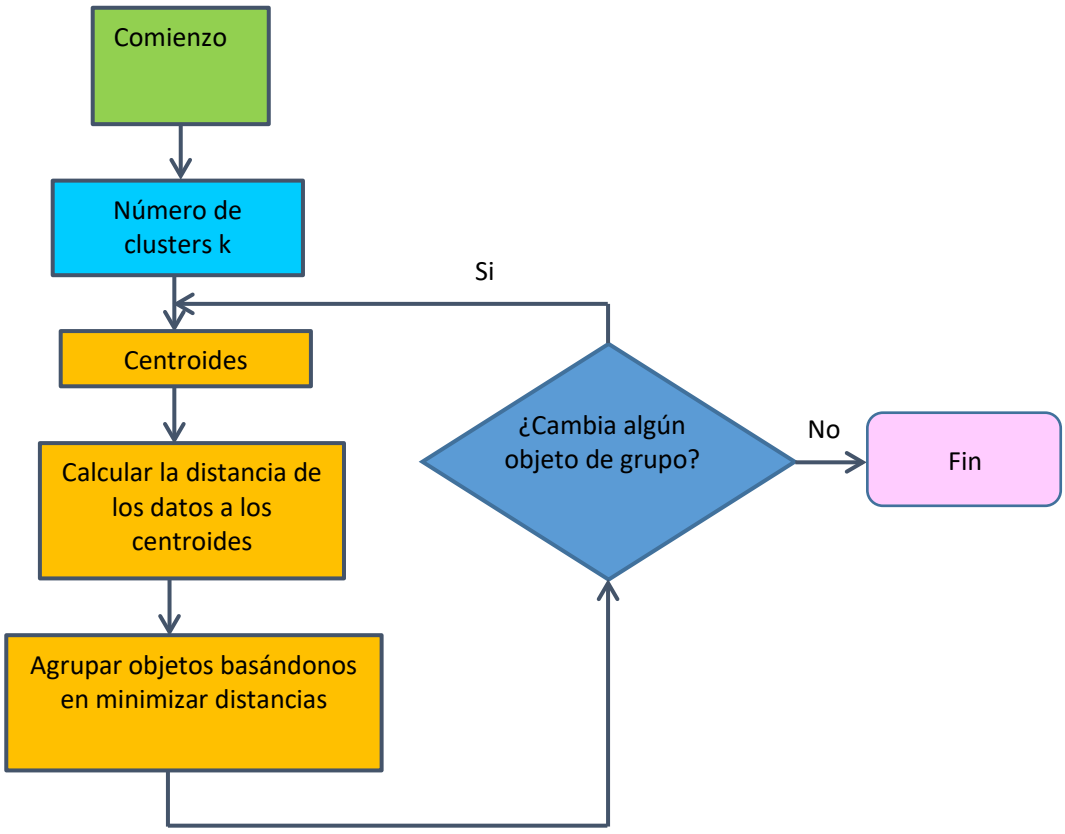


Figura 4.8-2: Diagrama de flujo del algoritmo k-means.

En la Figura 4.8-3 se muestra una imagen de unas cuantas iteraciones del algoritmo k-means.

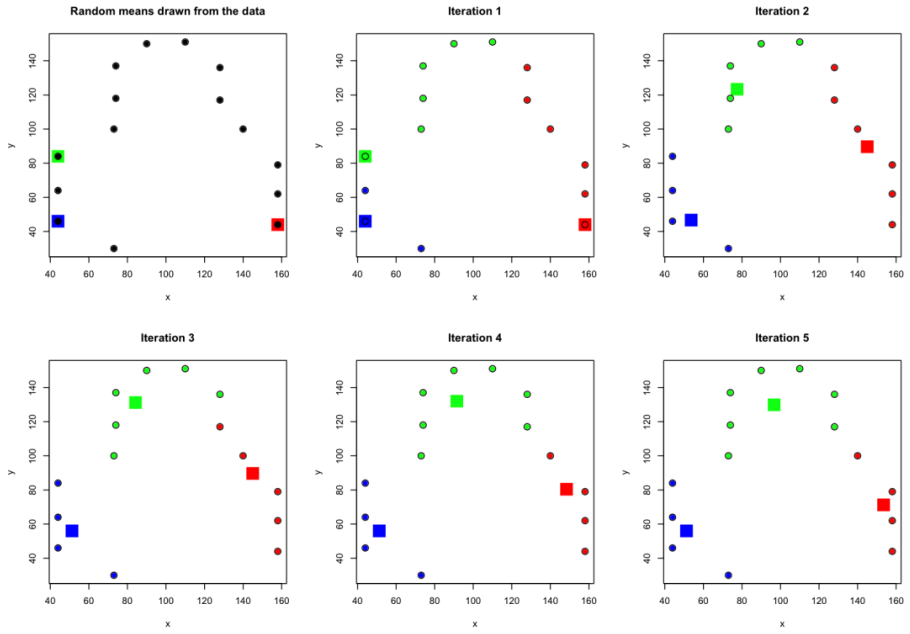


Figura 4.8-3: 5 Iteraciones del algoritmo k-means (con *k*=3). Centroides representados por cuadrados.

