Capítulo 1

Aprendizaje Automático

1.1. Supervisado

En el *aprendizaje supervisado*, un supervisor lleva a cabo algunas etapas en la construcción del modelo: determinación de las clases, elección y prueba de las características discriminantes, selección de la muestra, cálculo de funciones discriminantes y evaluación del resultado. Una de las primeras tareas en el aprendizaje supervisado es obtener el valor de una variable dependiente *continua* a partir de un vector de variables independientes.

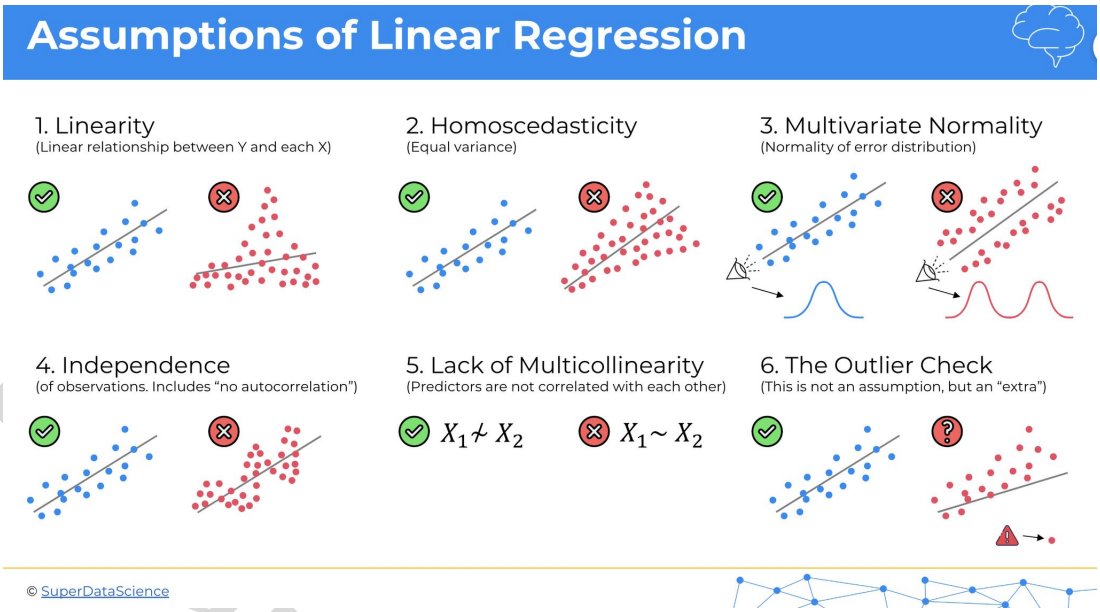
1.1.1. Regresión lineal

La regresión lineal es un área bien desarrollada y conocida de la estadística; la ubicuidad de sus métodos en el análisis de datos surge de la facilidad con la que se pueden ajustar modelos para describir las características principales de un proceso o una población. Además de ser útil para la descripción, también es muy útil para la predicción ya que los modelos obtenidos en general proveen buenas aproximaciones de relaciones complejas. Analizaremos dichos métodos para entender el potencial de éxito y falla que tienen; sin emabargo, el enfoque apunta a entender los aspectos esenciales y más útiles para la ciencia de datos: **los modelos ajustados**.

Introducción

El contenido de esta sección es aplicable a una amplia variedad de problemas de análisis de datos; estos problemas comparten una característica común: Una de las variables tiene mayor importancia y el objetivo es comprender mejor el proceso que la generó, además de poder predecir valores futuros de dicha variable. Dado que una parte importante de la ciencia de datos incluye análisis predictivos, entender regresión lineal es un componente importante para el análisis de datos. Debe recordarse que la regresión lineal puede resultar muy exitosa tanto para aprender sobre el origen de los datos como para predecir valores futuros y desconocidos de un proceso.

Es importante recordar los supuestos de la regresión lineal:



https://bit.ly/3YcxNe8

1.1.1.1. El modelo de regresión lineal

En el marco de la regresión lineal, los datos pueden verse como n parejas: $D = \{(y_1, \mathbf{x}_1), \ldots, (y_n, \mathbf{x}_n)\}$. La visión estadística supone que $y_i, i = 1, 2, \ldots, n$ provienen de una variable aleatoria, identificada como la variable de respuesta Y_i y que el vector explicativo que la acompaña $\mathbf{x}_i$ no es aleatorio y puede usarse para explicar y_i. En ocasiones, el objetivo estadístico es predecir un valor no observado y_0 a partir de un vector observado $\mathbf{x}_0$ usando una función de predicción obtenida a partir de los datos. En estadística se predicen valores de una variable aleatoria y se estiman los parámetros que describen una distribución de la misma.

Las variables $\mathbf{x}_i$, y_i son intrínsecamente diferentes; el objetivo es describir o modelar el valor esperado de la variable Y_i mientras que $\mathbf{x}_i$ es un vector de interés secundario. Comúnmente se asume que el vector explicativo no tiene una distribución, en cambio contiene valores fijos medidos sin error o en control absoluto del investigador. Esta suposición puede resultar engañosa y la regresión lineal es un método valioso para obtener información sobre el proceso o la población de la que provienen los datos. El vector $\mathbf{x}$ se conoce como *explicativo* o *predictor*, dependiendo del objetivo particular del análisis; por simplicidad se utilizará el término *vector predictor* sin importar el tipo de análisis que se esté realizando.

El modelo de regresión lineal especifica que el valor esperado de Y_i es una función lineal de $\mathbf{x}_i$ dado por:

$$E\left(Y_i|x_i\right) = \beta_0 + \beta_1 x_{i,1} + \ldots + \beta_p x_{i,p}$$

O en su versión vectorial:

$$E\left(Y_i|x_i\right) = \mathbf{x}_i^T \boldsymbol{\beta}$$

donde $\boldsymbol{\beta} = [\beta_0, \beta_1, \ldots, \beta_p]^T$ es un vector de constantes desconocidas. El vector $\mathbf{x}_i = [1, x_{i,1}, \ldots, x_{i,p}]$ contiene las observaciones de los p predictores variables; la longitud de los vectores $\boldsymbol{\beta}$ y $\mathbf{x}_i$ es $q = p + 1$.

La aplicación de la predicción es simple en el sentido de que los detalles del modelo predictivo no tienen especial importancia si produce predicciones precisas de Y; es más común que se tenga interés en entender la influencia de cada variable predictora sobre Y, más precisamente sobre el valor esperado de Y. El modelo de Y_i como función lineal de $\mathbf{x}_i$ puede expandirse para especificar la distribución de Y_i.

Además, si la distribución satisface ciertas condiciones (como normalidad), se pueden realizar un conjunto extendido de inferencias. Estas inferencias adicionales dependen de condiciones llamadas *modelo de inferencia*, el cual establece que:

$$\begin{aligned} Y &= E(Y|\mathbf{x}) + \epsilon \\ E(Y|\mathbf{x}) &= \mathbf{x}^T\boldsymbol{\beta} \\ \epsilon &\sim N\left(0, \sigma_\epsilon^2\right) \end{aligned}$$

La tercera condición especifica que la variable aleatoria *residual* ϵ se distribuye como una normal con esperanza 0 y varianza σ_ϵ^2. La varianza de ϵ es independiente de la varianza de x y es la misma para cada valor de $E(Y|\mathbf{x})$. La varianza constante implica que las diferencias entre los valores reales de Y y las estimaciones de $E(Y|\mathbf{x})$ deberían ser aproximadamente los mismos.

1.1.1.2. Mínimos cuadrados

Explicación en Prometeo: https://bit.ly/3AYiwPN.

La primera tarea computacional del análisis de regresión lineal es calcular un estimado del vector de parámetros $\boldsymbol{\beta}$. En estadística y ciencia de datos, el estimador de mínimos cuadrados casi siempre es la primera elección porque es fácil de entender y calcular. Aún más, su precisión compite con la de una gran variedad de métodos más complejos; los intervalos de confianza y las pruebas de hipótesis con respecto a los parámetros $\beta_0, \beta_1, \ldots, \beta_p$ son sencillos.

El objetivo de los mínimos cuadrados es minimizar la suma de los residuos al cuadrado; los residuos son las diferencias entre el valor observado y_i y los valores ajustados $\hat{y}_i = \mathbf{x}_i^T\hat{\boldsymbol{\beta}}, i = 1, \ldots, n$, donde $\hat{\boldsymbol{\beta}}$ es un vector de números reales de longitud q. La suma de los residuos al cuadrado se calcula con:

$$\begin{aligned} S\left(\hat{\boldsymbol{\beta}}\right) &= \sum_{i=1}^n \left(y_i - \hat{y}_i\right)^2 \\ &= \sum_{i=1}^n \left(y_i - \mathbf{x}_i^T\boldsymbol{\beta}\right)^2 \end{aligned}$$

Se minimiza con la elección de $\boldsymbol{\beta}$; al tener valor desconocido necesitamos calcularlo para poder continuar. El vector $\hat{\boldsymbol{\beta}}$ que minimiza $S(\cdot)$ se conoce como el estimador de mínimos cuadrados y, por definición, cuaquier otro vector entrega una suma de errores al cuadrado al menos tan grande como este estimador. Una derivación puede consultarse en https://bit.ly/2KfhAyl. Lo importante desde nuestra perspectiva es que el estimador $\boldsymbol{\beta}$ es la solución de las *ecuaciones normales*:

$$\mathbf{X}^T\mathbf{X}\boldsymbol{\beta} = \mathbf{X}^T\mathbf{y}$$

$\underset{n\times q}{\mathbf{X}}$ se forma al *apilar* los vectores predictivos $\mathbf{x}_1^T, \ldots, \mathbf{x}_n^T$. El vector $\mathbf{y} = [y_1, \ldots, y_n]^T$ consiste de n productos de las variables aleatorias $Y_1, \ldots, Y_n$. Si $\mathbf{X}^T\mathbf{X}$ es invertible, entonces la solución de la ecuación normal es:

$$\hat{\boldsymbol{\beta}} = \left(\mathbf{X}^T\mathbf{X}\right)^{-1}\mathbf{X}^T\mathbf{y}$$

Si el modelo inferencial anterior describe aproximadamante bien la relación entre $E(Y)$, $\mathbf{x}$ y la distribución de ϵ, entonces la varianza de Y con respecto a su esperanza condicional $E(Y|\mathbf{x}) = \mathbf{x}^T\boldsymbol{\beta}$ se estima como:

$$\sigma_\epsilon^2 = \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{n-q}$$

donde $\hat{y} = \mathbf{x}_i^T\hat{\boldsymbol{\beta}}$. La varianza del estimador $\hat{\boldsymbol{\beta}}$ es la matriz de $q \times q$:

$$var\left(\hat{\boldsymbol{\beta}}\right) = \sigma_\epsilon^2\left(\mathbf{X}^T\mathbf{X}\right)^{-1}$$

La diagonal de los elementos de $var\left(\hat{\boldsymbol{\beta}}\right)$ son las varianzas de los estimadores individuales $\hat{\beta}_0, \ldots, \hat{\beta}_p$, mientras que los elementos fuera de ella son las covarianzas entre los estimadores. El estimador usual de la matriz de varianzas es $\hat{var}\left(\hat{\boldsymbol{\beta}}\right) = \hat{\sigma}_\epsilon^2\left(\mathbf{X}^T\mathbf{X}\right)^{-1}$. La varianza estimada para un estimador particular $\hat{\beta}_i$ se extrae de la diagonal de $\hat{var}\left(\hat{\boldsymbol{\beta}}\right)$. La raíz cuadrada de la varianza se conoce como el error estándar de $\hat{\beta}_i$:

$$\hat{\sigma}\left(\hat{\boldsymbol{\beta}}_i\right) = \sqrt{\hat{\sigma}^2\left(\hat{\boldsymbol{\beta}}_i\right)}$$

Los errores estándar se utilizan de forma extensiva para construir intervalos de confianza y para pruebas de hipótesis. Hay que recordar que la propiedad de estos estimadores depende de qué tan real es el modelo inferencial.

Las estimaciones *puntuales* $\hat{\beta}_i, i = 0, 1, \ldots, p$, pueden interpretarse como la mejor estimación de β_i; a pesar de ser la mejor, debe tenerse en cuenta que casi con seguridad no será exactamente igual a β_i. En general existirán otros valores además de la estimación puntual cercanos al punto a estimar que son consistentes con los datos, en el sentido que respaldan la posibilidad de que cada uno podría ser el valor de β_i. Comúnmente se desea presentar un intervalo de valores consistentes además de la estimación puntual: un intervalo de confianza.

