Universidad de Alcalá
Escuela Politécnica Superior

Grado en Ingeniería Informática

Trabajo Fin de Grado

Estudio de técnicas de visión e inteligencia artificial aplicadas a un caso práctico

Autor: Pablo García García

Tutor: Adrián Domínguez Díaz

2024

UNIVERSIDAD DE ALCALÁ
ESCUELA POLITÉCNICA SUPERIOR

Grado en Ingeniería Informática

Trabajo Fin de grado

Estudio de técnicas de visión e inteligencia artificial aplicadas a un caso práctico

Autor: Pablo García García

Tutor: Adrián Domínguez Díaz

Tribunal:

Presidente:

Vocal 1º:

Vocal 2º:

Fecha de depósito:

"Computer Science is no more about computers than astronomy is about telescopes."
Edsger W. Dijkstra

Resumen

Palabras clave:

Abstract

Keywords:

Índice general

Índice de figuras

Capítulo 1

Introducción

Capítulo 2

Fundamentos de la Inteligencia Artificial y sus herramientas

En este capítulo se abordará una breve introducción al campo de la Inteligencia Artificial, comenzando desde su historia a lo largo del tiempo para contextualizar, pasando por entender los fundamentos de lo que busca lograr, y comentando alguna herramienta matemática que será de utilidad y que en ciertos casos causa confusiones.

2.1. Breve historia de la Inteligencia Artificial

2.2. ¿Qué es un modelo?

2.3. Tipos de aprendizaje y problemas

Una vez entendida cuál es la idea de un modelo, es momento de discutir cómo se hace posible que este desempeñe correctamente su tarea. Esto se logra mediante un algoritmo de aprendizaje o entrenamiento que ajusta de manera óptima los parámetros del modelo. Normalmente se dispone de dos tipos de aprendizaje, **aprendizaje supervisado** y **aprendizaje no supervisado**[1].

En los algoritmos de aprendizaje supervisados, se dispone de un conjunto de datos o *dataset*, que contiene los valores de salida deseados para diferentes valores de entrada, normalmente recogiendo situaciones del pasado para poder extrapolar este conocimiento a situaciones del futuro. Los principales problemas que utilizan algoritmos de aprendizaje supervisado son los problemas de clasificación y de regresión. En los **problemas de clasificación**, se dispone de una serie de clases $C_1, C_2, \ldots, C_n$, y para una serie de valores de entrada $x_1, x_2, \ldots, x_m$, debe decidirse a qué clase pertenece dicha entrada. Un ejemplo sería decidir si un paciente va a sufrir un cierto tipo de cáncer dada su edad, peso, y otras constantes vitales. Algunos de los modelos más populares para llevar a cabo este tipo de tareas son árboles de decisión, máquinas de soporte vectorial, Naïve Bayes, $k-$vecinos, y redes neuronales; siendo estas últimas objeto de estudio en este trabajo. Otro tipo de problema popular a la hora de disponer de datos etiquetados, son los **problemas de regresión**, que se diferencia principalmente de la clasificación en que en este caso, los valores no son clases (valores discretos) sino valores continuos. Para resolver este tipo de problemas se suelen utilizar regresiones lineales y no lineales (exponencial, polinómica, etc). Un ejemplo de un problema de regresión sería predecir las horas que dormirá una persona dada su edad, horas trabajadas en el día, horas de recreo en el día, etc.

Por otro lado, los algoritmos de aprendizaje no supervisado no reciben los valores de salida esperados para una cierta observación (justo al contrario que en el caso supervisado), pues será trabajo del algoritmo encontrar relaciones y patrones entre los datos proporcionados. En este tipo de aprendizaje también se trata el problema de clasificación, sin embargo, es más común llamarlo **clústering** o **segmentación**, pues

a priori no se conoce el número de clases y cuáles son, es el algoritmo el que deberá encontrar relaciones entre los datos para determinar esto. Algoritmos populares para realizar esta tarea son $k-$medias (y su variante $k-$medianas), clusterización jerárquica aglomerativa, modelos de mixtura gaussianos, y DBSCAN. Un ejemplo sencillo de este problema es detectar las diferentes regiones y objetos representados en una imagen, pues inicialmente no se conoce el número de regiones u objetos, y deben detectarse todas, asignando cada píxel de la imagen a cada una de ellas.

2.4. Convolución y correlación cruzada

Una de las operaciones matemáticas más conocidas y que es más usada al trabajar con señales e imágenes es la llamada convolución, denotada por $*$, y dadas las funciones $f(t)$ y $g(t)$, su convolución se define de la siguiente manera.

$$(f * g)(t) = \int_{-\infty}^{\infty} f(\tau)g(t-\tau)\,d\tau$$

En este caso se está asumiendo que el dominio de $f(\tau)g(t-\tau)$ es $\mathbb{R}$, lo que permite integrar sobre todo $\mathbb{R}$, de lo contrario, se modifica la definición para integrar solo sobre un intervalo $[a, b]$. En general, esta operación crea una nueva función a partir de otras dos, que indica cómo interactúan entre sí, y que permite aplicar filtros a señales e imágenes. Como se acaba de comentar para el caso de las imágenes, se puede tratar con señales que no dependan únicamente de una variable, pues estas se representan como una función de dos variables $f(u, v)$. En este caso, la convolución queda definida de la siguiente manera.

$$(f * g)(u, v) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} f(\xi, \eta)g(u-\xi, v-\eta)\,d\xi\,d\eta$$

Partiendo de las primera definición mostrada, se pueden demostrar algunas propiedades útiles que cumple la convolución[2]:

- