Redes Neuronales en R

Jaqueline Girabel y Agustín Muñoz González

UNIVERSIDAD DE BUENOS AIRES

25 de Agosto de 2020

Introducción

La inteligencia artificial (AI) es un campo de estudio dedicado a la simulación de la inteligencia humana en máquinas, programadas para pensar como humanos e imitar sus acciones.

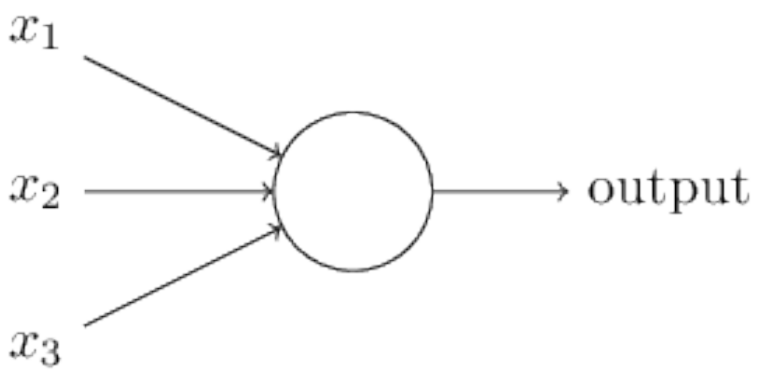
Machine learning es una rama de la AI centrada en el desarrollo de programas capaces de aprender automáticamente a partir de la experiencia, con el fin de resolver problemas y predecir comportamientos basados en los datos de entrada. En este contexto, las *redes neuronales artificiales* (ANN o directamente NN) surgen de la idea de imitar la funcionalidad del cerebro humano y son ejemplos de este tipo de algoritmos de aprendizaje automático.

Perceptron

Empecemos con el caso mas sencillo de neurona artificial, el perceptron.

Fueron desarrollados en los 50s y 60s por el científico Frank Rosenblatt, inspirado a su vez en trabajos previos de Warren McCulloch y Walter Pitts.

El perceptron toma varias entradas binarias, $x_1, x_2, \ldots$, y produce una única salida binaria.



Perceptron

Rosenblatt propuso una sencilla regla para computar la salida. Introdujo *pesos* $w_1, w_2, \ldots$, que son números reales que expresan la importancia de la respectiva input para con el output.

La salida de la neurona, 0 ó 1, es determinada según si la suma pesada $\sum_j w_j x_j = w \cdot x$ es mayor o menor que un cierto valor límite $-b$ llamado *umbral* (*bias* en inglés).

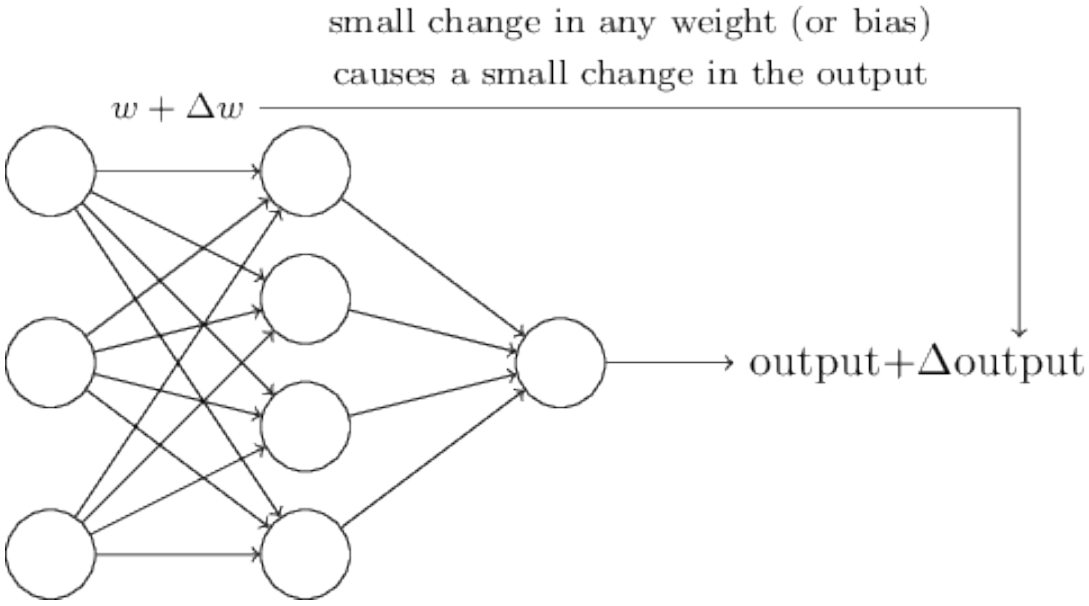
$$output = \begin{cases} 0, & \text{si } w \cdot x + b \leq 0, \\ 1, & \text{si } w \cdot x + b > 0. \end{cases}$$