Ejemplo: Depresión

Ilustraremos una situación en la que el modelo de inferencia lineal es útil con ayuda del conjunto de datos *Ginzberg*. Para el tratamiento de la depresión (enfermedad mental que puede llegar a ser incapacitante), las causas deben ser entendidas; se ha postulado que está asociada a un par de condiciones menos complejas: *fatalismo* y *simplicidad*. El fatalismo se refiere a la imposiblidad de controlar el mundo en el que vive la persona mientras que la medida en la que una persona observa eventos y circunstancias como positivas o negativas se describe con el término simplicidad. El grado en el que estas variables se asocian con la depresión puede iluminar el valor del tratamiento de la depresión al tomar en cuenta las percepciones de la persona. Medir cuantitativamente la relación entre fatalismo y simplicidad con la depresión puede ser un paso muy importante para comprender las relaciones entre estas condiciones.

El conjunto contiene 82 observaciones realizadas a pacientes hospitalizados por depresión; se calificaron en cuanto a la severidad de la depresión, simplicidad y fatalismo, las variables se escalaron de forma que los valores grandes reflejen manifestaciones mas fuertes de simplicidad

y fatalismo, además de una depresión más severa. La intensidad de la asociación lineal entre fatalismo y depresión, medida con el coeficiente de correlación de Pearson[1] es $r = 0.657$ y la correlación entre simplicidad y depresión es $r = 0.643$; estos coeficientes indican un grado de asociación lineal moderadamente positivo entre la depresión y cada variable, además muestra cierta certeza en la existencia de una relación conjunta entre depresión, fatalismo y simplicidad. La tarea es cuantificar la intensidad de la asociación conjunta y describir la relación: producir una aproximación objetiva y manejable de una relación complicada y cambiante a nivel de cada paciente.

Consideremos el modelo lineal:

$$E\left(Y|x\right) = \beta_0 + \beta_1 x_{fatalism} + \beta_2 x_{simplicity} = \mathbf{x}^T\boldsymbol{\beta}$$

donde Y es el nivel de depresión y $\mathbf{x} = [1, x_{fatalism}, x_{simplicity}]^T$. En realidad este modelo es sólo una aproximación a la relación real, teniéndolo en cuenta, se realiza el procesamiento con la esperanza de obtener información útil.

Los parámetros obtenidos de la estimación se observan en la tabla:

Variable	Parámetro estimado	Error estándar	Intervalo de confianza 95 % Límite inferior	Límite superior
Constante	0.2027	0.0947	0.0133	0.3921
Fatalism	0.4178	0.1006	0.2166	0.6190
Simplicity	0.3795	0.1006	0.1783	0.5807

Cuadro 1.1: Estimación de los parámetros, error estándar e intervalos de confianza para el modelo

Podemos interpretar matemáticamente los parámetros estimados; dado que los valores de fatalismo y simplicidad se escalaron para tener una media común de valor 1 y desviación estándar de 0.5, el parámetro estimado se puede comparar directamente. En particular, un incremento de una unidad en fatalismo con simplicidad constante induce un incremento estimado de 0.4178 en la escala de depresión, y se estima que una unidad incrementada en la simplicidad con fatalismo constante incrementa el nivel de depresión en 0.3795 unidades; por tanto, se puede argumentar que fatalismo y simplicidad son casi equivalentes para determinar el nivel de depresión, pero que el fatalismo es más importante. Por supuesto que el contexto es muy artificial: es prácticamente imposible que se pueda incrementar el valor de fatalismo o simplicidad (o cualquier otra actitud, creencia, etc.) manteniendo la otra fija; aún más, fatalismo y simplicidad está correlacionadas ($r = 0.631$). En general, los individuos con valores mayores de fatalismo tendrán también valores mayores en simplicidad. Posteriormente se explicará el significado de los intervalos de confianza mostrados en la tabla.

La figura siguiente muestra los datos y el plano generado por el modelo $\hat{y} = 0.2027 + 0.4178x_{fatalism} + 0.3795x_{simplicity}$.

```
1  install.packages('plot3D')
2  install.packages("car")
3  install.packages("rgl")
```

[1]Mide la relación estadística (asociación) entre dos variables continuas, está basado en el método de la covarianza.

```r
library("car") # Ginzberg dataset
library("rgl")
options(rgl.printRglwidget = TRUE)
plot3d(depression ~ fatalism + simplicity, Ginzberg, xlim= c(0,3), zlim=c(0,3))
n <- 20
x <- y <- seq(0, 3, length = n)
region <- expand.grid(x = x, y = y)
z <- matrix(0.2027+0.4178*region$x + 0.3795*region$y, n, n)
surface3d(x, y, z, back = 'line', front = 'line',
          col = 'red', lwd = 1.5, alpha = 0.4)
```

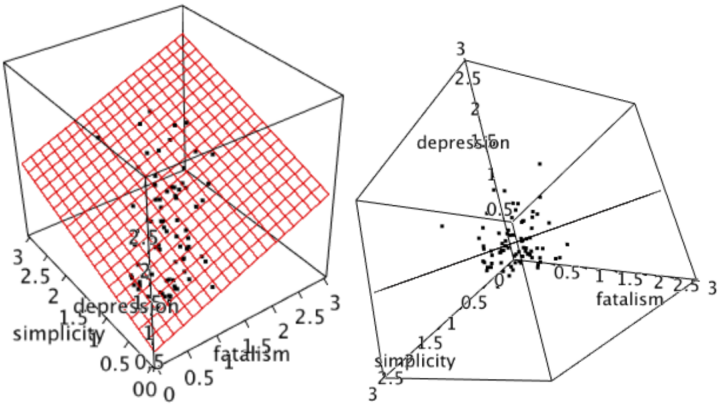


Figura 1.1: Plano del modelo de depresión ajustado usando *rgl*

```r
library("car") # Ginzberg dataset
library("plot3D")
# datos
x <- Ginzberg$fatalism
y <- Ginzberg$simplicity
z <- Ginzberg$depression
# lm para calcular el modelo de regresión lineal ax + by + cz + d = 0
# modelo ajustado (z = ax + by + d)
fit <- lm( z ~ x + y )
# predecir los valores  sobre una malla regular xy
grid_lines = 26
x_pred <- seq(min(x), max(x), length.out = grid_lines)
y_pred <- seq(min(y), max(y), length.out = grid_lines)
xy <- expand.grid( x = x_pred, y = y_pred)
z_pred <- matrix(predict(fit, newdata = xy),
                 nrow = grid_lines, ncol = grid_lines)
# puntos ajustados para las líneas sobre la superficie
fitpoints <- predict(fit)
# scatter con plano de regresión
scatter3D(x, y, z, pch = 18, cex = 2,
          theta = 20, phi = 20, ticktype = "detailed",
          xlab = "fatalism", ylab = "simplicity", zlab = "depression",
          surf = list(x = x_pred, y = y_pred, z = z_pred,
                      facets = NA, fit = fitpoints), main = "Ginzberg")
```

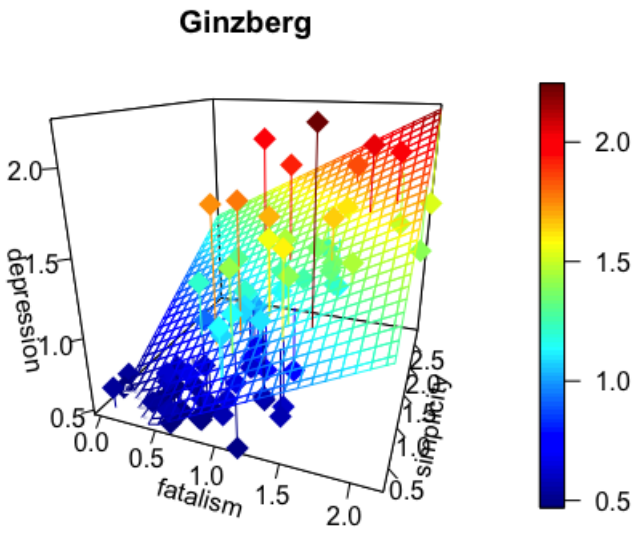


Figura 1.2: Plano del modelo de depresión ajustado usando *plot3D*

La aseveración sobre el cambio en la variable de respuesta (depresión) como una función de las variables explicativas (fatalismo y simplicidad), es un ejemplo de inferencia estadística; esta afirmación está basada en una muestra de datos, pero se puede generalizar a una población mayor de individuos. Pero debe tenerse cuidado, ya que la inferencia podría estar equivocada (incluso bastante), hasta el extremo de *no* existir relación entre las variables.

La posibilidad de una descripción incorrecta nace de que las estimaciones fueron calculadas a partir de una muestra, además de que el modelo propuesto es sólo una aproximación. El nivel del error para cada estimación particular se cuantifica con ayuda del error estándar; la cual puede ser interpretada como la diferencia absoluta promedio entre un parámetro verdadero (desconocido) y una estimación del mismo, determinada con una muestra de observaciones independientes, extraídas de la población o el proceso. La tabla 1.1 incluye los errores estandar $\hat{\sigma}_1\left(\hat{\beta}_1\right)$ y $\hat{\sigma}_2\left(\hat{\beta}_2\right)$ asociados con cada uno de los parámetros; en general, el error estándar no proporciona información útil. En cambio, los intervalos de confianza proveen una interpretación alternativa y muchas veces un mejor enfoque cuando se trata con imprecisiones en estimaciones basadas en muestras.

1.1.1.3. Intervalos de confianza

Un intervalo de confianza para un parámetro es un conjunto de valores posibles para el parámetro que son consistentes con los datos; por ejemplo, la tabla 1.1 presenta un intervalo de confianza de 95 %; para el caso de $\beta_{fatalism}$ se calcula como:

$$\left[\hat{\beta}_{fatalism} - 2\hat{\sigma}\left(\hat{\beta}_{fatalism}\right), \hat{\beta}_{fatalism} + 2\hat{\sigma}\left(\hat{\beta}_{fatalism}\right)\right] = [0.2166, 0.6190]$$

Esto significa que tenemos 95 % de certeza en que el valor de $\beta_{fatalism}$ está entre 0.2166 y 0.6190; todos los valores fuera de este intervalo se consideran como inconsistentes con lo datos y, por tanto, que no confiamos en que el valor verdadero este fuera de este intervalo. El término $2\hat{\sigma}\left(\hat{\beta}\right)$ se conoce como el *error marginal*; el ancho del intervalo es la diferencia entre el límite

mayor y el inferior, es decir, aproximadamente $4\hat{\sigma}\left(\hat{\beta}\right)$ o dos veces el error marginal. Se prefiere que los intervalos de confianza sean estrechos que amplios y si su amplitud es muy pequeña, se dice que es óptimo, dado que al 95 % de confianza el valor verdadero es esencialmente el estimado.

El *nivel de confianza* es la probabilidad de que el procedimiento entregue un intervalo de confianza que incluye al parámetro; matemáticamente el nivel de confianza es $1-\alpha$ con $0 < \alpha < 1$, valores comúnmente usados de α son 0.01, 0.05 y 0.1. El nivel de confianza puede interpretarse de la siguiente forma: si la recolección de la muestra y el cálculo del intervalo de confianza se repitiera muchas veces, entonces $100\,(1-\alpha)$ % de todos los intervalos de confianza contendrán el valor del parámetro (suponiendo que el modelo inferencial es correcto). Debe recordarse que el parámetro se encuentra dentro de un intervalo $[0.2166, 0.6190]$ ó no, pero es imposible determinar cuál de los dos se cumple. Por tanto, no es correcto decir que hay una probabilidad de 0.95 de encontrar a $\beta_{fatalism}$ está entre 0.2166 y 0.6190; la interpretación correcta es que la probabilidad de que el procedimiento genere un intervalo que contenga el valor real del parámetro es $1-\alpha$; por este motivo, sólo se puede establecer que estamos $100\,(1-\alpha)$ % seguro de que el parámetro está entre 0.2166 y 0.6190.

Que el intervalo de confianza contenga al parámetro con probabilidad $1-\alpha$ depende de dos eventos:

1. Que el intervalo esté centrado en el verdadero valor de β, que, a su vez depende de la correctitud del modelo lineal adoptado; si la relación entre $E\,(Y|x)$ y x no es aproximadamente lineal, el intervalo puede no estar centrado en el valor real. La exactitud de una aproximación lineal para una relación no lineal no depende del tamaño de las muestras, es decir, las fallas de un mal modelo no pueden evitarse incrementando el número de observaciones.

2. El error estandar estimado $\hat{\sigma}\left(\hat{\beta}_i\right)$ debe ser una estimación muy precisa de la variabilidad real en cada una de las estimaciones del parámetro en cada muestra y, la precisión de $\hat{\sigma}\left(\hat{\beta}_i\right)$ depende de varias condiciones de la distribución que describe a la población o muestra.

