

Tema 1: Regresión

Aprendizaje automático

Héctor Lacueva Sacristán

Índice

Regresión	2
Regresión lineal	2
Nomenclatura	2
Regresión polinómica	2
Productos cruzados	2
Redes Neuronales	2
Ingeniería de features vs Aprendizaje con Redes Neuronales	2
Ingeniería de features	2
Redes neuronales	3
Mínimos cuadrados	3
Mínimos cuadrados con matrices	3
Algoritmo de Descenso de Gradiente	3
Ecuación Normal	3
Descenso de Gradiente vs Ecuación Normal	3
Descenso de Gradiente Estocástico (SGD)	4
Escalado de Atributos	4

Regresión

En este tema se ven conceptos relacionados con la regresión. La regresión sirve para predecir una variable continua.

Regresión lineal

En base a un conjunto de atributos o entradas con un conjunto de pesos asociados se puede predecir el valor de la salida de la función.

En el aprendizaje supervisado se conoce la “**respuesta correcta**” para cada ejemplo de entrenamiento.

Sirve para resolver **problemas muy sencillos**.

Nomenclatura

- **Muestras de entrenamiento:** $\mathcal{D} = \{(x^{(i)}, y^{(i)})\}_{i=1}^N$
- **Variables de entrada o atributos:** x_1, \dots, x_D y $x_0 = 1$
 - El conjunto se representa como $X = (x_0, x_1, \dots, x_D)^T$
- **Variable de salida u objetivo:** y
- **Parámetros o pesos:** $\theta = (\theta_0, \theta_1, \dots, \theta_D)^T$ o $w = (w_0, w_1, \dots, w_D)^T$
 - Al w_0 o θ_0 se le llama **intercept o bias** y representa la intersección de la recta con el eje Y.
 - El resto se llaman **weights o pesos**.

La función de regresión lineal tiene la siguiente forma:

- Con un atributo: $\hat{y} = h_\theta(x) = \theta_0 + \theta_1 x$
- Con varios atributos: $\hat{y} = h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \dots + \theta_n x_n$
 - Donde cada x_i es un atributo distinto.
- Generalizando: $\hat{y} = h_\theta(x) = \theta^T X$

Regresión polinómica

Si queremos hacer un ajuste de un polinomio:

- Con un solo atributo: $h_\theta(x) = \theta_0 + \theta_1 x + \theta_2 x^2 + \dots + \theta_n x^n$
- Con varios atributos: $h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1^2 + \theta_4 x_2^2$

Podemos resolverlo con regresión lineal tomando:

$$\phi(x) = (1, x, x^2, \dots, x^n)^T$$

$$h_\theta(x) = \theta^T \phi(x)$$

Productos cruzados

Sirven para capturar dependencias entre atributos. Por ejemplo:

$$h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1^2 + \theta_4 x_2^2 + \theta_5 x_1 x_2$$

Estaría bien informarse mejor de esto ...

Redes Neuronales

Para funciones simples (polinomios y **¿algo más?**):

- Una Red Neuronal con 1 capa oculta puede aproximar cualquier función con el grado de precisión que se desee, con suficiente número de neuronas.

Para funciones más complejas:

- Redes Neuronales con más capas (donde la mejor función de activación es la RELU) y menos neuronas por capa.

Ingeniería de features vs Aprendizaje con Redes Neuronales

Ingeniería de features

- Se usa para modelos sencillos basados en la intuición y que requieren pocos datos.
- Son fáciles de interpretar.