El umbral puede ser pensado como una medida de cuan fácil es para el perceptron devolver un 1.

Aprendizaje

La idea será entonces variar los pesos y el umbral de forma que la salida de la red sea lo mas parecido a lo que buscamos.

La manera en que el algoritmo **aprende** es generando pequeños cambios en los pesos y el umbral de la red de forma de obtener pequeños cambios en la salida de la red, y repetir este proceso hasta obtener el resultado buscado.

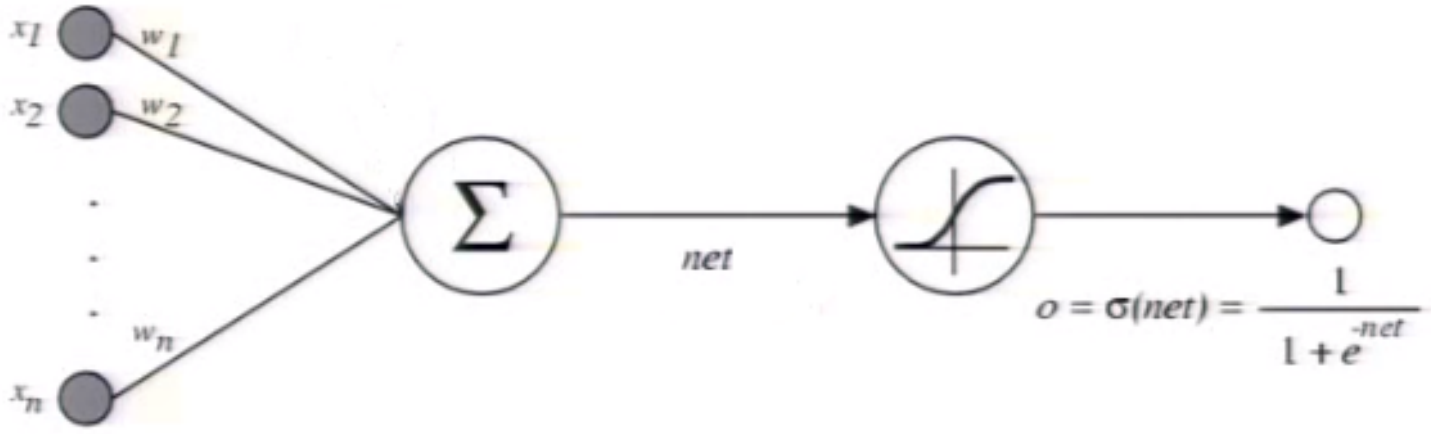


Neuronas sigmoides o logísticas

El problema es que este aprendizaje no es posible con una red de perceptrones, ya que un pequeño cambio en el peso de un perceptron podría cambiar la salida de 0 a 1...y eso no sería un pequeño cambio en la salida.

La forma de superar este problema es usando **neuronas sigmoides o logísticas** (sigmoid neurons).

Una neurona sigmoide es similar a un perceptron pero modificado para que un pequeño cambio en la entrada genere un pequeño cambio en la salida.