En el tratamiento de los intervalos de confianza hasta este momento se ha supuesto que el tamaño de la muestra n es grande; cuando n es pequeño (menor a 100), se debe realizar un ajuste para corregir la falta de precisión derivada de este tamaño de muestra. Para valores de $n-q < 80$, un intervalo de confianza $100\,(1-\alpha)$ % más preciso para β es:

$$\hat{\beta} \pm t^*_{n-q,\alpha/2}\hat{\sigma}\left(\hat{\beta}\right)$$

donde $t^*_{n-q,\alpha/2}$ es el percentil $100\alpha/2$ de la distribución-$\mathcal{T}$ con $n-q$ grados de libertad. La distribución-$\mathcal{T}$ es muy similar a la distribución normal, pero tiene colas alargadas cuando $n-q < 80$. En otro caso, se puede calcular un intervalo de confianza para β con $\hat{\beta} \pm \hat{\sigma}\left(\hat{\beta}\right)$.

1.1.1.4. Condiciones de la distribución

El nivel de confianza otorgado a un intervalo de confianza sólo es aproximado, dado que su precisión depende del cumplimiento de un conjunto de condiciones que en la realidad no pueden lograrse con total exactitud; si las condiciones son cumplidas *aproximadamente*, entonces el nivel de confianza está cerca de ser verdadero. Comúnmente estas condiciones son conocidas como *suposiciones*, término a veces malinterpretado y que indica que el analista sólo necesita

suponer o *fingir* que dichas condiciones se cumplen para el problema y los datos con los que está tratando. Sin embargo, las condiciones deben ser cumplidas al menos de forma aproximada para obtener resultados confiables. Hay tres condiciones que describen a la distribución de la variable de respuesta y la cuarta describe las observaciones:

1. *Linealidad*: la relación entre la esperanza de Y y x es lineal, en otras palabras, el modelo $E(Y|\mathbf{x}) = \mathbf{x}^T\boldsymbol{\beta}$ es correcto.
2. *Varianza constante*: para todo valor de $\mathbf{x}$, la distribución de Y con respecto a $E(Y|\mathbf{x})$ tiene la misma varianza σ_ϵ^2.
3. *Normalidad*: para cada valor de $\mathbf{x}$, la distribución de Y es normal; de forma equivalente, $\epsilon \sim N\left(0, \sigma_\epsilon^2\right)$.
4. *Independencia*: Las observaciones son instancias de n variables aleatorias independientes. Específicamente, si y_i es una instancia de una variable aleatoria Y_i, para $i = 1, \ldots, n$, entonces $Y_1, \ldots, Y_n$ son variables aleatorias independientes.

Las condiciones 1 y 2 especifican la media y la varianza de Y, por lo tanto la condición de distribución normal puede combinarse con las condiciones de media y varianza: $Y \sim N\left(E(Y|\mathbf{x}), \sigma_\epsilon^2\right)$ para todo valor de $\mathbf{x}$.

Combinando todas las condiciones se llega a la condición de que para Y_i, para $i = 1, \ldots, n$, son variables aleatorias distribuídas independientemente como $N\left(\mathbf{x}_i^T\boldsymbol{\beta}, \sigma_\epsilon^2\right)$.

La importancia de cada condición depende del propósito de un modelo ajustado; si no se cumple la condición de linealidad, el modelo estará sesgado localmente y sobre- o sub-estimará $E(Y|\mathbf{x})$ para algunos valores de $\mathbf{x}$. Sin embargo, aún si la relación no es lineal, el *modelo lineal* aún podría proveer una aproximación razonablemente precisa de la relación real desconocida. Si el objetivo es solamente la predicción, entonces la no linealidad de la relación se tolera dado que el modelo ajustado es *suficientemente* preciso para este propósito.

Violaciones a la condición de varianza constante tiene poco efecto sobre los parámetros estimados y, por tanto, en la utilidad predictiva de un modelo ajustado. Varianzas no constantes pueden sesgar los errores estándar $\hat{\sigma}\left(\hat{\beta}_i\right)$; la consecuencia de los errores estándar sesgados es que los intervalos de confianza son o muy anchos o muy delgados y que la probabilidad de que el procedimiento capture el parámetro no sea $1 - \alpha$ como se había propuesto.

La condición de normalidad tiene relativamente poco impacto en la estimación del parámetro y la utilidad predictiva de un modelo ajustado; pero es crítica para los intervalos de confianza si el tamaño de la muestra es pequeño ($n - q < 80$), porque el teorema del límite central asegura que las combinaciones lineales de variables aleatorias será aproximadamente normal para valores grandes de n. Un error común es buscar la normalidad en las respuestas $y_1, \ldots, y_n$; dado que las respuestas tienen esperanza distinta se enturbia el análisis de normalidad. En cambio, la condición de normalidad debe revisarse utilizando los residuos: $\hat{\epsilon}_1 = y_1 - \hat{y}_1, \ldots, \hat{\epsilon}_n = y_n - \hat{y}_n$, o sobre versiones estandarizadas de los residuos. Si el tamaño de la muestra es pequeño y los residuos son claramente no normales, entonces los intervalos de confianza pueden ser inconsistentes con el nivel de confianza.

1.1.1.5. Ejemplo de regresión lineal con R

El conjunto de datos de atletas australianos consiste de mediciones de hematología y morfología recolectadas de atletas de alto rendimiento que participan en 12 deportes. Este conjunto está

disponible en el paquete DAAG de R y puede ser usado como ejemplo del análisis de regresión; el objetivo es determinar la relación entre el pliegue de la piel, el porcentaje de grasa corporal y el género. El porcentaje de grasa corporal se midió sumergiendo a cada individuo em un tanque de agua para determinar la cantidad de agua que desplaza, se cree que es la mejor forma de hacer esta medición. El pliegue cutáneo puede medirse con un *calibre* que puede ayudar a determinar la cantidad de grasa que se encuentra debajo de la piel; generalmente se calcula como el promedio de medidas en diversas partes del cuerpo. Medir pliegues en la piel requiere menos tiempo y es más económico que tener un tanque para determinar la grasa corporal, pero se han presentado preocupaciones respecto a su exactitud para obtener la grasa corporal en niños. Se determinará la relación entre ambas mediciones utilizando datos de atletas australianos con ayuda de R.

Para instalar la biblioteca DAAG:

```
> install.packages('DAAG')
```

Después, en un archivo de *script*, podemos comenzar a escribir:

```
library(DAAG)
head(ais)
str(ais)
```

En la consola de R se puede escribir *?ais* para ver la ayuda de este conjunto y conocer más detalles de sus columnas, especialmente son de interés: *bmi* índice de masa corporal, *ssf* suma de pliegues cutáneos y *pcBfat* porcentaje de grasa corporal.

Para crear una gráfica básica de pligue cutáneo vs porcentaje de grasa, se agrega en el código fuente:

```
plot(ais$ssf,ais$pcBfat)
```

Parece existir una fuerte relación lineal entre ambas variables; se puede agregar una línea de ajuste de mínimos cuadrados con ayuda de las funciones:

- ♢ *lm*: realiza el ajuste de mínimos cuadrados, devuelve un objeto con los elementos de la regresión de y sobre x si el argumento es $y \sim x$.
- ♢ *abline*: extrae la pendiente y la ordenada al origen del objeto que produco *lm* y agrega una línea sobre la gráfica original.

En la consola se pueden obtener los atributos del objeto que devuelve la llamada a *lm*:

El atributo más importante se llama *coefficients* y se puede almacenar en una variable propia:

Regresando al *script*, se puede añadir color a los puntos de la gráfica para distinguir hombres de mujeres:

Para ver una resumen estadístico del modelo ajustado, en el *script* se puede añadir:

```
summary(lm)
confint(lm)
```

La última línea calcula y muestra el intervalo de confianza del 95 % para β_0 y β_1.

Como la relación es diferente entre hombres que entre mujeres, se puede agregar una variable que ayude a distinguirlos:

```
mujeres = as.integer(ais$sex == 'f')
print(mujeres)
```

Este tipo de variables se conocen como *indicador* y, en este caso se define como:

$$x_{i,mujer} = \begin{cases} 1, & \text{si la i-ésima atleta es mujer} \\ 0, & \text{si el i-ésimo atleta es hombre} \end{cases}$$

Se puede incorporar la variable indicadora en el modelo de porcentaje de masa corporal esperada:

$$E(Y|x) = \beta_0 + \beta_1 x_{i,ssf} + \beta_2 x_{i,mujer}$$

Ajustando el modelo con esta nueva variable:

El modelo ajustado resultante para el i-ésimo atleta puede expresarse como:

$$\begin{aligned} \hat{y}_i &= 1.1307 + 0.1579 x_{i,ssf} + 2.9844 x_{i,mujer} \\ &= \begin{cases} 4.1151 + 0.1579 x_{i,ssf}, & \text{si la i-ésima atleta es mujer} \\ 1.1307 + 0.1579 x_{i,ssf}, & \text{si el i-ésimo atleta es hombre} \end{cases} \end{aligned}$$

porque $2.9844 + 1.1307 = 4.1151$. Se estima que las atletas australianas tienen 2.98 % más grasa corporal que los hombres con iguales pliegues cutáneos.

Se pueden añadir líneas de tendencia para hombres y mujeres:

```
abline(a=1.1307, b=0.1579, col='blue')
abline(a=4.1151, b=0.1579, col='red')
```

Finalmente, si se desea revisar si hay diferencias en el porcentaje de grasa corporal entre los atletas agrupados por deporte; el uso de una gráfica de tipo *boxplot* puede ayudar:

```
boxplot(ais$pcBfat ~ ais$sport, cex.axis=.8,
        ylab = 'Porcentaje de grasa corporal')
```

El argumento *ces.axis* sirve para reducir el tamaño de las etiquetas mostradas.

Conclusión

El análisis realizado muestra que el uso de medición de pliegues cutáneos puede ser usado para determinar el porcentaje de grasa corporal con poca pérdida de exactitud; sobre todo si se realiza tomando en cuenta el género de los individuos, es decir, separando los análisis de hombres y mujeres.

Programa:

♢ Implementar el cálculo de los estimadores:

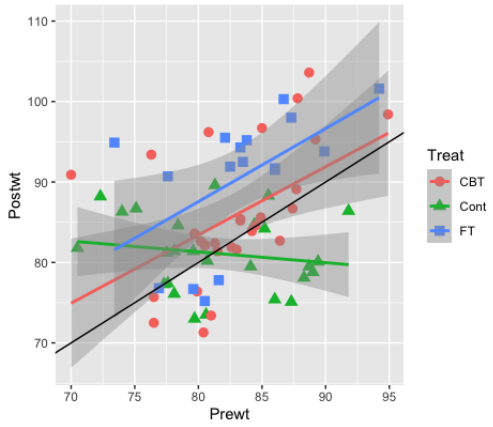
$$\hat{\boldsymbol{\beta}} = \left(\mathbf{X}^T\mathbf{X}\right)^{-1}\mathbf{X}^T\mathbf{y}$$

$$\sigma_\epsilon^2 = \frac{\sum_{i=1}^{n}\left(y_i - \hat{y}_i\right)^2}{n-q}$$

$$var\left(\hat{\boldsymbol{\beta}}\right) = \sigma_\epsilon^2\left(\mathbf{X}^T\mathbf{X}\right)^{-1}$$

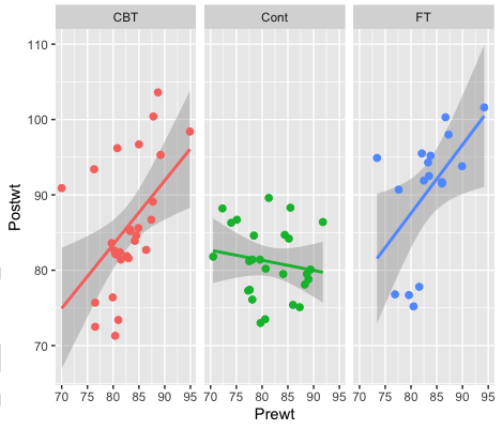
♢ El conjunto de datos *anorexia*, dentro del paquete *MASS* de R reune datos de 72 pacientes que participaron en un experimento para determinar la efectividad de terapias. Se aplicaron tres tipos: terapia familiar (*FT*), terapia cognitiva conductual (*CBT*) y un grupo de control sin terapia (*Cont*). El objetivo del análisis es determinar la efectividad de los tratamientos, así como estimar la media de ganancia de peso, comparando el peso antes y después del tratamiento (*Prewt* y *Postwt*). como evidencia de que son ganancias reales

- Grafica el peso antes y después del experimento identificando los puntos de cada grupo con diferentes formas y colores (como se añadirán más elementos a la gráfica, se sugiere usar *ggplot*)
- Agrega líneas de regresión por mínimos cuadrados usando los mismos colores para identificarlos (ggplot regala los intervalos de confianza)
- Agrega una línea con pendiente 1 y ordenada al origen 0



- ¿Se puede concluir algo sobre la eficacia de los tratamientos?

