Computer Science

15 de agosto de 2025

Índice general

Lista de Algoritmos

Capítulo 1

Machine Learning

1.1. Aprendizaje Supervisado

1.1.1. Clasificación binaria

Tenemos un dataset de pares clasificados tales que:

$$D = \{(x_1, y_1), (x_2, y_2) \dots (x_n, y_n)\}$$

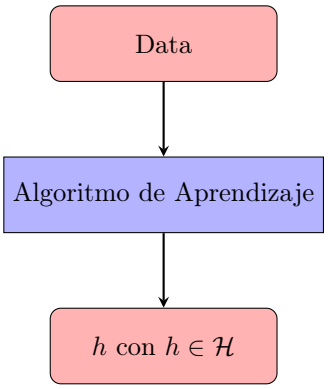
Donde:

$$y_i \in \{-1, +1\}$$

x_i puede ser cualquier cosa. Y en caso que $x_i \notin \mathbb{R}^d$, entonces se aplicará una transformación por medio de una función $\varphi : \mathbb{D} \to \mathbb{R}^d$, considerando que $x_i \in \mathbb{D}$ para poder tener una entrada $x_i' \in \mathbb{R}^d$.

Función de hipótesis

Para realizar las predicciones, necesitaremos una **función de hipótesis** h, la cual obtendremos tras usar un algoritmo de aprendizaje en un conjunto de funciones $\mathcal{H}$. Este conjunto de funciones $\mathcal{H}$ puede tener funciones de cualquier tipo. Nuestro propósito es encontrar una h de un dado $\mathcal{H}$ que se ajuste lo mejor posible a nuestro dataset.



Función de Pérdida

Nos ayuda a saber si nuestra predicción fue acertada, o no. Esta función la denotaremos de la siguiente manera:

$$L(g, a)$$

Donde g es nuestra predicción y a es el valor real.

Existen muchas funciones de pérdida para casos distintos, por ahora, enfoquémonos en las siguientes:

0 — 1 **Loss**

$$L(g, a) = \begin{cases} 1 & g \neq a \\ 0 & g = a \end{cases}$$

Squared Loss

$$L(g, a) = (g - a)^2$$

Linear Loss

$$L(g, a) = |g - a|$$

Asymmetric Loss

$$L(g, a) = \begin{cases} 1 & g = 1 \wedge a = -1 \\ 10 & g = -1 \wedge a = 1 \\ 0 & \text{else} \end{cases}$$

Error

Podemos medir que tan bien se desempeña nuestro modelo con los datos con los que lo entrenamos por medio del **error de entrenamiento** (*training error*):

$$\mathcal{E}_n(h) = \frac{1}{n} \sum_{i=1}^{n} L(h(x_i), y_i)$$

En donde n es la cantidad de datos que utilizamos para entrenar el modelo, L es la función de pérdida, h es nuestro modelo (la función de hipótesis), y x_i con y_i son un par ordenado de nuestro dataset.

De igual forma, puesto que normalmente se deja una parte de los datos separada para pruebas, el **error de pruebas** (*test error*) se puede calcular con:

$$\mathcal{E}(h) = \frac{1}{n}\sum_{i=1}^{n} L(h(x_i), y_i)$$

En donde n es la cantidad de datos que guardamos para probar nuestro modelo, L es la función de pérdida, h es nuestro modelo (la función de hipótesis), y x_i con y_i son un par ordenado de nuestro dataset. Es importante mencionar que los datos de prueba no deben ser iguales ni incluir datos que se hayan usado para entrenar el modelo. La idea es que sean datos que el modelo no haya visto aún.

1.1.2. Clasificador Lineal

En este caso tendremos un conjunto de funciones:

$$\mathcal{H} = \{h(x; \theta, \theta_0) \quad \text{s.t.} \quad \theta \in \mathbb{R}^d \quad \wedge \quad \theta_0 \in \mathbb{R}\}$$

Por lo que estaremos buscando las θ y θ_0 que mejor se ajusten a nuestro dataset.

También, consideraremos que:

$$h(x; \theta, \theta_0) = sign(\theta \cdot x + \theta_0) = \begin{cases} +1 & \theta \cdot x + \theta_0 > 0 \\ -1 & \text{else} \end{cases}$$

En otras palabras:

$$h : \mathbb{R}^d \to \{-1, +1\} \quad \text{s.t.} \quad h \in \mathcal{H}$$

Y ahora necesitamos un algoritmo que selecciones una buena h.

Algoritmo aleatorio de clasificación lineal (RLC)

Random Linear Classifier (RLC) consiste, como su nombre indica, a buscar de forma aleatoria y elegir el que mejor funciones.

No es necesario explicar mucho. El algoritmo prueba k veces con datos aleatorios y retorna aquellas (θ, θ_0) que poseen menor error.

Algoritmo del Perceptrón

Siguiendo con otro algoritmo de clasificación binaria, ahora el algoritmo del perceptrón. Igual que antes, es para un dataset $D = \{(x_1, y_1), \cdots (x_n, y_n)\}$ donde $x_i \in \mathbb{R}^d$ y $y \in \{-1, +1\}$.