Neuronas sigmoides o logísticas

Al igual que el perceptron, las neuronas sigmoides tienen entradas $x_1, x_2, \ldots$, pero en vez de ser binarias pueden tomar cualquier valor real.

También poseen pesos $w_1, w_2, \ldots$ y un umbral b, pero la salida no es binaria sino una cierta función

$$\sigma(w \cdot x + b) = \frac{1}{1 + \exp(-\sum_j w_j x_j + b)},$$

llamada *función sigmoide o logística*.

Notar que incluso matemáticamente hay similitud en la salida de las neuronas sigmoides y los perceptrones, ya que

- si $z = w \cdot x + b >> 0$ entonces $\sigma(z) \approx 1$;
- si $z = w \cdot x + b << 0$ entonces $\sigma(z) \approx 0$.

Neuronas sigmoides

Más formalmente, las neuronas sigmoides son perceptrones suavizados de forma que

$$\Delta output \approx \sum_j \frac{\partial output}{\partial w_j} \Delta w_j + \frac{\partial output}{\partial b} \Delta b,$$

donde $\frac{\partial output}{\partial w_j}$, $\frac{\partial output}{\partial b}$ son las derivadas parciales de la salida con respecto a los pesos w_j y el umbral b.

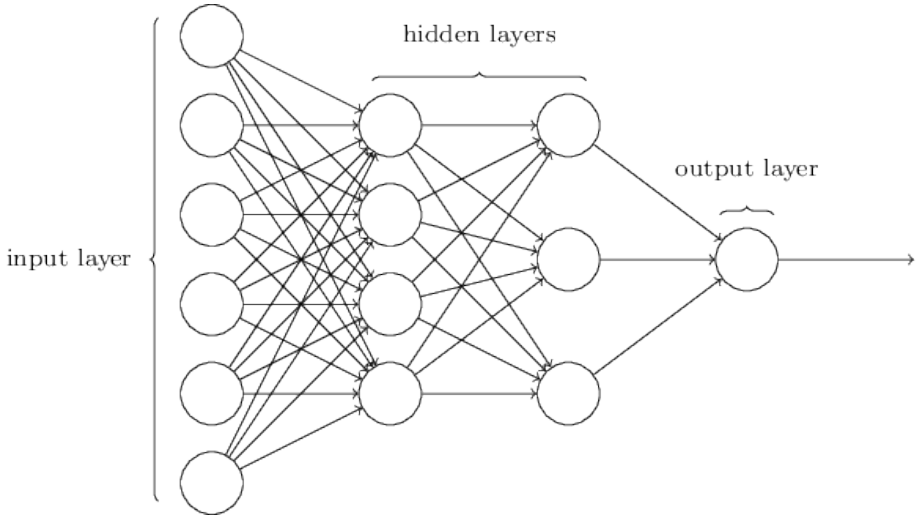
Lo que la fórmula nos dice es que la variación en la salida ($\Delta output$) es lineal con respecto a variaciones en los pesos (Δw_j) y el umbral (Δb). Y esta linealidad facilita la elección de pesos y umbrales para obtener el cambio deseado en la salida!

En general, uno puede tomar como salida $f(w \cdot x + b)$, para cierta f llamada *función de activación*.

Arquitectura de una NN

Las redes neuronales están formadas por las siguientes **capas** (layers)

- Capa de entrada o input layer: Es la primer capa de la red y está formada por las entradas $x_1, x_2, \ldots$.
- Capa oculta o hidden layer: Son las capas intermedias y es donde ocurre la 'magia' de las NNs. De hecho hasta hace pocos años esta zona de la NN era una especie de caja negra dónde no se terminaba de comprender bien la interacción entre todas las neuronas...aún hoy no está totalmente clara esta interacción.
- Capa de salida o output layer: Es la capa final de la red formada por una única neurona, la salida de la red.