- Otra forma de ver los datos es dividir la gráfica de acuerdo al tipo de tratamiento; se puede realizar con *facet_grid*:




```
plt = ggplot(anorexia, aes(Prewt, Postwt)) + facet_grid(.~Treat)
plt = plt + geom_smooth(aes(colour = Treat), method ='lm')
plt = plt + geom_point(aes(colour = Treat), size = 2)
plt = plt + scale_colour_discrete(guide="none")
plt
```

- Muestra el resumen de cada modelo de regresión, ¿Para qué grupos existe evidencia de una relación entre los pesos? Explica la diferencia entre el grupo de control y los grupos con terapias; genera una conclusión al respecto.

*

1.1.1.6. Regularización

Un modelo obtenido como la solución de las ecuaciones normales:

$$\mathbf{X}^T\mathbf{X}\boldsymbol{\beta} = \mathbf{X}^T\mathbf{y}$$

$$\hat{\boldsymbol{\beta}} = \left(\mathbf{X}^T\mathbf{X}\right)^{-1}\mathbf{X}^T\mathbf{y},$$

funciona muy bien si el número de muestras (n) es mucho mayor que el número de parámetros (p):

$$n \gg p \Rightarrow \textit{poca varianza}$$

Cuando esta condición no se cumple, se pueden presentar altas correlaciones entre las variables predictoras, provocando que el modelo esté *sobreajustado*.

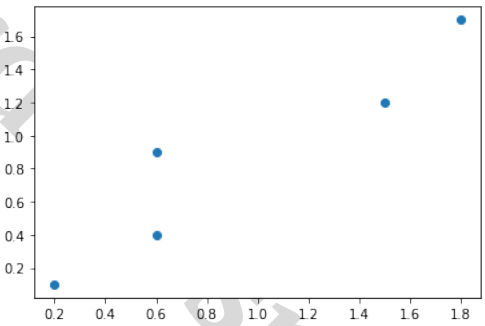
1.1.1.7. Ejemplo de regresión lineal con Python

Revisemos ahora un ejemplo sencillo con Python que servirá para motivar la revisión del concepto de *regularización* para la regresión lineal.

```
import matplotlib.pyplot as plt
import numpy as np
```

Un conjunto de datos *artificial*:

```
x = np.array([0.2,0.6,0.6,1.5,1.8])
y = np.array([0.1,0.4,0.9,1.2,1.7])
plt.scatter(x,y)
```



Al igual que la *selección* de datos de entrenamiento y prueba:

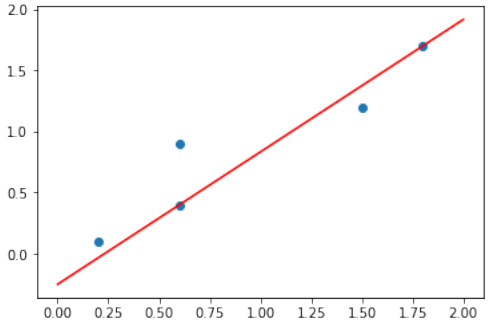
Entrenamos un modelo de regresión lineal, se encuentra dentro de *sklearn.linear_model* y mostramos la recta obtenida:

```
y = -0.2500 + 1.0833x
```

Y su gráfica:

```
xx = np.linspace(0.0,2,2)
yy = modelo.predict(xx.reshape(len(xx),1))

plt.scatter(x,y)
plt.plot(xx,yy,c='r')
```



Parece un buen modelo, pero en realidad está sobreajustado, podemos verlo calculando el error cuadrático medio:

```
from sklearn.metrics import mean_squared_error as mse
mse1 = mse(y_test, modelo.predict(X_test.reshape(len(X_test), 1)))
print('Error : {}'.format(mse1))
```

```
Error : 0.09946759259259248
```

Existen alternativas a la regresión lineal convencional que pueden obtener mejores resultados.

Regularización En general, cuando se realiza una regularización, se utiliza el error cuadrático medio (MSE) como función de costo, añadiendo un término que penaliza la complejidad del modelo:

$$J = MSE + \alpha C,$$

C es la medida de complejidad del modelo. Dependiendo de cómo se mide la complejidad, se tienen distintos tipos de regularización. El hiperparámetro α indica qué tan importante es que el modelo sea simple en relación a qué tan importante es su rendimiento. Cuando se utiliza regularización, se reduce la complejidad del modelo al mismo tiempo que se minimiza la función de costo. Existen dos formas básicas: Lasso ($L1$) y Ridge ($L2$).

Regularización *Lasso* ($L1$) En esta regularización, la complejidad C se mide como la media del valor absoluto de los coeficientes del modelo. Esto se puede aplicar a regresión lineal, polinómicas, regresión logística, redes neuronales, máquinas de soporte vectorial, etc. Matemáticamente se define como:

$$C = \frac{1}{n} \sum_{i=1}^{n} |w_1|,$$

con esto, para el caso del MSE, la función de costo queda:

$$J = \frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 + \alpha \frac{1}{n} \sum_{i=1}^{n} |w_1|$$

Lasso es útil cuando se sospecha que varios de los atributos de entrada (*características*) son irrelevantes. Al usar Lasso, se fomenta que la solución sea poco densa, es decir, se favorecen que algunos de los coeficientes tengan valor 0. Esto puede ser útil para descubrir cuáles de los atributos de entrada son relevantes y, en general, para obtener un modelo que generalice mejor. Lasso funciona mejor cuando los atributos no están muy correlacionados entre ellos.

Regularización *Ridge* ($L2$) En este tipo de regularización, la complejidad C se mide como la media del cuadrado de los coeficientes del modelo. También se puede aplicar a diversas técnicas de aprendizaje automático. Matemáticamente se define como:

$$C = \frac{1}{2n} \sum_{i=1}^{n} w_1^2,$$

y la función de costo:

$$J = \frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 + \alpha \frac{1}{2n} \sum_{i=1}^{n} w_1^2$$

Ridge funciona bien cuando se sospecha que varios de los atributos de entrada están correlacionados entre ellos. Ridge hace que los coeficientes obtenidos sean más pequeños, esta disminución de los coeficientes minimiza el efecto de la correlación entre los atributos de entrada y hace que el modelo generalice mejor. Ridge funciona mejor cuando la mayoría de los atributos son relevantes.

Regularización *ElasticNet* ($L1$ y $L2$) Combina las regularizaciones $L1$ y $L2$; con el parámetro r podemos indicar que importancia relativa tienen Lasso y Ridge respectivamente. Matemáticamente:

$$C = r \times Lasso + (1 - r) \times Ridge,$$

desarrollado para el caso del error cuadrático medio, se obtiene:

$$J = \frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 + r \times \alpha \frac{1}{n} \sum_{i=1}^{n} |w_1| + (1 - r) \times \alpha \frac{1}{2n} \sum_{i=1}^{n} w_1^2$$

ElasticNet se recomienda cuando se cuenta con gran número de características: algunos serán irrelevantes y otros estarán correlacionados.

Nota *scikit-learn* ofrece el hiperparámetro *"penalty"* en muchos de sus modelos; se puede utilizar "$l1$" o "$l2$" en *penalty* para elegir la regularización a usar.

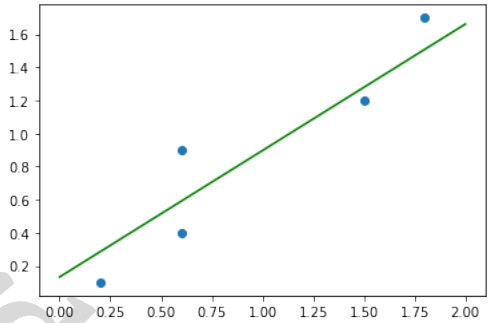
Tomando los mismos valores que el ejemplo de regresión convencional, ahora utilizamos Ridge que también se encuentra dentro de *sklearn.linear_model*:

```
y = 0.1324 + 0.7647x
```

Y su gráfica:

```
xx = np.linspace(0.0,2,2)
yy = modelo2.predict(xx.reshape(len(xx),1))

plt.scatter(x,y)
plt.plot(xx,yy,c='g')
```



Se observa que la recta de ajuste para este modelo pasa más centrada con respecto a los puntos de la muestra, podemos verlo calculando el error cuadrático medio:

```
mse2 = mse(y_test, modelo2.predict(X_test.reshape(len(X_test), 1)))
print('Error : {}'.format(mse2))
```

```
Error : 0.04533737024221454
```

Este error es menor que el obtenido con el modelo de regresión convencional.

Tarea:

- ♢ Iterar el modelo de regresión Ridge variando el valor de α para obtener el mejor para el ejemplo con datos *artificiales* (de preferencia con la implementación propia).
- ♢ Revisar el uso de las regresiones *Lasso* y *ElasticNet*. Comparar con Ridge.

*

	Model	Penalty function
a)	Ridge	$\|J\|_2^2$
b)	LORETA	$\|\mathbb{L}J\|_2^2$
c)	LASSO	$\|J\|_1$
d)	LASSO Fusion	$\|\mathbb{L}J\|_1$
e)	MPLS	$\sum_k \lambda_k \psi_k(J)$
f)	Fused LASSO	$\lambda_1 \sum_{i=1}^{S-1} \|J_{i,.}\|_1 + \lambda_2 \sum_{i=1}^{S-1} \|J_{i+1,.} - J_{i,.}\|_1$
g)	ENET	$\lambda_1 \|J\|_2 + \lambda_2 \|J\|_1$
h)	ENETL	$\lambda_1 \|\mathbb{L}J\|_2 + \lambda_2 \|\mathbb{L}J\|_1$
i)	Smooth LASSO	$\lambda_1 \|\mathbb{L}J\|_2 + \lambda_2 \|J\|_1$
j)	MXN	$\|J\|_{p,q} = \left(\sum_{t=1}^{T} \left(\sum_{i=1}^{S} \|J_{i,t}\|^p\right)^{\frac{q}{p}}\right)^{\frac{1}{q}}$
k)	GLASSO	$\sum_{s=1}^{S} \left(\sum_{t=1}^{T} \|J_{i,t}\|^2\right)^{\frac{1}{2}}$
l)	ELASSO	$\sum_{t=1}^{T} \left(\sum_{i=1}^{S} \|J_{i,t}\|\right)^2$

Figura 1.3: Otras funciones de penalización (https://bit.ly/3Pxv1dr)

1.1.2. Regresión polinomial

En ocasiones una linea recta no es la mejor forma de hacer regresión sobre un conjunto de datos; es común que la relación entre las características y la variable de respuesta (objetivo) no se puede describir correctamente con una línea recta. Es posible que un polinomio funcione mejor para realizar las predicciones de la variable buscada.

Un polinomio puede expresarse de la siguiente manera:

$$y = \beta_0 + \beta_1 x + \beta_2 x^2 \ldots + \beta_n x^n$$

Donde:

- ♢ y es la variable de respuesta que se desea predecir.
- ♢ x es la característica utilizada para realizar la predicción.
- ♢ β_0 es el término independiente.
- ♢ n es el grado del polinomio.
- ♢ $\beta_1 \ldots, \beta_n$ son los coeficientes que deseamos estimar al ajustar el modelo con los valores de x, y disponibles.

Si comparamos la expresión de la regresión polinomial con la usada para regresión lineal:

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 \ldots + \beta_n x_n$$

Notación que son muy similares, esto no es una coincidencia: **la regresión polinomial es un modelo lineal** usado para describir relaciones no lineales; la *magia* recae en crear nuevas características elevando las características originales a alguna potencia.

Por ejemplo, si tenemos una característica x y deseamos un polinomio de tercer grado, la regresión incluirá tanto a x^2 como a x^3 para la estimación de los coeficientes.

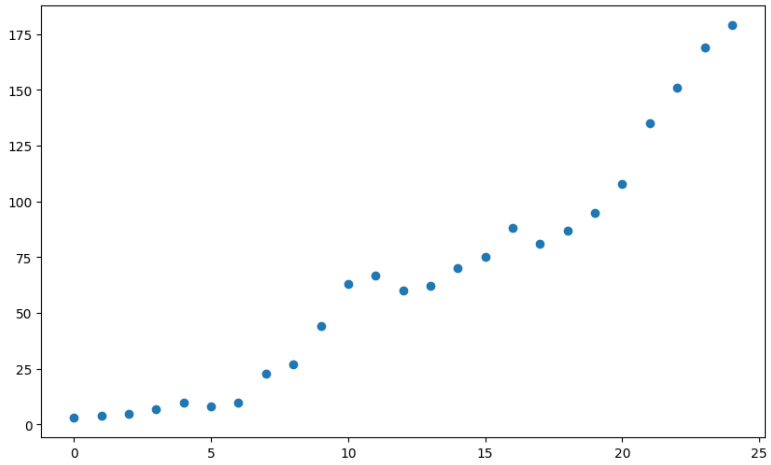
Ejemplo.