En el siguiente ejemplo mostramos cómo utilizar el algoritmo k-means en R.

EJEMPLO: Algoritmo k-means

El código de este ejemplo lo podemos encontrar en el script **practcluster.R**

Tenemos una serie de datos almacenados en el fichero **Libro1.txt.** Estos datos son 75 observaciones de 2 variables.

Empezamos por importar el fichero en R y visualizar los datos con el comando `plot` (ver Figura 4.8-4):

```
#Importamos los datos del fichero Libro1.txt
datoscluster <- read.delim("D:/DATOS/Libro1.txt", header=FALSE)
```

Visualizamos unos cuantos datos del fichero para ver su estructura ejecutando:

```
head(datoscluster)
  V1 V2
1  4 53
2  5 63
3 10 59
4  9 77
5 13 49
```

```
6 13 69
```

```
attach(datoscluster)

#pintamos los datos del fichero
plot(datoscluster)
```

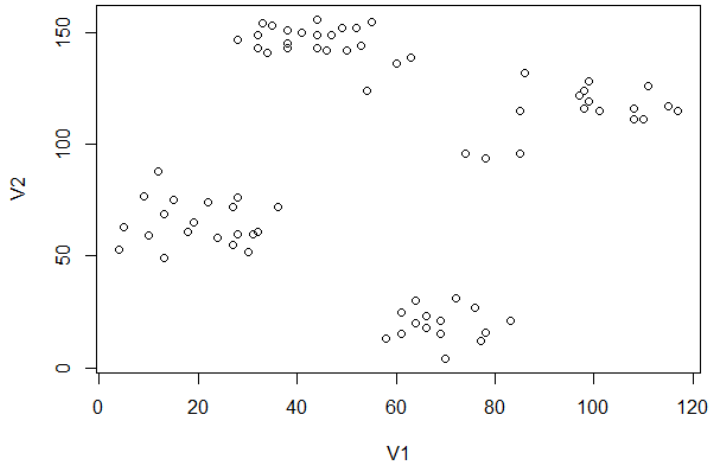


Figura 4.8-4: Representación de los datos del fichero Libro1.txt.

Los nombres de las variables del fichero los podemos obtener ejecutando:

```
names(datoscluster)
[1] "V1" "V2"
```

Una vez visualizados los datos, vamos a realizar un clustering sobre ellos con el algoritmo k-means. Como la asignación inicial de clusters es aleatoria, vamos a fijar la semilla para garantizar que podemos reproducir exactamente el ejemplo. Para ello ejecutamos la siguiente instrucción:

```
set.seed(20)
```

y aplicamos el algoritmo de k-medias sobre el conjunto de datos ejecutando el código siguiente:

```
k.means.fit <- kmeans(datoscluster,2,20) #k=2
```

donde `datoscluster` es la matriz o data frame (con columnas numéricas) que contiene los datos, 2 es el número de grupos en los que queremos particionar nuestros datos y 20 es el número máximo de iteraciones del algoritmo (por defecto este número es de 10). `k.means.fit` es el objeto en el que almacenamos todos los resultados de salida del algorimo k-means. Podemos ver todos los elementos contenidos en `k.means.fit` utilizando el comando:

```
attributes(k.means.fit)
```

Por ejemplo podemos obtener las coordenadas correspondientes a los centroides de los dos clusters mediante:

```
k.means.fit$centers
        V1        V2
1 41.05714  45.42857
2 66.97500 132.80000
```

O determinar para cada dato el cluster al que ha sido asignado mediante:

```
k.means.fit$cluster
 [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2
2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
[53] 2 2 2 2 2 2 2 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1
```

O bien el tamaño de cada cluster (número de datos que contienen) utilizando el comando:

```
> k.means.fit$size
[1] 35 40
```

Para visualizar el resultado gráficamente, representando cada cluster de un color y junto con los centroides de cada cluster (ver Figura 4.8-5) utilizamos los comandos siguientes:

```
#Dibujo los clusters, cada uno de un color
plot(datoscluster,col=k.means.fit$cluster, xlab="V1",ylab="V2")
#Centroides de los clusters
points(k.means.fit$centers, col = 1:2, pch = 8, cex=2)
```