Algorithm 1 *Random Linear Classifier*

procedure RLC($\mathcal{D} = \{(x_i, y_i)\}_{i=1}^{n}$, k)
  for $j = 1 \to k$ **do**
    $\theta_{(j)} \leftarrow \text{random}(\mathbb{R}^d)$ ▷ Vector aleatorio en $\mathbb{R}^d$
    $\theta_{0(j)} \leftarrow \text{random}(\mathbb{R})$ ▷ Escalar aleatorio
  end for
  $j^* \leftarrow \arg\min_{j \in \{1,\dots,k\}} \mathcal{E}_n(\theta_{(j)}, \theta_{0(j)})$ ▷ Selecciona el mejor clasificador
  return $(\theta_{(j^*)}, \theta_{0(j^*)})$ ▷ Retorna parámetros óptimos
end procedure

Algorithm 2 *Perceptron Algorithm*

procedure PERCEPTRON($\mathcal{D} = \{(x_i, y_i)\}_{i=1}^{n}$, k)
  $\theta \leftarrow [0, 0, \dots 0]$ ▷ Inicializa θ en $\mathbb{R}^d$
  $\theta_0 \leftarrow 0$ ▷ Inicializa θ_0 en $\mathbb{R}$
  for $t = 1 \to k$ **do**
    for $i = 1 \to n$ **do**
      if $y_{(i)} \cdot (\theta \cdot x_{(i)} + \theta_0) \leq 0$ **then**
        $\theta \leftarrow \theta + y_{(i)} \cdot x_{(i)}$
        $\theta_0 \leftarrow \theta_0 + y_{(0)}$
      end if
    end for
  end for
  return (θ, θ_0) ▷ Retorna parámetros óptimos
end procedure

Lo que hace el algoritmo del perceptrón consiste en variar los parámetros de acuerdo a los fallos de estos. Si falla, se actualiza.

Para hacer esto, utiliza internamente una función de pérdida en el condicional de la línea 4:

$$L((\theta \cdot x_{(i)} + \theta_0), y_{(i)}) = \begin{cases} (\theta \cdot x_{(i)} + \theta_0) > 0 & \wedge & y_{(i)} = +1 & \text{false} \\ (\theta \cdot x_{(i)} + \theta_0) < 0 & \wedge & y_{(i)} = -1 & \text{false} \\ \text{else} & & & \text{true} \end{cases}$$

Capítulo 2

Interbloqueo

2.1. Demostración de Prevención de Interbloqueo mediante un Algoritmo de Asignación Atómica

2.1.1. Planteamiento

Sea R el conjunto de recursos de un sistema de computación multiprocesado y multiprogramado, y sea H el conjunto de procesos (o hilos) que pueden ejecutarse en el sistema.

Definimos un algoritmo A con las siguientes propiedades:

1. Todo proceso debe pasar por A para solicitar recursos.
2. A verifica, de forma atómica, si todos los recursos requeridos por el proceso están disponibles.
3. Si están disponibles, se le conceden todos a la vez.
4. Si no están disponibles, el proceso espera sin obtener ningún recurso.
5. Si un proceso necesita más recursos después, debe liberar todos los que ya posee y volver a solicitarlos desde cero a través de A.
6. A actúa como un **monitor**: solo un proceso puede estar dentro de él a la vez.

Queremos demostrar que con este algoritmo es **imposible que ocurra un interbloqueo (deadlock)**.

2.1.2. Demostración por inducción sobre el número de procesos

Caso base: $P(1)$

Supongamos que hay un único proceso p_1 que solicita un conjunto de recursos.

Como es el único proceso en el sistema, todos los recursos están disponibles. Entonces, el algoritmo A le asigna todos los recursos necesarios.

Conclusión: No hay espera ni bloqueo. El sistema no presenta interbloqueo.

Hipótesis inductiva: $P(k)$

Supongamos que con k procesos en ejecución, utilizando el algoritmo A, no se produce interbloqueo.

Paso inductivo: $P(k+1)$

Agregamos un proceso más: p_{k+1}. Este entra al algoritmo A para solicitar un conjunto de recursos.

- Si todos los recursos que necesita están disponibles, se le conceden de inmediato. No hay bloqueo.
- Si alguno no está disponible, no se le concede ninguno, y espera fuera del monitor sin retener recursos.

Además, si otro proceso p_c quiere recursos adicionales, debe liberarlos todos antes de hacer una nueva solicitud, lo cual evita la retención y espera.

Esto impide la formación de cadenas de espera circulares, condición necesaria para que ocurra un interbloqueo.

Conclusión: Al agregar un proceso más, el sistema sigue libre de interbloqueo.

Conclusión final

Por el principio de inducción matemática, se concluye que:

Ningún conjunto finito de procesos que usen el algoritmo A puede entrar en interbloqueo.