- Rápida de entrenar, solución analítica y coste convexo (obtiene atributos óptimos).
- **Requiere el arte de elegir los atributos adecuados para el problema.**

Redes neuronales

- Se usan para modelos más complejos, redes profundas y que requieren muchos datos.
- No son fáciles de interpretar, se suele ignorar el comportamiento interior (Modelo de caja negra).
- Costosa de entrenar, necesita de GPU o TPU¹ y puede tener mínimos locales (atributos buenos pero no óptimos).
- **Requiere el arte de elegir una estructura y tamaño de la RN.**

Mínimos cuadrados

Estimación de Máxima Verosimilitud (MLE)

Minimizar el coste cuadrático (o coste L_2).

$$\hat{w} = \operatorname{argmin}_w J(w)$$

$$J(w) = \frac{1}{2} \sum_{i=1}^N (w^T x^{(i)} - y^{(i)})^2 = \frac{1}{2} \sum_{i=1}^N (y_{pred}^{(i)} - y^{(i)})^2$$

La función de coste $J(\theta_0, \theta_1)$ depende de los parámetros $\theta_0, \theta_1, \dots, \theta_n$

Mínimos cuadrados con matrices

Matriz de diseño	Pesos	Salidas
$X = \begin{pmatrix} 1 & x_1^{(1)} & \dots & x_D^{(1)} \\ 1 & x_1^{(2)} & \dots & x_D^{(2)} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_1^{(N)} & \dots & x_D^{(N)} \end{pmatrix}$	$w = \begin{pmatrix} w_0 \\ w_1 \\ \vdots \\ w_D \end{pmatrix}$	$y = \begin{pmatrix} y^{(1)} \\ y^{(2)} \\ \vdots \\ y^{(N)} \end{pmatrix}$

- **Salidas predichas:** $\hat{y} = Xw$
- **Residuos:** $r = (Xw - y)$
- **Coste L_2 :** $J(w) = \frac{1}{2} r^T r$
- **Gradiente:** $g(w) = X^T r$
- **Hessiano:** $X^T X$ ¿Definido positivo ya que J es convexa?

Algoritmo de Descenso de Gradiente

Es un algoritmo muy simple, no se usa en la práctica por problemas de divergencia.

Es muy sensible al escalado de los atributos.

Ecuación Normal

$$X^T \hat{\theta} = X^T y \Rightarrow \hat{\theta} = X^+ y \text{ (Pseudo-inversa de Moore-Penrose)}$$

En la práctica usar

```
from numpy import linalg
theta = linalg.inv(X.T @ X) @ (X.T @ y) # Mala idea
theta = linalg.pinv(X) @ y             # Mala idea
theta = linalg.lstsq(X, y)              # Mucho mejor así x3 o x4 velocidad
```

Descenso de Gradiente vs Ecuación Normal

Descenso de gradiente	Ecuación normal
Solución iterativa: $\theta_{k+1} := \theta_k - \alpha g(\theta_k)$ Se necesita elegir α	Solución analítica directa: $\hat{\theta} = (X^T X)^{-1} X^T y$ No es necesario elegir α

¹TPU: Tensor Procesor Unit, procesadores para redes neuronales

Descenso de gradiente	Ecuación normal
Funciona bien incluso si D es muy grande	Hay que invertir $X^T X$, que tiene dimensión $(D + 1)^2$, $O(D^3)$.
Algoritmo más general, válido con otras funciones de coste no cuadráticas	Si $D \gg 1000$ mejor descenso de gradiente
En el caso general, puede converger a un mínimo local	Convergencia global, siempre converge

Descenso de Gradiente Estocástico (SGD)

Es el algoritmo que suelen implementar las librerías.

- Ordena **aleatoriamente** las muestras, permite abandonar mínimos locales.
- Aplica descenso de gradiente muestra a muestra, o por paquetes de muestras (batches).
- Es sensible al escalado (Si hay atributos grandes y pequeños no funcionará correctamente).
- Hay que ajustar el factor de aprendizaje² (α).
- Muy útil en problemas muy grandes ($D \gg 1000$ y $n \gg 10000$).
- Con funciones de coste complejas, puede escapar de mínimos locales.

Escalado de Atributos

² α o factor de aprendizaje: **Añadir definición básica, ¿para qué sirve?**