Arquitectura de una NN

El procesamiento de la información a través de las diferentes capas es de dos tipos

- Propagación hacia adelante o forward propagation: Es el procesamiento de la capa de entrada a las ocultas, y de estas a la capa de salida. Digamos es el flujo **directo** de una capa a la siguiente.
- Propagación hacia atrás o backpropagation: Es el procesamiento de la información desde la capa de salida hacia las capas ocultas para así, junto con un algoritmo de optimización, calibrar los pesos y el umbral con el objetivo de minimizar el error de predicción. Es decir, es el flujo **cíclico** de la capa de salida a las ocultas. Estos ciclos se repiten las veces que sean necesarias hasta obtener el error deseado. Cada cíclo se llama *epoch*.

Loss Function

Lo que se pretende es que el algoritmo encuentre los pesos y el umbral necesarios para que la salida aproxime el valor real $y(x)$, para todas las entradas x.

Para cuantificar qué tan cerca nos encontramos de este objetivo, se utiliza alguna función de error (como, por ejemplo, el Error Cuadrático Medio), comúnmente llamada *función de costo*. Se descompone el error sobre el dataset completo como el promedio de los errores en cada ejemplo:

$$L(w, b) = \frac{1}{n} \sum_{1 \leq i \leq n} L_i(y(x^i), a_i),$$

donde w son los pesos, b es el umbral, n es el número de entradas, a_i es la i-ésima predicción y L_i es la i-ésima pérdida.

Y como $L(w, b) \geq 0$ para cualquier w, b, lo que queremos es $L(w, b) \approx 0$.

Descenso por el gradiente

Recurrir a un análisis de optimización de L para identificar sus mínimos locales puede resultar costoso y computacionalmente inviable: las NNs más grandes pueden llegar a tener funciones de costo que dependen de millones de parámetros.

Descenso por el gradiente es el nombre que recibe uno de los algoritmos iterativos más populares, que en el contexto de redes neuronales es empleado con el fin de actualizar los parámetros de la función de costo $L(w, b)$ en la dirección del gradiente ∇L.

Descenso por el gradiente y Learning rate

Más concretamente, dado que ∇L apunta hacia la dirección de máximo crecimiento de L, el algoritmo consiste en:

- Inicializar los parámetros $\theta_0 = [w_0, b_0]$ de forma aleatoria. Por ejemplo, $\theta_0 = 0$.
- En el paso t, actualizar los parámetros en la dirección indicada por el gradiente, mediante la instrucción

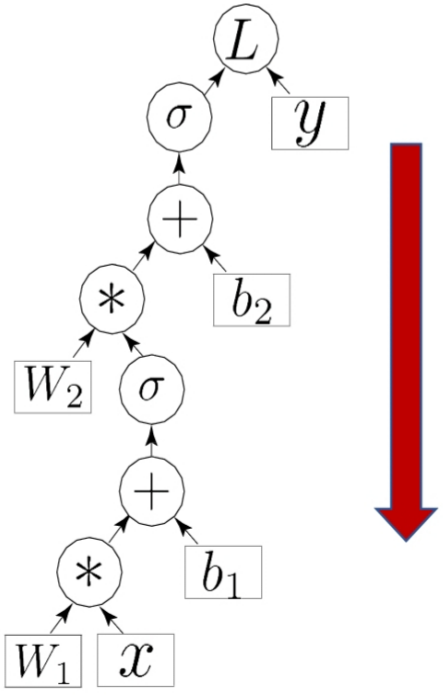
$$\theta_t \leftarrow \theta_{t-1} - \eta \nabla L,$$

donde $\eta > 0$ es conocido como *learning rate* y su elección no es aleatoria pero tampoco hay una regla general para determinarlo.

Ajustar el *learning rate* es esencial para lograr que el algortimo anterior funcione correctamente: si η es demasiado chico, el procedimiento se vuelve muy lento; y si, por el contrario, η es muy grande, posiblemente los parámetros den saltos alejados de un mínimo local, provocando tal vez que el algoritmo diverja.