Bibliotecas:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

Conjunto de datos:

```
y = [3, 4, 5, 7, 10, 8, 10, 23, 27, 44, 63, 67, 60, 62, 70, 75, 88, 81,
     87, 95, 108, 135, 151, 169, 179]
x = np.arange(len(y))
plt.figure(figsize=(10,6))
plt.scatter(x, y)
plt.show()
```



Vemos que no parece funcionar bien una regresión lineal y buscamos una curva que describa mejor la relación. En particular, opdemos inferir que un polinomio de grado 2. Entonces nuestro modelo es:

$$y = \beta_0 + \beta_1 x + \beta_2 x^2$$

Para crear las características polinomiales, existe *PolynomialFeatures* dentro de *sklearn*:

include_ bias debe tener valos *false* porque la regresión lineal se encargará del término independiente.

Ajustamos y transformamos el objeto *poly*:

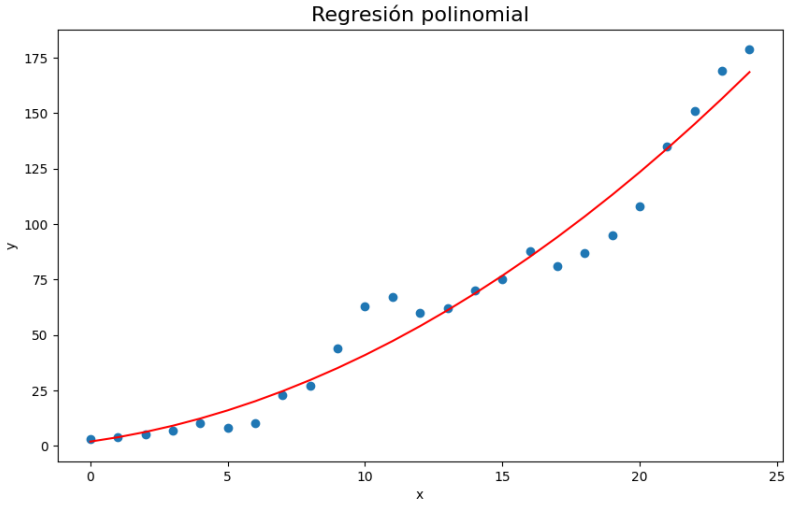
reshape$(-1, 1)$ trnasforma el arreglo de 1D a 2D, necesario para el ajuste. Al observar la salida vemos que el segundo elemento son las características originales elevadas al cuadrado, tal como lo deseamos.

Creamos el modelo lineal:

Ajustamos y predecimos los valores:

Graficamos el resultado:

```
plt.figure(figsize=(10, 6))
plt.scatter(x, y)
plt.plot(x, y_pred, c="red")
plt.title('Regresión polinomial', size=16)
plt.xlabel('x')
plt.ylabel('y')
plt.show()
```



1.1.3. Análisis predictivo

Un problema común en ciencia de datos es identificar el grupo (clase) al cual pertenece una observación, utilizando un conjunto de atributos de dicha observación. Por ejemplo, muchos clientes de email envían los mensajes recibidos a diferentes carpetas, tales como *entrada*, *promociones*, etc. En ocasiones los mensajes contienen *links* a sitios web con malware, en otras son mensajes enviados masivamente, es decir a gran cantidad de personas; resulta deseable que el cliente pueda enviar este tipo de mensajes a la carpeta de *spam* o ponerlos en cuarentena. Sin embargo, es raro que un mensaje entrante contenga todos los atributos para clasificarlo como peligroso, por tanto debe usarse una función predictiva.

Un algoritmo para clasificar mensajes de correo debe extraer datos de los mensajes, tales como la presencia de palabras específica o el número de letras mayúsculas utilizadas; con esta

información se genera un vector de atributos predictores y se procesa con la función de predicción; esta función se encargará de devolver la clase a la que pertenece el mensaje recibido. Antes de utilizar una función para predecir, se debe revisar a detalle para buscar que tenga alta precisión; el tópico de la predicción y el aseguramiento de la precisión se conoce como *análisis predictivo*[2].

1.1.4. La tarea de predicción

El propósito de este tipo de algoritmos es obtener el valor de una variable objetivo a partir de un vector predictor; generalmente el objetivo es una variable categórica: una etiqueta que indica el grupo al que pertenece la observación. El analista no posee datos de la etiqueta pero, en cambio, tiene información codificada en los atributos de su vector.

1.1.5. Funciones de predicción $k-$vecinos

Una función de predicción de este tipo utiliza un conjunto de de observaciones en pares $D = \{(y_1, \mathbf{x}_1), \ldots, (y_n, \mathbf{x}_n)\}$ para los cuales se conocen todos los y_i valores objetivo; una observación objetivo es un par $(y_0, \mathbf{x}_0)$ para el que se desea calcular el valor de su variable objetivo y_0. La versión más sencilla de esta función de predicción para problemas de clasificación opera determinando las k observaciones más cercanas (similares) a $(y_0, \mathbf{x}_0)$, basada en las distancias entre $\mathbf{x}_0$ y $\mathbf{x}_i$; el valor de la predicción será la etiqueta más entre las observaciones más cercanas: sus $k-$vecinos. En el caso de variables cuantitativas, la predicción del algoritmo básico es la media de sus $k-$vecinos.

1.1.5.1. Notación

Una observación con valor objetivo y_0 desconocido, se denota como $\mathbf{z}_0 = (y_0, \mathbf{x}_0)$, en donde el vector predictor $\mathbf{x}_0$ de longitud p ha sido observado y se utilizará para predecir el valor de y_0. La función de predicción $f(\cdot|D)$ se construye a partir del conjunto de datos $D = \{\mathbf{z}_1, \ldots, \mathbf{z}_n\}$; la notación condicional de $f(\cdot|D)$ enfatiza el papel del conjunto de entrenamiento D. La predicción de y_0 se denota como $\hat{y}_0 = f(\mathbf{x}_0|D)$. Por ejemplo, si el objetivo es un valor **cuantitativo**, se puede utilizar una regresión lineal como función predictiva: $f(\mathbf{x}_0|D) = \mathbf{x}_0^T \hat{\boldsymbol{\beta}}$.

En general, la función de predicción no tiene una forma simple y se trata de un algoritmo de varios pasos que toma como entrada $\mathbf{x}_0$ y devuelve la predicción y_0. Si la variable es **cualitativa**, se asume que la variable objetivo es una etiqueta que identifica la pertenencia a algún grupo; si el número de grupos es g, entonces por convención el conjunto de etiquetas es $\{0, 1, \ldots, g-1\}$.

Además, es recomendable contar con funciones indicadoras que identifiquen la pertenencia a los grupos; el indicador de pertencia al grupo j se define como:

$$I_j(y) = \begin{cases} 1, & \text{si } y = j \\ 0, & \text{si } y \neq j \end{cases},$$

donde y es una etiqueta de grupo.

[2]Este es el término usado en ciencia de datos; en estadística se conoce como aprendizaje estadístico y *machine learning* (aprendizaje automático) en computación.

1.1.5.2. Métricas de distancia

Es común utilizar las distancias euclideana y Manhattan para calcular distancias entre vectores predictores.

La **distancia euclideana** entre los vectores $p-$dimensionales $\mathbf{x}_0$ y $\mathbf{x}_i$ se obtiene como:

$$d_E\left(\mathbf{x}_i, \mathbf{x}_0\right) = \left[\sum_{j=1}^{p}\left(x_{i,j} - x_{0,j}\right)^2\right]^{1/2}$$

Si las variables del predictor difieren mucho con respecto a la variabilidad de las variables del conjunto de entrenamiento, es muy recomendable escalar cada variable con la desviación estándar de la misma; sin este proceso, las diferencias serán muy grandes y tenderán a sesgar la predicción. El escalamiento puede calcularse al momento de obtener la distancia como:

$$d_S\left(\mathbf{x}_i, \mathbf{x}_0\right) = \left[\sum_{j=1}^{p}\left(\frac{x_{i,j} - x_{0,j}}{s_j}\right)^2\right]^{1/2},$$

donde s_j es la desviación estándar estimada de la $j-$ésima variable predictora; la varianza se calcula como:

$$s_j^2 = (n-g)^{-1}\sum_{k=1}^{g}\sum_{i=1}^{n} I_k\left(y_i\right)\left(x_{i,j} - \bar{x}_{j,k}\right)^2,$$

donde $\bar{x}_{j,k}$ es la media del atributo j dentro del grupo k; la presencia de $I_k\left(y_i\right)$ asegura que en la sumatoria interna sólo se consideren las observaciones del grupo correspondiente.

La **distancia Manhattan** (*city-block*) entre $\mathbf{x}_0$ y $\mathbf{x}_i$ se calcula como:

$$d_C\left(\mathbf{x}_i, \mathbf{x}_0\right) = \sum_{j=1}^{p}\left|x_{i,j} - x_{0,j}\right|,$$

de igual forma, puede utilizarse escalamiento para evitar sesgos en el cálculo.

En realidad en la mayoría de las aplicaciones de este algoritmo, es más importante elegir el tamaño k del vecindario, dado que las métricas suelen devolver ordenamientos similares para los vecindarios.

Si un predictor consiste de p variables **cualitativas**, entonces la distancia más usada es la de **Hamming** para comparar $\mathbf{x}_0$ y $\mathbf{x}_i$; esta métrica determina el número de atributos que son diferentes entre ambos vectores; se calcula como:

$$d_H\left(\mathbf{x}_i, \mathbf{x}_0\right) = p - \sum_{j=1}^{p} I_{x_{i,j}}\left(x_{0,j}\right),$$

nótese que $I_{x_{i,j}}\left(x_{0,j}\right)$ es 1 siempre que $x_{i,j} = x_{0,j}$ y 0 en otro caso; entonces la suma cuenta el número de atributos distintos entre ambos vectores.

1.1.5.3. La función de predicción de los $k-$vecinos más cercanos

La predicción de y_0 con el algoritmo de $k-$vecinos se obtiene determinando un vecindario de las k observaciones de entrenamiento más cercanas a $\mathbf{x}_0$; después se calcula la proporción de los k vecinos que pertenecen al grupo j y se predice el valor de y_0 como el grupo con la mayor proporción entre los vecinos. Esta regla de predicción es equivalente a predecir que $\mathbf{z}_0$ pertenece al grupo más común entre los k vecinos más próximos.

Formalmente, la función de predicción que estima la probabilidad de pertenencia al grupo j se define como la proporción de los miembros del grupo j entre los k vecinos más cercanos:

$$\widehat{Pr}\left(y_0 = j|\mathbf{x}_0\right) = \frac{n_j}{k}, j = 1, \ldots, g,$$

donde n_j es el número de vecinos dentro de los k más próximos que pertenecen a j. Resulta útil tener una expresión para $\hat{Pr}\left(y_0 = j|\mathbf{x}_0\right)$ en términos de variables indicadoras:

$$\widehat{Pr}\left(y_0 = j|\mathbf{x}_0\right) = k^{-1} \sum_{i=1}^{k} I_j\left(y_{[i]}\right)$$

Los corchetes denotan el valor objetivo del $i-$ésimo vecino más cercano; es decir, el vecino más próximo $\mathbf{z}_{[1]}$ tiene como valor objetivo $y_{[1]}$. Las probabilidades de pertenencia estimadas se agrupan como un vector:

$$\underset{g\times 1}{\widehat{\mathbf{p}}_0} = \left[\widehat{Pr}\left(y_0 = 1|\mathbf{x}_0\right), \ldots, \widehat{Pr}\left(y_0 = g|\mathbf{x}_0\right)\right]^T$$

El último paso para calcular la predicción obtiene el valor más grande dentro de $\widehat{\mathbf{p}}_0$:

$$f\left(\mathbf{x}_0|D\right) = \arg\max\left(\widehat{\mathbf{p}}_0\right)$$

La función arg máx devuelve el índice del elemento más grande de un vector, en este caso, de $\widehat{\mathbf{p}}_0$; en el caso de existir empate entre varios valores, devolverá el primero de ellos. Sin embargo, es mejor idea romper los empates de otra forma, por ejemplo incrementando el valor de k en uno y recalculando $\widehat{\mathbf{p}}_0$.

Un algoritmo computacional eficiente para la predicción con k vecinos consiste de dos funciones principales:

- ♢ obtener el arreglo ordenado $\mathbf{y}^o = \left(y_{[1]}, \ldots, y_{[n]}\right)$ para las entradas $\mathbf{x}_1, \ldots, \mathbf{x}_n$
- ♢ calcular los elementos de $\widehat{\mathbf{p}}$ y utilizar los primeros k elementos de $\mathbf{y}^o$

Las funciones de predicción $k-$vecinos más cercanos son conceptualmente simples y rivalizan con otras más sofisticadas en cuanto a la precisión de sus predicciones; el problema principal es que este tipo de algoritmos pueden ser computacionalmente muy costosos.

1.1.5.4. $k-$vecinos más próximos ponderados exponencialmente

El algoritmo convencional puede mejorarse en precisión; la idea es utilizar todos los vecinos y no solamente k de ellos. A cada vecino se le asigna una medida de importancia (peso) para determinar $f\left(\mathbf{x}_0|D\right)$ de acuerdo a su distancia relativa con $\mathbf{x}_0$; de esta forma, el vecino más próximo recibe el mayor peso y los demás decaen a cero conforme nos movemos a vecinos más distantes. Además, esta forma del algoritmo evita el problema de los empates en la predicción.