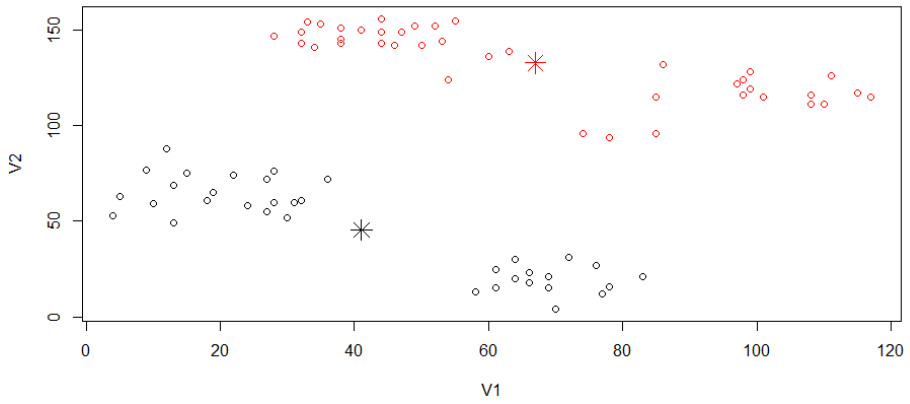


Figura 4.8-5: Representación de clusters y centroides.

La instrucción `points()` añade puntos a una gráfica existente. Admite dos formas de especificar las coordenadas de los puntos que queremos dibujar: o bien especificando un vector `x` y otro `y` que contiene las abcisas y las ordenadas de los puntos respectivamente, o bien especificando una matriz con dos columnas, la primera que contiene las abscisas mientras que la segunda las ordenadas (este

es el formato escogido para nuestra instrucción). El argumento `pch` especifica el símbolo del punto que tiene que dibujar (admite valores desde 0 hasta el 18), `cex` hace referencia al tamaño del punto que representa el centroide.

Podemos también añadir a cada punto una etiqueta con el número del cluster al que ha sido asignado (ver Figura 4.8-6) mediante:

```
text(datoscluster$V1,datoscluster$V2,labels=k.means.fit$cluster,pos=4)
```

el argumento `pos` = 4 indica que las etiquetas se deben colocar a la derecha de cada punto.

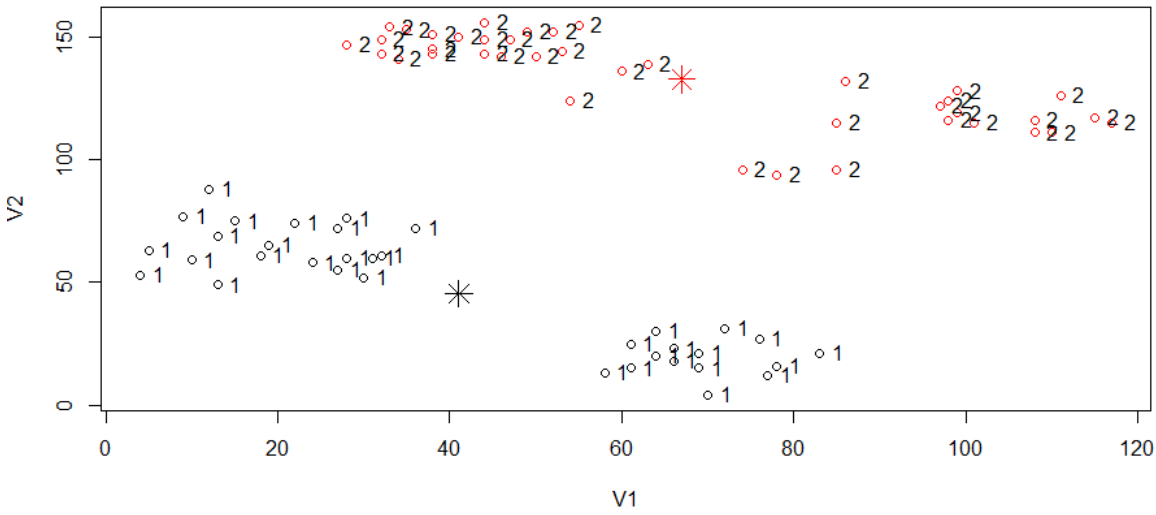


Figura 4.8-6: Representación de clusters y centroides.