Backpropagation

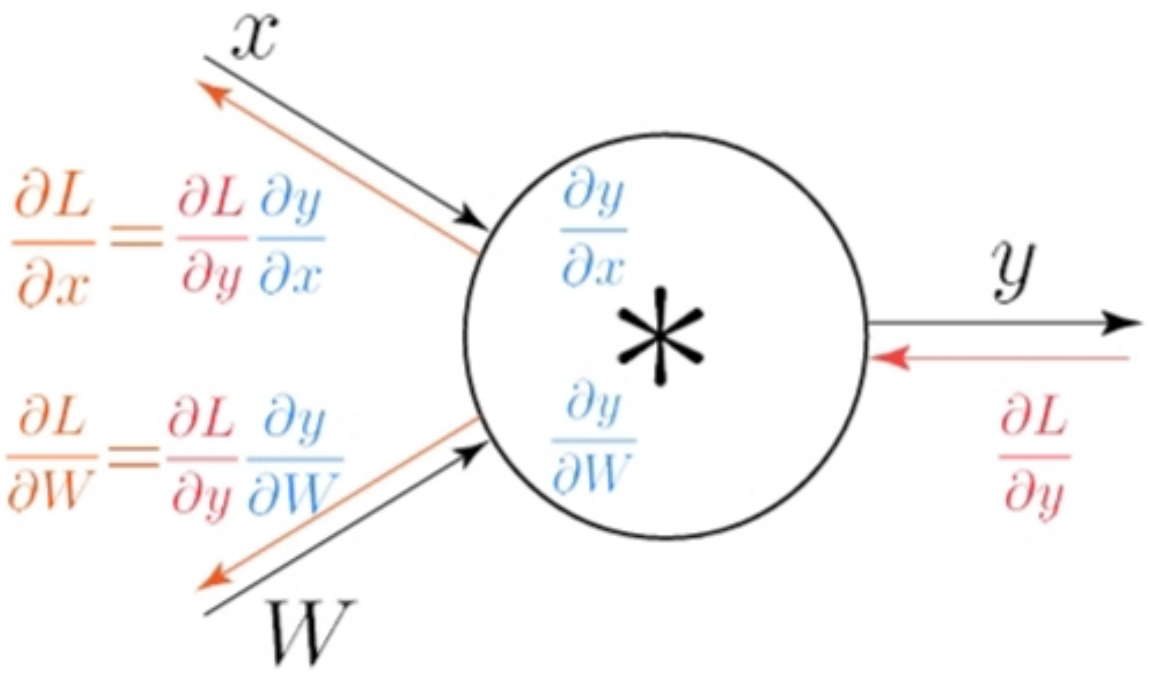
¿Cómo obtenemos el gradiente ∇L?



Backpropagation es una de las técnicas utilizadas para obtener las derivadas parciales $\frac{\partial L}{\partial w}$ y $\frac{\partial L}{\partial b}$. Este algoritmo recorre el grafo de cómputo de la red neuronal en sentido inverso: calcula el gradiente del error con respecto a cada calculo intermedio y cada parámetro.