Primero, se debe obtener la suma de los pesos para todos los n vecinos; los pesos son:

$$w_i = \begin{cases} 1/i, & \text{si } i \leq k \\ 0, & \text{si } i > k \end{cases},$$

para $i = 1, \ldots, n$; con esto el estimador alternativo de la probabilidad de pertenencia

$$\widehat{Pr}\left(y_0 = j|\mathbf{x}_0\right) = \sum_{i=1}^{k} w_i I_j\left(y_{[i]}\right)$$

Este estimador indica que los pesos son iguales a $1/k$ para los primeros k vecinos y se vuelven 0 para el resto; es difícil justificar esta caída tan abrupta en los valores; una forma más plausible es que el contenido de la información decaiga con cada vecino más lejano, es decir, que tengan una caída suavizada.

El algoritmo de $k-$*vecinos más próximos ponderados exponencialmente* estima la probabilidad de pertenencia utilizando un conjunto de pesos diferentes. En la función convencional, la influencia de los vecinos está determinada por el tamaño k del vecindario; en cambio, con el algoritmo de pesos ponderados, la influencia de los vecinos depende de una constante α, un valor entre 0 y 1 que sirve para afinar la predicción; los pesos se definen como:

$$w_i = \alpha\,(1-\alpha)^{i-1}\,, i = 1, \ldots, n,$$

para $0 < \alpha < 1$; la suma de los pesos será aproximadamente 1, dado que n es grande.

Tarea: _____

♢ Demuestra que para $0 < \alpha < 1$, se cumple que $1 = \sum_{i=1}^{\infty} \alpha\,(1-\alpha)^{i-1}$

_____*

La parte izquierda de la figura 1.4 muestra los pesos correspondientes al algoritmo convencional de $k-$vecinos para valores de 3, 5, 10, y 20; mientras que el derecho presenta los pesos similares en el algoritmo ponderado exponencialmente para diferentes valores de α. Es importante notar que los pesos en $0.33\overline{3}$, 0.2, 0.1 y 0.05 son iguales a los pesos correspondientes a los valores recíprocos de k.

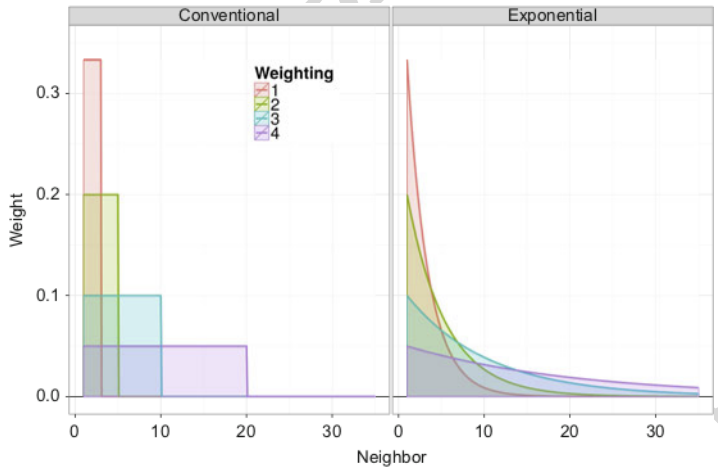


Figura 1.4: $k-$vecinos más próximos ponderados exponencialmente

La elección de α, generalmente es simple si se toma en cuenta que el vecino más próximo recibe el peso α; por ejemplo, $\alpha = 0.2$ implica que el vecino más cercano recibirá el mismo peso que corresponde a la función de predicción convencional de $5-$vecinos más cercanos.

1.1.5.5. Ejemplo de clasificación

Predicción de diabetes; bibliotecas y conjunto de datos:

Conjuntos de entrenamiento y prueba:

Objeto *knn* y ajuste:

Evaluación del modelo:

No parece muy buen rendimiento, algo de preprocesamiento:

Aquí hay un ejemplo de construcción de un sistema de recomendación de *anime* basado en similitudes con $k-$vecinos:
https://gist.github.com/Tahsin-Mayeesha/81dcdafc61b774768b64ba5201e31e0a

1.1.5.6. Regresión $k-$vecinos

Las funciones de predicción $k-$vecinos pueden adaptarse para ser usadas con variables objetivo cuantitativas: para predecir variables cuantitativas, las funciones $k-$vecinos son alternativas a funciones de predicción basadas en regresión.

El objetivo es predecir la variable objetivo y_0, usando un vector predictor $\mathbf{x}_0$ y una función entrenada con un conjunto de pares de variables objetivo y predictoras $D = \{(y_1, \mathbf{x}_1), \ldots, (y_n, \mathbf{x}_n)\}$. La predicción es un promedio ponderado del conjunto ordenado de objetivos $y_{[1]}, \ldots, y_{[n]}$ dado por:

$$\widehat{y} = \sum_{j=1}^{n} w_i y_{[i]}$$

donde el orden $y_{[1]}, \ldots, y_{[n]}$ se determina con las distancias desde $\mathbf{x}_0$ hasta $\mathbf{x}_1, \ldots, \mathbf{x}_n$; los w_i pueden ser tanto los pesos usados función $k-$vecinos convencional como los usados en la función $k-$vecinos exponencialmente ponderados.

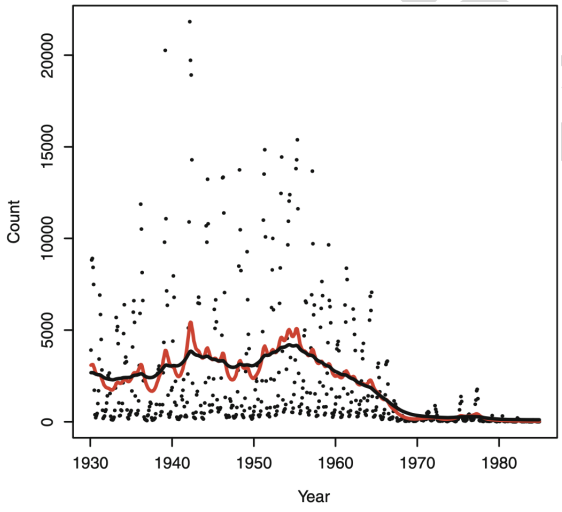


Figura 1.5: $k-$vecinos más próximos usado para regresión

La figura 1.5 muestra el número de casos reportados de **sarampión** por mes en California entre 1930 y 1985; adicionalmente, muestra el número de casos predicho por la función de regresión $k-$vecinos ponderados exponencialmente para dos valores de la constante de *suavizado* α: Las predicciones para el valor $\alpha = 0.05$ (rojo) es mucho menos suave que las predicciones obtenidas con $\alpha = 0.02$ (negro). Estas líneas, que muestran ciertas *tendencias* al eliminar variaciones de corto plazo, se conocen comúnmente como *suavizados*.

En la gráfica se puede observar que existió gran cantidad de variaciones en el número de casos entre 1930 y 1960; en 1963 se obtuvo una vacuna eficiente para el sarampión y los casos comenzaron a bajar hasta ser casi nulos para 1980. El suavizado rojo varios ciclos anuales en el número de casos mientras el suavizado negro captura la tendencia de largo plazo.

1.1.5.7. Ejemplo de regresión

Predicción de la edad de un abulón con el número de anillos; bibliotecas y conjunto de datos:

Tarea: _____

♢ Implementar *desde cero* el algoritmo de clasificación $k-$vecinos más próximos convencional con la medida de distancia euclideana y *city-block*

_____*

1.1.6. Función de predicción: clasificador bayesiano multinomial

La función de predicción bayesiana (inocente, *naive*) es un algoritmo conceptual y computacionalmente simple; se dice que es *inocente* debido a que se presupone que las características son independientes una de la otra. En general su rendimiento no es el mejor cuando se trata de variables predictoras cuantitativas; se comporta aceptablemente bien cuando se tienen variables categóricas y es bastante bueno cuando se tienen variables categóricas con gran cantidad de categorías.

1.1.6.1. Introducción

Considerando problemas de predicción en los que las variables predictoras son categóricas; por ejemplo, los clientes que realizan sus compras en una tienda departamental se pueden clasificar en uno de múltiples grupos según sus hábitos de compra; los datos de entrenamiento pueden consistir de una enorme lista de productos adquiridos, cada uno puede ser identificado por una categoría, tales como perecederos, electrónica, *hardware*, etc. Fácilmente el número de grupos puede exceder los cientos, incluso miles; esto imposibilita el uso del algoritmo de $k-$vecinos; existen varios algoritmos alternativos para atacar este tipo de problemas, pero la *función de predicción multinomial bayesiana* (*inocente*) sobresale por la simplicidad de su algoritmo. Antes de los ejemplos de uso, se muestra el desarrollo de su fundamento matemático.

1.1.6.2. La función de predicción multinomial bayesiana

El problema se establece de la siguiente manera: la clase a la que pertenece la observación $\mathbf{z}_0 = (y_0, \mathbf{x}_0)$, es y_0 y $\mathbf{x}_0$ es su vector predictor; una función predictora $f(\cdot|D)$ se construye a partir de D, el conjunto de observaciones de entrenamiento. La predicción de y_0 es $\widehat{y}_0 = f(\mathbf{x}_0|D)$; la diferencia es que $\mathbf{x}_0$ es un vector de contadores: cada elemento de $\mathbf{x}_0$ almacena el número de veces en que un tipo particular (categoría) o nivel de una variable cualitativa fue observada.

Sea $x_{0,j}$ el número de veces en que un tipo t_j aparece en el *documento* F_0; por ejemplo, si un cliente compró tres productos en el departamento de electrónica, en su nota se puede contabilizar esa cantidad (la nota es el *documento* del que se extrae el dato). Entonces, $\mathbf{x}_0 = [x_{0,1}, \ldots, x_{0,n}]^T$ contiene las frecuencias de los tipos para el documento F_0 y existen n tipos diferentes dentro del conjunto de tipos T.

El *propietario* (clase) de F_0 se denota como y_0 y pertenece al conjunto de propietarios: $y_0 \in \{P_1, \ldots, P_m\}$; la probabilidad de que el tipo t_j se encuentre en F_0 está dada por:

$$\pi_{k,j} = \Pr(t_j \in F_0 | y_0 = P_k)$$

La probabilidad $\pi_{k,j}$ es específica para cada clase, pero no varía entre documentos del mismo propietario. Como t_j pertenece necesariamente a T, entonces $\sum_{j=1}^{n} \pi_{k,j} = 1$. La probabilidad de ocurrencia del tipo t_j puede variar entre diferentes propietarios; si las diferencias son substanciales, entonces se puede utilizar una función de predicción para discriminar entre los propietarios dado un vector de frecuencias $\mathbf{x}_0$.

Por ejemplo, si sólo existen dos tipos, dos propietarios y sus apariciones entre documentos son independientes, entonces la probabilidad de observar un vector particular, digamos $\mathbf{x}_0 = [27, 35]^T$, se puede calcular usando la distribución binomial:

$$\Pr(\mathbf{x}_0 | y_0 = P_k) = \frac{(27+35)!}{27! \times 35!} \pi_{k,1}^{27} \pi_{k,2}^{35}$$

Recordando que la variable aleatoria binomial calcula la probabilidad de observar x *éxitos* entre n *intentos*; la probabilidad de x éxitos y $n-x$ fallas se obtiene como:

$$\Pr(x) = \frac{n!}{x!\,(n-x)!}\pi^x\,(1-\pi)^{n-x},$$

para $x \in \{0, 1, \ldots, n\}$; la expresión anterior toma en cuenta el hecho que $\pi_2 = 1 - \pi_1$.

Los valores de cada $\pi_{k,i}$ son estimaciones basadas en conjuntos de datos previos y se sustituyen sus valores al momento del cálculo.

Con esto, la definición de la función de predicción es:

$$\widehat{y}_0 = f(\mathbf{x}_0|D) = \arg\max\{\Pr(\mathbf{x}_0|y_0 = P_1), \ldots, \Pr(\mathbf{x}_0|y_0 = P_m)\}$$

En general, el número de categorías es sustancialmente mayor que dos y el cálculo de la probabilidad requiere una extensión al de la distribución binomial. Al igual que en la distribución binomial, la versión multinomial supone que el número de ocurrencias de cierto tipo son eventos independientes, entonces la probabilidad de observar al vector de frecuencias $\mathbf{x}_0$ está dada por la función de probabilidad multinomial:

$$\Pr(\mathbf{x}_0|y_0 = P_k) = \frac{(\sum_{i=1}^{n} x_{0,i})!}{\prod_{i=1}^{n} x_{0,i}!}\pi_{k,1}^{x_{0,1}} \times \ldots \times \pi_{k,1}^{x_{0,n}}$$

El término que incluye factoriales se conoce como *coeficiente multinomial*, tal como el coeficiente binomial $n!/x!(n-x)!$. Al ser computacionalmente costoso, se busca evitar su cálculo y puede realizarse para determinar la función de predicción.