En el algoritmo de k-means determinar el valor de k es una de las decisiones más importantes. Esto es, ¿cómo decidimos el número de grupos en los que vamos a particionar nuestros datos? Una de las formas de determinarlo es utilizar el denominado "criterio del codo" (*elbow method*). Este método dibuja la suma de cuadrados de la distancia entre clusters en función del número de clusters y la trata de minimizar (ver Figura 4.8-7). El código R para realizar esta gráfica es el siguiente:

```
wss <- (nrow(datoscluster)-1)*sum(apply(datoscluster,2,var))
for (i in 2:10) wss[i] <- sum(kmeans(datoscluster,centers=i)$withinss)
#10 nº máximo de clusters a analizar
plot(1:10, wss, type="b", xlab="Número de Clusters",ylab="Suma de cuad
rados dentro de los clusters",main="Cálculo del número óptimo de clust
ers con el criterio del codo")
```

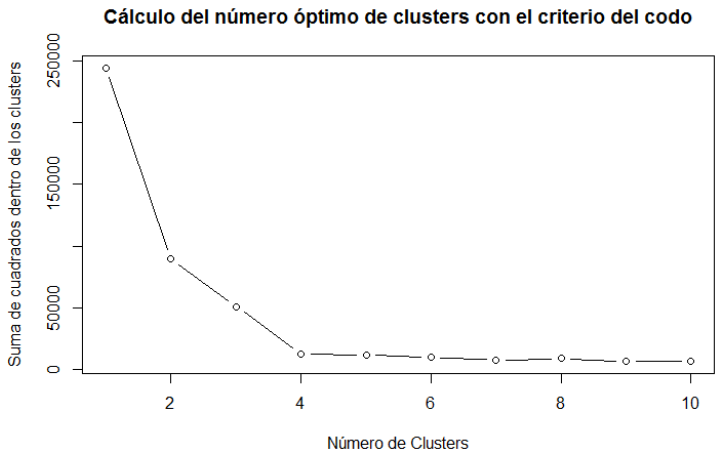


Figura 4.8-7: Criterio del codo.

En este caso, el codo se está produciendo con 4 clusters, siendo entonces k=4 el valor recomendado por este criterio.

****

EJERCICIO 4: k-means

Con el mismo fichero **Libro1.txt** que se ha utilizado en la explicación del ejemplo del algoritmo k-means:

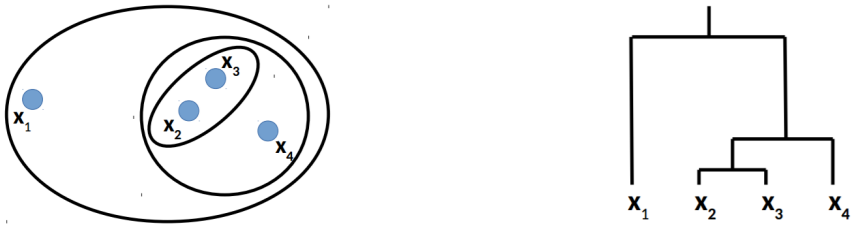
a) Realizar un clustering de los datos utilizando el algoritmo k-means para *k* = 3, 4 y 5 grupos.
b) Realizar una representación gráfica de las agrupaciones obtenidas.
c) Calcular la suma de distancias de los datos a los centroides de los clusters en cada caso.
d) Añadir al conjunto de datos una columna adicional que se llame grupo y que contenga el número del cluster al que ha sido asignado cuando *k*= 5 grupos.

****

4.8.2. Algoritmo Jerárquico Aglomerativo

En los algoritmos de clustering jerárquico los datos se agrupan de manera arborescente. Estos algoritmos pueden ser de tipo aglomerativo (*top-down*) o divisivo (*bottom-up*). En esta subsección nos centramos en explicar los de tipo aglomerativo en los que el objetivo es ir agrupando clusters. Para formar un nuevo cluster se deberá minimizar alguna distancia.

Los métodos aglomerativos, también conocidos como ascendentes, comienzan el análisis con tantos grupos como datos haya. A partir de estas unidades iniciales se van formando grupos, de forma ascendente, hasta que al final del proceso todos los datos están englobados en un mismo cluster (ver Figura 4.8-8). El árbol de clasificación que se va construyendo en los métodos jerárquicos se llama dendograma. En él se puede seguir de forma gráfica el procedimiento de creación de clusters que sigue el algoritmo. Muestran qué grupos se van uniendo y en qué nivel lo hacen.



(a) (b)

Figura 4.8-8: (a) Formación de clusters en algoritmo jerárquico aglomerativo. (b) Dendograma.

El algoritmo se puede formular de la siguiente manera:

Datos: $\{x_1, \ldots, x_n\}$ datos.

Objetivo: particionar datos en *k* clusters.
Algoritmo:

- Definir un cluster por cada dato.
- Repetir:
  Fusionar los dos clusters más cercanos.
- Hasta que quede un único cluster.

Existen muchos tipos de algoritmos aglomerativos. Casi todos tienen en común su estructura iterativa. La iteración del algoritmo requiere del cálculo de distancias entre clusters. ¿Cómo se puede definir la distancia entre clusters? Hay distintas formas de hacerlo. En la Figura 4.8-9 se muestran algunas de las posibles estrategias que se pueden emplear para unir los clusters en las diferentes etapas o niveles de un algoritmo jerárquico.