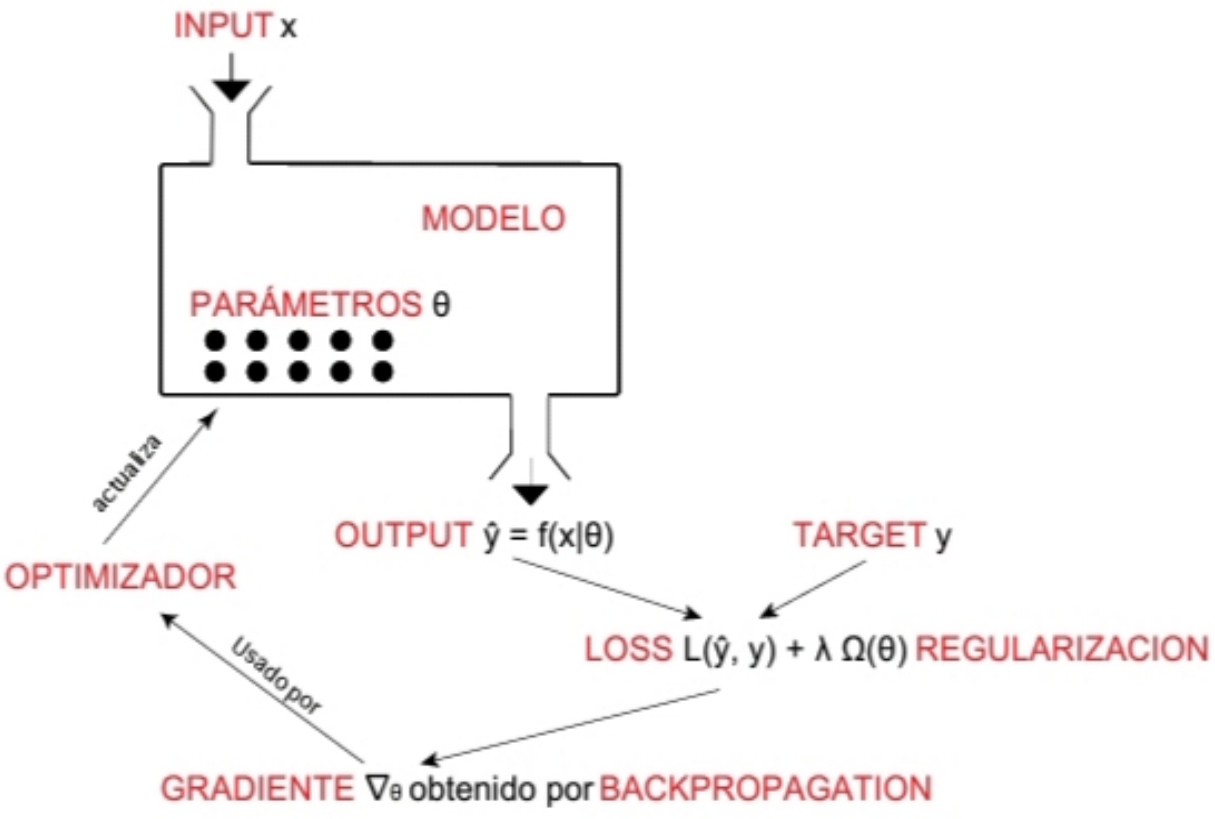
Backpropagation

Backpropagation en el grafo de cómputo



Ciclo de entrenamiento

Entrenamiento de una red neuronal



Pros y Contras de las NNs: Pros

Analicemos por último algunas ventajas y desventajas de las redes neuronales. Empecemos por las ventajas.

- Son flexibles y pueden ser usadas tanto para problemas de regresión como para problemas de clasificación. Todo dato que pueda ser transformado numéricamente puede usarse en estos modelos, ya que en el fondo se trata de un modelo matemático con aproximación de funciones;
- Son buenas para modelar datos no lineales con un gran número de entradas; por ejemplo imágenes;
- Una vez entrenada, las predicciones son muy rápidas;
- Pueden ser entrenadas con cualquier número de entradas y de capas;

Pros y Contras de las NNs: Contras

Mencionemos ahora algunas desventajas.

- Todavía siguen siendo cajas negras en el sentido de que no sabemos exactamente cómo influyen las variables independientes en las variables dependientes;
- Los entrenamientos son muy costosos en términos computacionales y en términos de tiempo, con la tecnología actual;
- Dependen mucho de los datos de entrenamiento. Esto lleva a problemas de over-fitting y de generalización;
- Trabajan mejor con muchos datos. Con lo cual si no se tiene una buena cantidad de información del evento que estamos estudiando no es un buen modelo.

Para terminar...un poquito sobre optimización de los datos

Reducción de la dimensión

La *reducción de la dimensión* o *reducción de la dimensionalidad* es el proceso de reducción del número de variables aleatorias explicativas que caracterizan a la variable respuesta que estamos tratando.