Tarea:

♢ Con $\mathbf{x}_0 = [27, 35]^T$, suponiendo que existen tres propietarios y que $\pi_{1,1} = 0.5$, $\pi_{1,2} = 0.3$, $\pi_{2,1} = 0.3$, $\pi_{2,2} = 0.4$, $\pi_{3,1} = 0.4$ y $\pi_{3,2} = 0.5$:

- Calcula $\Pr(\mathbf{x}_0|y_0 = P_k)$ para $k = 1, 2, 3$ y determina la predicción:

$$\widehat{y}_0 = \arg\max\{\Pr(\mathbf{x}_0|y_0 = P_1), \Pr(\mathbf{x}_0|y_0 = P_2), \Pr(\mathbf{x}_0|y_0 = P_3)\}$$

*

1.1.6.2.1. Probabilidad posterior La función predictora de Bayes puede mejorarse al tomar en cuenta información previa. Por ejemplo, al tratar de asignar un documento a algún propietario se puede tomar en cuenta cierto conocimiento histórico sobre documentos ya clasificados para obtener el valor de las *probabilidades previas*. Incluso en ausencia de más evidencias (p.e. el vector de frecuencias), al buscar determinar al propietario de algún documento, se debería elegir a aquel que tenga la mayor cantidad hasta el momento.

La fórmula de Bayes conjunta la información contenida en el vector de frecuencias y la información previa para determinar la probabilidad posterior de la propiedad de algún documento:

$$\Pr(y_0 = P_k|\mathbf{x}_0) = \frac{\Pr(\mathbf{x}_0|y_0 = P_k) \times \Pr(y_0 = P_k)}{\Pr(\mathbf{x}_0)}$$

La predicción de la propiedad será aquella con la mayor probabilidad subsecuente; la predicción se obtiene como:

$$\begin{aligned}\widehat{y}_0 &= \arg\max_k \{\Pr(y_0 = P_k|\mathbf{x}_0)\} \\ &= \arg\max_k \left\{\frac{\Pr(\mathbf{x}_0|y_0 = P_k) \times \Pr(y_0 = P_k)}{\Pr(\mathbf{x}_0)}\right\}\end{aligned}$$

El denominador $\Pr(\mathbf{x}_0)$ escala las probabilidades subsecuentes por el mismo factor y no afecta el orden de las mismas; por tanto puede simplificarse a una expresión más práctica:

$$\widehat{y}_0 = \arg\max_k \{\Pr(\mathbf{x}_0|y_0 = P_k) \times \Pr(y_0 = P_k)\}$$

Como el coeficiente multinomial depende solamente del vector $\mathbf{x}_0$ y tiene el mismo valor para cualquier P_k, puede ser ignorado en el cálculo de $\Pr(\mathbf{x}_0|y_0 = P_k)$, dado que sólo nos interesa determinar cual de las probabilidades posteriores es mayor.

Las probabilidades multinomiales ($\pi_{k,j}$) son desconocidas, pero pueden estimarse a partir de la información previa, evitando el cálculo del coeficiente multinomial; así, la función de Bayes en términos de las probabilidades estimadas es:

$$\begin{aligned}\widehat{y}_0 &= \arg\max_k \left\{\widehat{\Pr}(y_0 = P_k|\mathbf{x}_0)\right\} \\ &= \arg\max_k \left\{\widehat{\pi}_{k,1}^{x_{0,1}} \times \ldots \times \widehat{\pi}_{k,n}^{x_{0,n}} \times \widehat{\pi}_k\right\}\end{aligned},$$

donde $x_{0,j}$ es la frecuencia de ocurrencia del tipo t_j en el documento F_0; la probabilidad estimada $\widehat{\pi}_{k,j}$ es la frecuencia relativa de la ocurrencia de t_j entre todos los documentos pertenecientes a P_k; finalmente, $\widehat{\pi}_k = \widehat{\Pr}(y_0 = P_k)$ es la probabilidad previa estimada de que el documento pertenezca a P_k.

Una vez que se observa al vector $\mathbf{x}_0$, se utiliza la información que contiene para actualizar las probabilidades posteriores. Si no se cuenta con información previa, se deben utilizar probabilidades previas *no informativas*: $\pi_1 = \ldots = \pi_g = 1/g$, suponiendo que hay g propietarios.

Para utilizar la fórmula de Bayes, se necesita estimaciones de las probabilidades $\pi_{k,j}$, $j = 1, \ldots, n$ y $k = 1, \ldots, g$; para obtenerlas se usan las proporciones muestrales o probabilidades empíricas:

$$\widehat{\pi}_{k,j} = \frac{x_{k,j}}{n_k},$$

donde $x_{k,j}$ es el número de ocurrencias de t_j en los documentos pertenecientes a P_k y $n_k = \sum_j x_{k,j}$ es el total de todas la frecuencias de tipos para el propietario P_k. Se puede presentar un subdesbordamiento (*underflow*) si los exponentes de $\widehat{\pi}_{k,j}^{x_{0,j}}$ son grandes y las bases son pequeñas; para evitarlo se busca al propietario con la mayor probabilidad *logarítmica-posterior*:

$$\arg\max_k \left\{\widehat{\Pr}(y_0 = P_k|\mathbf{x}_0)\right\} = \arg\max_k \left\{\log\left[\widehat{\Pr}(y_0 = P_k|\mathbf{x}_0)\right]\right\}$$

Con esto, la versión final de la función de predicción bayesiana multinomial es:

$$\begin{aligned}\widehat{y}_0 &= \arg\max_k \left\{\log\left[\widehat{\Pr}(y_0 = P_k|\mathbf{x}_0)\right]\right\} \\ &= \arg\max_k \left\{\log(\widehat{\pi}_k) + \textstyle\sum_{j=1}^{n} x_{0,j} \log(\widehat{\pi}_{k,j})\right\}\end{aligned}$$

Tarea: _____

♢ Para el mismo ejemplo, suponiendo las probabilidades previas $\pi_1 = 0.8$, $\pi_2 = \pi_3 = 0.1$, determina las probabilidades posteriores $\Pr(y_0 = P_k|\mathbf{x}_0)$ para $k = 1, 2, 3$ y determina la predicción $\widehat{y}_0$.

_____*

1.1.6.3. Ejemplo

Construcción de un detector de *spam* en textos cortos (SMS) en inglés:

Se han realizado estudios y se ha determinado que los mensajes de *spam* contienen palabras como *free, win, winner, cash* y *prize*; además suelen incluir letras mayúsculas y signos de admiración. Este es un problema de clasificación supervisada binaria, dado que los mensajes serán identificados como "*spam*" o "*no spam*" y los datos de entrenamiento están etiquetados, se encuentran en https://archive.ics.uci.edu/ml/datasets/SMS+Spam+Collection.

Como el conjunto de datos es grande (5572 filas) y el modelo sólo acepta valores numéricos como entrada, es necesario hacer un preprocesamiento; utilizaremos el concepto de *Bag of Words* (*BoW* bolsa de palabras). BoW es un término usado para especificar problemas que tienen datos de texto en los que se requiere obtener la frecuencia de cada palabra en el texto tratando cada palabra como un elemento independiente sin importar el orden.

Para esto puede usarse la clase *CountVectorizer* de *sklearn*, esta clase realiza lo siguiente:

♢ Crea *tokens* individuales para cada palabra asignando un ID entero para cada *token*

♢ Cuenta las ocurrencias de cada *token*

♢ Convierte todas las palabras en minúsculas para no tratar la misma palabra como dos distintas si están escritas con alguna letra mayúscula

♢ Ignora los signos de puntuación para no tratar como distintas aquellas palabras que incluyan alguno de estos símbolos

♢ Un parámetro importante es *stop_words* que sirve para no tomar en cuenta las palabras más comúnes del idioma especificado; p.e. si se indica *english* se ignoraran entradas como "a*m*", "*and*", "*the*", etc.

Como ejemplo sencillo, se consideran cuatro documentos y se aplica *CountVectorizer*:

```
# textos a usar
documents = ['Hello, how are you!',
             'Win money, win from home.',
             'Call me now.',
             'Hello, Call hello you tomorrow?']
# objeto CountVectorizer
from sklearn.feature_extraction.text import CountVectorizer
count_vector = CountVectorizer()
# información del objeto
print(count_vector)
# ajuste del modelo
count_vector.fit(documents)
names = count_vector.get_feature_names()
names
```

Ahora se crea una matriz en la que las filas corresponden con cada documento y las columnas con cada palabra; cada celda es la frecuencia de la palabra en ese documento, para esto se utiliza *transform*:

```
doc_array = count_vector.transform(documents).toarray()
doc_array
```

```
array([[1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1],
       [0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 2, 0],
       [0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0],
       [0, 1, 0, 2, 0, 0, 0, 0, 0, 1, 0, 1]])
```

Para facilitar el entendimiento de los datos, se convierte el *array* en un *dataframe* con los nombres de las columnas:

```
import pandas as pd
frequency_matrix = pd.DataFrame(data=doc_array, columns=names)
frequency_matrix
```

Una vez entendido el uso de *BoW*, es posible continuar con el detector de *spam* en SMS.

```
# Importar pandas
import pandas as pd
df = pd.read_csv('https://bit.ly/2kCy3CN',
                 sep='\t',
                 names=['label','sms_message'])
# primeros 5 renglones
# La primera columna determina el tipo de mensaje: 'spam' o 'ham'
df.head()
```

Preprocesamiento Como *sci-kit learn* sólo manipula valores numéricos, se convierten las etiquetas a valores binarios: $0 => "ham"; 1 => "spam"$

```
# Conversion
df.label = df.label.map({'ham':0, 'spam':1})
df.head()
```

Dividiendo los datos para entrenamiento y prueba .

Aplicar BoW a los datos

♢ Primero se ajusta el objeto de tipo *CountVectorizer* con los datos de entrenamiento y devuelve una matriz

♢ Después se transforman los datos del conjunto de pruebas

Clasificador bayesiano multinomial de Sci-Kit Learn Se utiliza la clase *MultinomialNB* de *sklearn*; este modelo es adecuado para clasificación con características discretas, como el caso del *spam*:

Una vez entrenado el modelo, se pueden realizar las predicciones para contrastarlas con el conjunto de pruebas:

```
y_pred = naive_bayes.predict(testing_data)
```

Evaluación del modelo Existen varios mecanismos para realizarlo:

♢ *Accuracy* (exactitud): Determina la frecuencia con la que el modelo realiza una predicción correcta; matemáticamente es la razón entre el número de predicciones correctas entre el número de total de predicciones

$$accuracy = \frac{true\,positives + true\,negatives}{true\,positives + true\,negatives + false\,positives + false\,negatives}$$

♢ *Precision* (precisión): Es la proporción de mensajes clasificados como *spam* (*true positives*) y que realmente lo son (*all positives*)

$$precision = \frac{true\,positives}{true\,positives + false\,positives}$$

♢ *Recall* (sensitividad): Es la proporción de mensajes que son realmente *spam* y que fueron clasificados de esa forma

$$recall = \frac{true\,positives}{true\,positives + false\,negatives}$$

♢ *F1*: Combina las medidas de precisión (*precision*) y sensitividad (*recall*)

$$F1 = 2\frac{PRE \times REC}{PRE + REC}$$

Estas formas de medir modelos están incluídas dentro de *sklearn*:

```
from sklearn.metrics import accuracy_score,precision_score,recall_score,f1_sco
print(' Accuracy: ', format(accuracy_score(y_test, y_pred)))
print('Precision: ', format(precision_score(y_test, y_pred)))
print('   Recall: ', format(recall_score(y_test, y_pred)))
print('       F1: ', format(f1_score(y_test, y_pred)))
```

1.1.6.4. Ventajas y desventajas

Naive Bayes es un algoritmo de clasificación muy usado para procesamiento de lenguaje natural; por esto es importante tener en cuenta sus limitaciones y desventajas.

1.1.6.4.1. Ventajas

♢ **Sencillez y eficiencia**: es un algoritmo simple y computacionalmente eficiente, lo que lo hace sencillo de entender e implementar; además, requiere pocos datos de entrenamiento para estimar los parámetros necesarios para la clasificación.

♢ **Entrenamiento y predicción rápidos**: debido a su sencillez requiere poco tiempo de entrenamiento comparado con otros algoritmos más complejos; esto lo hace apropiado para aplicaciones de tiempo real.

♢ **Trabaja bien con datos categóricos**: este algoritmo supone que las características son categóricas o discretas; se puede utilizar con datos numéricas si se *discretizan* en intervalos.

♢ **Útil con datos de alta dimensionalidad**: se comporta bien cuando el número de caractarísticas es grande comparado con el número de muestras de entrenamiento; por esto es útil para la clasificación de textos y tareas de detección de *spam*.