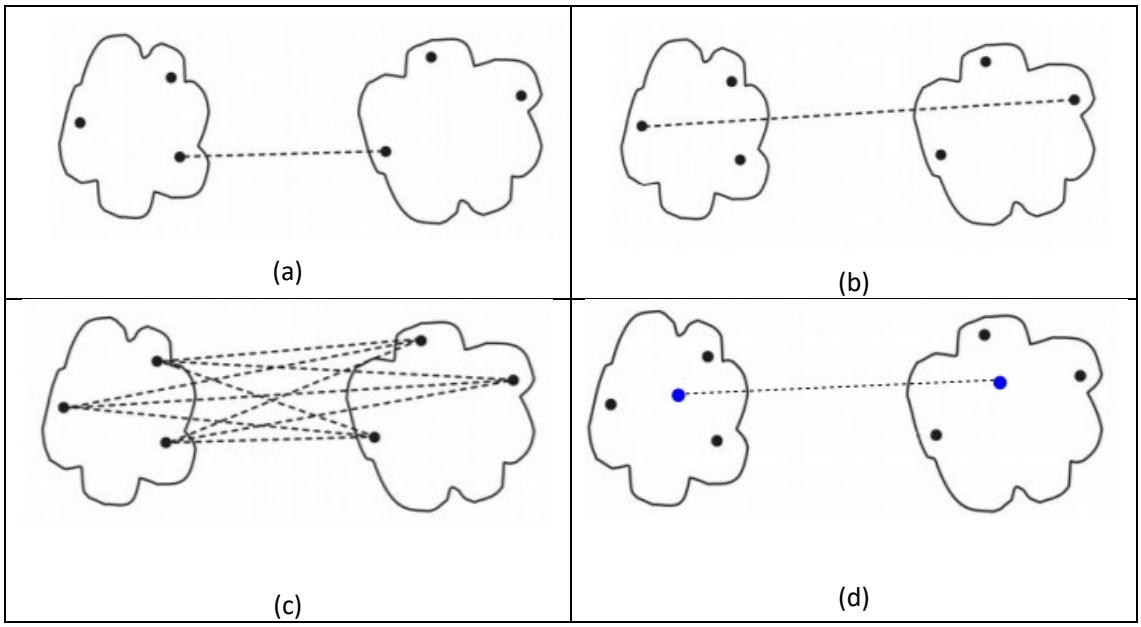


Figura 4.8-9: Distintos métodos de cálculo de distancias entre grupos: (a) MIN (single link.). (b) MAX (complete link). (c) Group average. (d) Cluster centroid.

En la Figura 4.8-9 (a) está representada la estrategia de distancia mínima o similitud máxima (single linkage). En este método se considera que la distancia o similitud entre dos clusters viene dada, respectivamente, por la mínima distancia (o máxima similitud) entre sus componentes. La Figura 4.8-9 (b) representa el método conocido como complete linkage. Se considera que la distancia o similitud entre dos clusters hay que medirla atendiendo a sus elementos más dispares, o sea, la distancia o similitud entre clusters viene dada, respectivamente, por la máxima distancia (o mínima similitud) entre sus componentes. En la estrategia Group average (Figura 4.8-9 (c)) la distancia, o similitud, entre dos clusters se obtiene como la media aritmética entre la distancia, o similitud, de las componentes de dichos clusters. En los métodos basados en los centroides (Figura 4.8-9 (d)), la distancia entre dos clusters viene dada por la distancia entre sus centroides.

Las ventajas del clustering jerárquico es que no hay que especificar de antemano *k*, el número de clusters. Además, el dendograma es útil para crear taxonomías. Entre las desventajas: No busca optimizar una función objetivo global (toma sólo decisiones locales), es caro computacionalmente y sensible a outliers.

EJEMPLO: Algoritmo jerárquico aglomerativo

El código correspondiente a este ejemplo lo tenemos disponible en el script **practcluster.R**

Vamos a ilustrar el funcionamiento del algoritmo jerárquico aglomerativo con un ejemplo sencillo, de tan solo 4 datos. Los datos son las observaciones de 2 variables (x, y):

x	y
0.3	0.6
0.35	0.4
0.7	0.8
0.8	0.5

Llamaremos a estos 4 puntos A, B, C y D respectivamente.