Permiten simplificar la complejidad de espacios muestrales con muchas dimensiones a la vez que conservan su información.

Supongamos que existe una muestra con n individuos cada uno con p variables $(X_1, X_2, \ldots, X_p)$, es decir, el espacio muestral tiene dimensión p.

El objetivo de estos métodos es encontrar un número de factores subyacentes $(z < p)$ que expliquen aproximadamente lo mismo que las p variables originales, i.e. donde antes se necesitaban p valores para caracterizar a cada individuo, ahora bastan z valores.

PCA: Nociones básicas

Una de las técnicas de reducción de la dimensión más utilizada es la llamada *Análisis de Componentes Principales* o *Principal Component Analysis* (PCA).

Es una técnica utilizada para describir un conjunto de datos en términos de nuevas variables (componentes) no correlacionadas.

Las componentes se ordenan por la cantidad de varianza original que describen, de modo de poder 'descartar' las componentes menos significativas. Por lo que la técnica es útil para reducir la dimensionalidad de un conjunto de datos.

Esta convierte un conjunto de observaciones de variables posiblemente correlacionadas en un conjunto de variables sin correlación lineal llamadas *componentes principales*.

PCA: Construcción

Cada componente principal Z_i se obtiene por combinación lineal de las variables originales.

La primera componente principal de un grupo de variables $(X_1, X_2, \ldots, X_p)$ es la combinación lineal con mayor varianza

$$Z_1 = \phi_{11} X_1 + \phi_{21} X_2 + \cdots + \phi_{p1} X_p,$$

donde el vector de coeficientes $(\phi_{11}, \ldots, \phi_{p1})$ tiene norma 1.

Estos coeficientes, o *loadings*, son los que definen a la componente y pueden interpretarse como el peso/importancia asignado a cada variable.

PCA...seguimos construyendo

Dado un set de datos X con n observaciones y p variables, el proceso a seguir para calcular las componentes principales es:

- Centralización de las variables: se resta a cada valor la media de la variable a la que pertenece de forma que todas las variables tengan media cero.
- Se resuelve un problema de optimización para encontrar el valor de los loadings con los que se maximiza la varianza. Para ello se calculan los autovectores y autovalores de la matriz de covarianzas, se ordenan los autovectores ordenados de mayor a menor autovalor (que son todos positivos) y se 'descartan' los autovectores menos significativos.

La selección de componentes principales dependerá de cada problema, dado que no existe un método general que permita identificar cual es el número óptimo de componentes principales a utilizar.

Estadístico S2N (Signal-to-Noise)

Otra de las técnicas empleadas con el fin de reducir la cantidad de variables explicativas tiene que ver con la detección del impacto y de la relevancia que cada una de ellas tiene sobre la variable respuesta.

La relación *señal-ruido* es una medida utilizada con el objetivo de comparar el nivel de una señal deseada con el nivel de ruido de fondo, que compiten en algún conjunto de observaciones.

Una *señal* es una descripción de la información significativa, o deseada, asociada a una o más variables. Por otro lado, el nivel de variación en una variable es a menudo denominado *ruido*, y es usualmente definido como la desviación estándar.

Estadístico S2N (Signal-to-Noise)

Existen muchas definiciones de relaciones señal-ruido, utilizadas en diferentes contextos. Nosotros exhibimos y utilizaremos la siguiente:

Dados vectores v y w, con medias μ_v, μ_w y desvíos σ_v y σ_w, respectivamente, se define el estadístico signal-to-noise como

$$S/N = \frac{\mu_v - \mu_w}{\sigma_v + \sigma_w}.$$

Este cociente permite identificar vectores con diferencias considerables. Cuanto mayor es la diferencia de los promedios de ambos vectores, mayor es el valor absoluto de signal-to-noise y, por lo tanto, la magnitud de la señal - relativa al ruido - es relevante y representativa.

FIN :)