♢ **Robusto ante características irrelevantes**: este algoritmo ignora las dependencias entre características, esto implica que aún si algunas columnas no son informativas o son redundantes, no afectarán significativamente la exactitud de la clasificación.

1.1.6.4.2. Desventajas

- ♢ **Fuerte dependencia de la suposición**: Si mayor debilidad es la fuerte suposición de independencia entre las características.
- ♢ **Nula interacción entre características**: El algoritmo no puede capturar relaciones entre las características; supone que el efecto de una característica en particular en la clasificación es independiente de la presencia o ausencia de otras características.
- ♢ **Poder de representación limitado**: Este algoritmo se comporta bien cuando las clases son fácilmente separables, pero puede presentar dificultados con límites de decisión complejas; por tal motivo su poder de representación es limitado en comparación con otros algoritmos, como SVM o redes neuronales.
- ♢ **Dependencia de la calidad de los datos**: es un algoritmo que depende en gran medida de la calidad de los datos de entrenamiento; si estos datos con ruidosos o incompletos, puede afectar negativamente en la exactitud del clasificador.

Programa: _____

- ♢ Implementar *desde cero* el algoritmo del clasificador Bayesiano multinomial

_____*

1.1.7. Árboles de decisión/regresión

Los árboles de decisión son un tipo importante de modelos predictivos de aprendizaje artificial (*machine learning*). Existen varios algoritmos para este tipo de árboles desde hace varias décadas y algunas variantes como *random forest* se consideran como técnicas muy poderosas en el área de minería de datos (*data mining*).

El término *Classification and Regression Trees* (CART) fue introducido por Leo Breiman (https://bit.ly/2RvIgwl) en 1984 para referirse a árboles de decisión y regresión que pueden usarse como modelos de clasificación o regresión. El algoritmo CART es el fundamento de otros importantes algoritmos tales como *bagged decision trees*, *random forest* y *boosted decision trees* (https://bit.ly/2NtIUXZ). La idea básica es dividir repetidamente los registros disponibles buscando maximizar la *homogeneidad* en los conjuntos obtenidos en dicha división.

Es un modelo jerárquico de aprendizaje que puede ser usado tanto para clasificación como para regresión: permiten explorar relaciones complejas entre entradas y salidas sin necesidad de hacer suposiciones sobre los datos. Los árboles de decisión pueden verse como una función f que realiza una estimación de un *mapeo* desde un espacio de entrada X hacia un espacio objetivo y: $y = f(X)$. Para el caso de la clasificación, el espacio objetivo y es numérico: $y \in \mathbb{R}$; para la regresión, los valores de y son categóricos: $y \in \{C_1, C_2, \ldots\}$. Se trata de un modelo de aprendizaje supervisado; por tanto, se tienen muestras con la salida esperada. La representación más común para CART es un árbol binario.

Un árbol de decisión divide el espacio X en regiones locales utilizando alguna medida de distancia y el objetivo es determinar particiones bien separadas y homogéneas. Las regiones se dividen de acuerdo a preguntas de prueba y, con esto, se obtiene una división $n-$aria, de forma que mientras se recorre el árbol, en cada nodo se realiza toman decisiones.

Ejemplo 1

La figura 1.6 se muestra un ejemplo sencillo. Los datos se encuentran en dos dimensiones $\{x_1, x_2\}$, también llamadas características, variables ó atributos. Se trata de una clasificación binaria con clases *Redonda* y *Cuadrada*. En el nodo raíz se pregunta si $x_1 > w_{10}$ que divide los datos en un nodo hijo (derecho) con instancias de la clase R y un nodo hijo (izquierdo) que tiene instancias tanto de R como de C. En el hijo izquierdo se realiza una segunda pregunta $x_2 > w_{20}$ para obtener finalmente instancias separadas de las clases R y C. Es importante notar que los resultados obtenidos realizan particiones disjuntas en las variables de entrada.

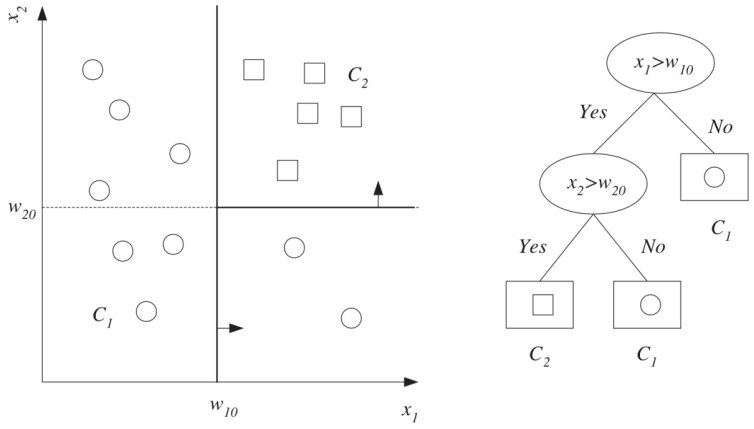


Figure 1.6: Ejemplo de árbol de decisión/regresión (derecha) y sus datos (izquierda)

Ejemplo 2

La figura 1.7 muestra otro ejemplo sencillo para determinar si se debe cargar paraguas o no, de acuerdo a la predicción del estado del tiempo.

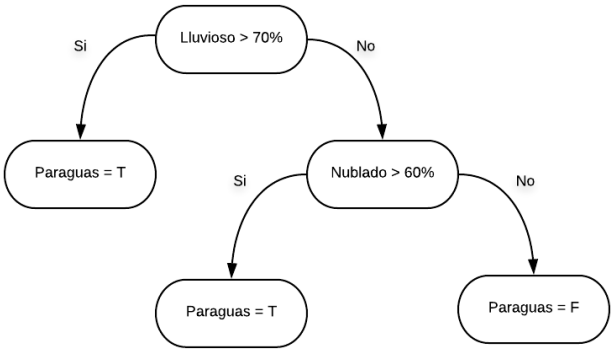


Figure 1.7: Ejemplo de árbol de decisión para llevar paraguas

El algoritmo 1.1 muestra la forma de obtener los valores para las particiones.

Algorithm 1.1 Obtención de particiones

elegir una variable x_i

- ♢ elegir algún valor s_i para x_i que divida los datos de entrenamiento en dos particiones (no necesariamente iguales)
- ♢ medir la *pureza* (homogeneidad) resultante en cada partición
- ♢ usar otro valor s_j para x_i buscando incrementar la pureza de las particiones hasta alcanzar el umbral de *pureza aceptable*

repetir el proceso de partición con una variable diferente (posiblemente alguna usada previamente)
cada valor obtenido se convierte en un nodo en el árbol

Para determinar la homogeneidad de las particiones, se tienen dos posibilidades:

1.1.7.1. Grado de impureza de Gini

El índice de impureza en un rectángulo A que contiene m clases, se calcula como:

$$I(A) = 1 - \sum_{i=1}^{m} p_i^2$$

Con p la proporción de casos en el rectángulo A que pertenecen a la clase i. Es importante notar que:

- ♢ $I(A) = 0$ cuando todos los casos pertenecen a la clase i; es decir, es totalmente homogénea.
- ♢ El valor máximo sucede cuando todas las clases se encuentran igualmente representadas (0.5 para el caso binario).

1.1.7.2. Grado de entropía

El grado de entropía en un área A que contiene m clases, se calcula como:

$$E(A) = \sum_{i=1}^{m} p_i \times log_2(p_i)$$

Con p la proporción de casos en A que pertenecen a la clase i. Es importante notar:

- ♢ $E(A) = 0$ cuando todos los casos pertenecen a la clase i; es decir, es totalmente homogénea.
- ♢ El valor máximo $log_2(m)$ sucede cuando todas las clases se encuentran igualmente representadas.

Problemas con CART

- ♢ Pueden ser poco robustos: un cambio pequeño en los datos de entrenamiento generan grandes cambios en la estructura del árbol.
- ♢ Obtener el mejor árbol de decisión es $NP - completo$; por tanto, en la práctica para su construcción se utilizan heurísticas basadas en óptimos locales.

♢ No es tan difícil obtener modelos sobreajustados que no generalizan bien las muestras; por esto es necesario establacer un límite en la obtención de particiones.

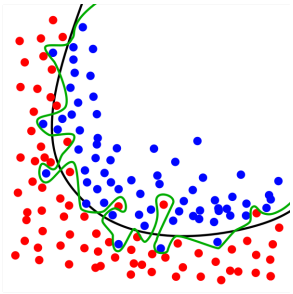


Figure 1.8: Ejemplo de sobreajuste

Ejemplo 3

Para el caso de variables categóricas el proceso es más simple. En la figura 1.9 se observa un ejemplo de datos categóricos para determinar si se debe jugar tennis o no, de acuerdo a la predicción del estado del tiempo.

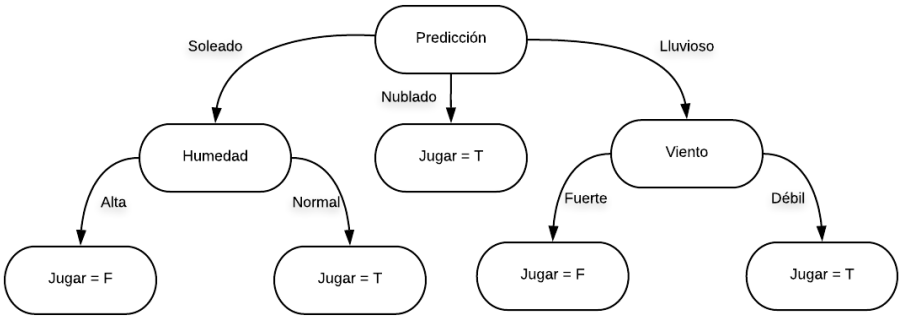


Figure 1.9: Ejemplo de árbol de decisión con variables categóricas

Extracción de reglas

Una vez constuidos, los árboles de decisión son altamente interpretables y es fácil entender la información que representan como reglas del tipo *if - then*: cada camino desde un nodo hasta una hoja es una conjunción de condiciones que deben ser satisfechas para llegar a ese nodo terminal. Para el último ejemplo, tenemos:

♢ **if** P=S **and** H=N **then** J=T

♢ **if** P=N **then** J=T

♢ **if** P=Ll **and** V=D **then** J=T

1.1.8. Función de predicción: *eXtreme Gradient Boosting* (*XGBoost*)

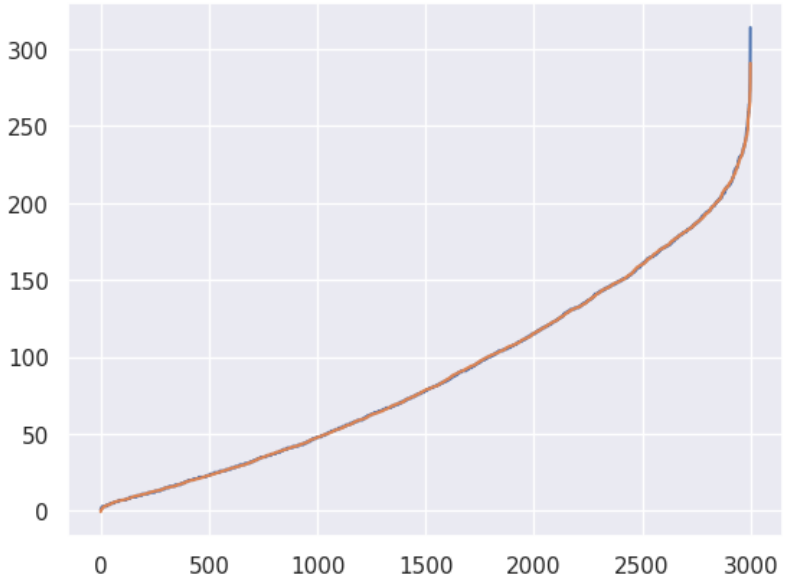
XGBoost es una biblioteca optimizada y distribuída de mejora del gradiente diseñada para ser altamente eficiente, flexible y portable. implementa algoritmos de aprendizaje automático bajo la idea del mejora del gradiente (*Gradient Boosting*). *XGBoost* provee mejora en la construcción de árboles paralelizada.

XGBoost es una de las implementaciones más populares y eficientes del algoritmo *Gradient Boosted Trees*, un método supervisado de aprendizaje basado en aproximación de funciones al optimizar las funciones de pérdida mientras aplica varias técnicas de regularización.

La propuesta original se encuentra en: https://arxiv.org/pdf/1603.02754.pdf.

Error medio absoluto:

```
mae = metrics.mean_absolute_error(y_test, y_hat)
print("Mean Absolute Error = ", mae)
```



Tarea:

♢ Dentro de los conjuntos de datos incluidos en SciKitLearn existe uno con dígitos escritos a mano

```
from sklearn.datasets import load_digits
digits = load_digits()
```

♢ Aplica los modelos *k-vecinos*, *bayesiano multinomial* y *XGBoost* y compara los resultados con las cuatro métricas presentadas

*