En primer lugar vamos a introducir los datos en un dataframe y damos nombre a las columnas de la forma siguiente:

```
x=c(0.3,0.35,0.7,0.8)
y=c(0.6,0.4,0.8,0.5)
# Formamos un vector de nombres
nombres=c("A","B","C","D")
# Juntamos las dos columnas en un data frame
xy.dat=data.frame(x,y,row.names=nombres)
```

A continuación visualizamos el conjunto con un diagrama de dispersión, añadiendo las etiquetas de los puntos (ver Figura 4.8-10). Como el dataframe `xy.dat` sólo tiene dos columnas, podemos hacerlo directamente mediante el código:

```
plot(xy.dat)
text(xy.dat,labels=nombres,pos=4)
```

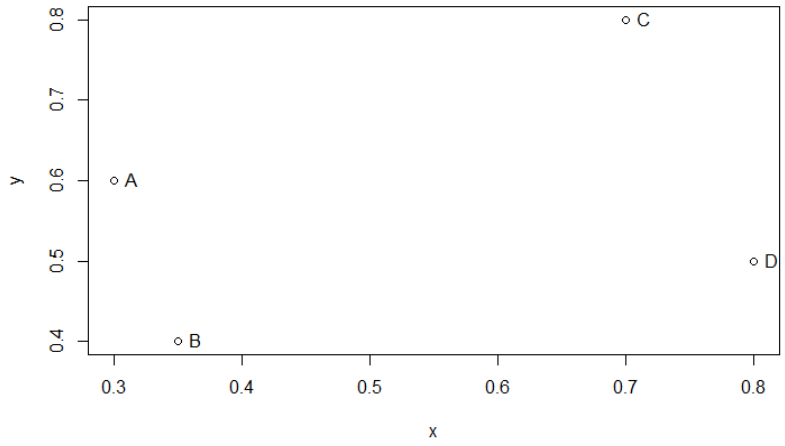


Figura 4.8-10: Representación gráfica de los datos del ejemplo de clustering jerárquico aglomerativo.

Existen muchos tipos de algoritmos aglomerativos y casi todos tienen en común su estructura iterativa. Los algoritmos suelen diferir entre ellos en cómo implementan la iteración, en concreto en: ¿cómo medimos la distancia entre dos puntos del espacio euclídeo?, y una vez que hemos definido una distancia entre dos puntos del espacio, ¿cómo calculamos la distancia entre dos grupos? (ver Figura 4.8-9).

R nos permite distintas elecciones tanto de la distancia entre dos puntos, como del método de cálculo de la distancia entre dos grupos.

La instrucción `dist` nos permite calcular todas las distancias entre los puntos (las filas) de un dataframe.

Los algoritmos de clasificación jerárquica se basan en una matriz de distancias entre los puntos. Podemos calcular esta matriz en R para nuestros datos mediante:

```
dist(xy.dat)
          A         B         C
B 0.2061553
C 0.4472136 0.5315073
D 0.5099020 0.4609772 0.3162278
```

La instrucción `dist` admite el argumento `method`, que permite especificar qué tipo de distancia entre los puntos queremos calcular, siendo la distancia euclídea la opción por defecto (más detalles con `help(dist)`).

En el caso de nuestro conjunto de datos, a partir de la salida de la función `dist` sabemos que los dos puntos más próximos son A y B, cuya distancia es 0.2062. Serán por lo tanto los dos primeros "grupos" que agrupemos en la primera etapa.

Para llevar a cabo el algoritmo jerárquico aglomerativo, R incluye la instrucción `hclust`, que, partiendo de una matriz de distancias proporcionada por el usuario junta en cada etapa los dos clusters más próximos. Se puede utilizar la función `dist` para calcular la matriz de distancias. Por defecto se usa la distancia euclídea.

En su versión más sencilla, almacenamos el resultado de `hclust` en un objeto que llamamos, por ejemplo, `xy.hcl`. Esto lo hacemos mediante la siguiente instrucción:

```
xy.hcl=hclust(dist(xy.dat))
```

`hclust` admite como argumento `method` para elegir distintos métodos de fusión entre grupos. En este caso la sintaxis de `hclust` es la siguiente:

```
hclust(d, method = "complete")
```

`method` puede tomar uno de los siguientes valores: `D.ward`: método de Ward, basado en una medida de la suma de cuadrados entre grupos; `single`: single linkage (vecino más próximo); `complete`: complete linkage (vecino más lejano), este es el método por defecto; `centroid` mide la distancia entre dos grupos cómo la distancia entre sus dos centroides; `medians` etc....

¿Qué componentes están incluidos en el objeto resultante `xy.hcl`? Podemos reconstituir el historial de agrupamientos utilizando el comando:

```
> xy.hcl$merge
     [,1] [,2]
[1,]   -1   -2
[2,]   -3   -4
[3,]    1    2
```

Cada fila representa una etapa en el proceso de agrupamiento. Cuando el elemento j de una fila, aparece con el signo negativo, indica que, en esta etapa, uno de los grupos está formado únicamente por el individuo j. Por ejemplo, en la primera etapa (primera fila), se juntan los individuos 1 y 2 (es decir A y B), y en la segunda etapa, los individuos 3 y 4 (es decir C y D). En cambio, si el elemento j aparece con signo positivo, se refiere al agrupamiento resultante de la etapa j del proceso de agrupamiento. En nuestro ejemplo, deducimos el historial siguiente:

Etapa 1: -1 -2, se agrupan el individuo 1 y el individuo 2, es decir A y B.

Etapa 2: -3 -4, se agrupan el individuo 3 y el individuo 4, es decir C y D.

Etapa 3: 1 2, se agrupan el grupo que se formó en la etapa 1 (es decir AB) con el grupo que se formó en la etapa 2 (es decir CD).

Para visualizar el resultado del proceso de agrupamiento podemos representar el dendograma (ver Figura 4.8-11). Para ello ejecutamos la instrucción:

```
plot(xy.hcl)
```

que admite los argumentos clásicos de los gráficos (en particular: `xlab`, `ylab` ...)

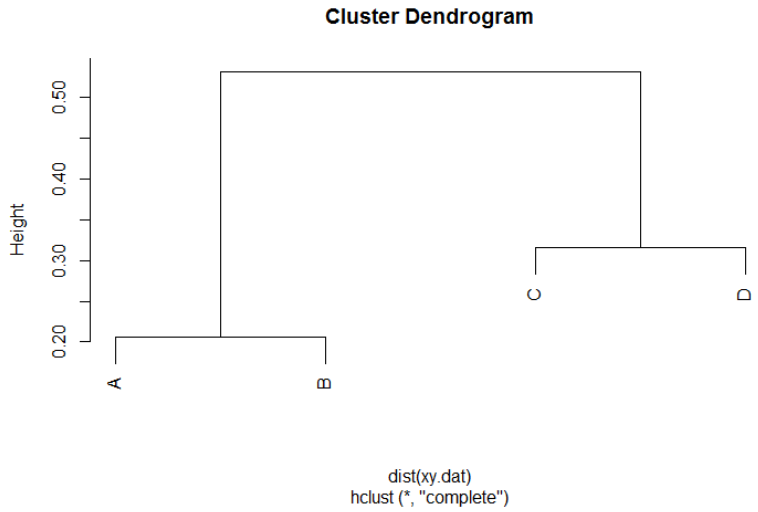


Figura 4.8-11: Dendograma.

La identificación de grupos la podemos hacer visualmente en el dendograma. Esta identificación se puede facilitar con la instrucción `rect.hclust`, que admite como argumento un objeto resultado de `hclust`, y el número de grupos `k` que queremos identificar. Por ejemplo, si queremos identificar 2 grupos escribimos:

```
rect.hclust(xy.hcl,k=2)
```

y obtenemos el dendograma con dos agrupaciones. Los cortes de las agrupaciones obtenidas se muestran en la Figura 4.8-12.

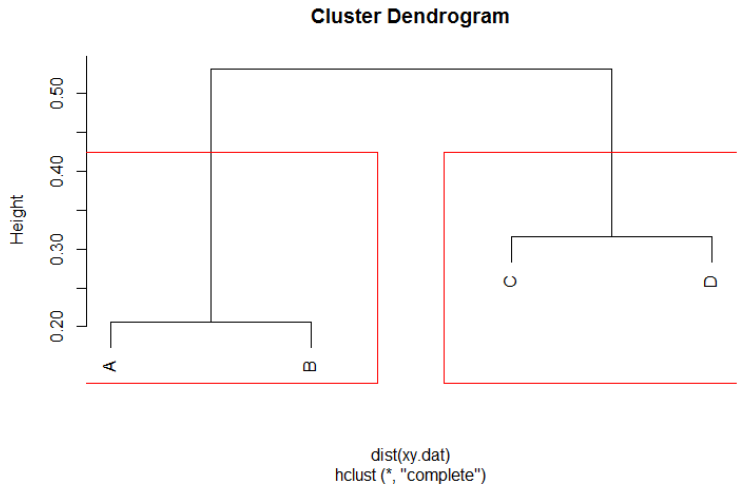


Figura 4.8-12: Identificación de 2 grupos.

En este ejemplo tan sencillo, la instrucción `rect.hclust` no ayuda mucho. Podemos también obtener con `rect.hclust` directamente la lista de los clusters con sus miembros. En este caso, si queremos obtener por separado los dos grupos generados ejecutamos:

```
print(rect.hclust(xy.hcl,k=2))
[[1]]
A B
1 2

[[2]]
C D
3 4
```

Por otra parte, si decidimos particionar en un determinado número de grupos, podemos obtener para cada dato el número del cluster en el que está asignado. Supongamos por ejemplo que queremos particionar el conjunto en tres grupos, bastaría con ejecutar la instrucción `cutree` de esta forma:

```
cutree(xy.hcl,k=3)
A B C D
1 1 2 3
```

Podemos obtener las asignaciones de distintas particiones si, en lugar de especificar un número para `k`, especificamos un vector, por ejemplo:

```
> cutree(xy.hcl,k=2:3)

  2 3
A 1 1
B 1 1
C 2 2
D 2 3
```

En caso de particionar en 2 grupos A y B van al primer grupo y C y D al segundo. Si particionamos en 3 grupos A y B van al primer grupo, C al segundo y D al tercero.

Por ejemplo, si queremos añadir al conjunto `xy.dat` tres columnas que contengan las asignaciones de los individuos en caso de que decidamos una partición en 2, 3 y 4 grupos respectivamente ejecutaríamos:

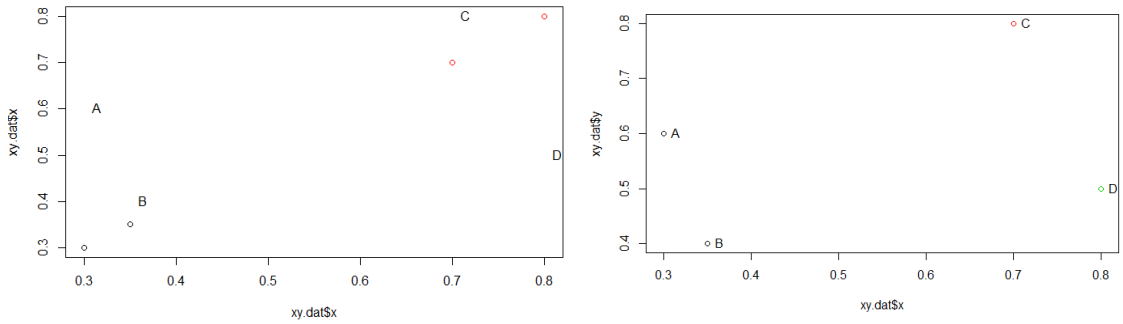
```
xy.dat=data.frame(xy.dat,cutree(xy.hcl,k=2:4))
```

Con lo que `xy.data` contiene ahora:

```
> xy.dat
     x   y X2 X3 X4
A 0.30 0.6  1  1  1
B 0.35 0.4  1  1  2
C 0.70 0.8  2  2  3
D 0.80 0.5  2  3  4
```

Si queremos dibujarlos los datos de los clusters mostrando cada una de las agrupaciones obtenidas de un color para `k`= 2 y 3 (ver Figura 4.8-13) podemos ejecutar el siguiente código.

```
#Dibujo los clusters, cada uno de un color
#para k=2
plot(xy.dat$x,xy.dat$x,col=xy.dat[1:4,3])
text(xy.dat,labels=nombres,pos=4)
#para k=3
plot(xy.dat$x,xy.dat$y,col=xy.dat[1:4,4])
text(xy.dat,labels=nombres,pos=4)
```



(a) (b)

Figura 4.8-13: Clusters obtenidos por agrupamiento aglomerativo jerárquico (a) k= 2 y (b) k =3.

****

EJERCICIO 5: Algoritmo Jerárquico Aglomerativo

En el fichero **proteinas.txt** están los datos del año 1973 correspondientes al consumo de proteínas en 25 países europeos correspondientes a nueve grupos de alimentos. Los nombres de las variables son los siguientes:

1. Country: País
2. RdMeat: Carne roja
3. WhMeat: Carne Blanca
4. Eggs: Huevos
5. Milk: Leche
6. Fish: Pescado
7. Cereal: Cereales
8. Starch: Feculentos
9. Nuts: Frutos secos, y aceites
10. FrVeg: Frutas y verduras

a) Realizar una clasificación jerárquica de los países en base a su consumo de proteínas según las distintas fuentes de alimentación. Utilizar el método de Ward (`D.Ward`), y especificar que los casos se etiqueten con la variable `Country`.

   Nota: habría que trabajar con los datos tipificados (escalados) para evitar que unas pocas variables dominen el análisis. Para tipificar el dataframe `proteinas.dat` y almacenar el resultado en el dataframe `proteinas.tipif` podemos utilizar la instrucción `proteinas.tip=scale(proteinas.dat)`

b) Contestar, examinando el historial de agrupamientos, a las siguientes preguntas: ¿qué dos países se combinan primero? ¿En qué consiste la segunda etapa? ¿Cuándo es la primera vez que se forma un agrupamiento con más de dos países?
c) Examinar el dendograma: si queremos quedarnos con tres grupos, realizar la lista de los países que pertenecen a cada grupo. ¿Y con 4 grupos?
d) Realizar el análisis en componentes principales. Guardar las puntuaciones de los países según la primera componente principal. Ordenar los países por orden creciente de estas puntuaciones. ¿El orden obtenido parece guardar relación con los grupos obtenidos en el apartado anterior? ¿Cómo podemos explicar esta relación?

****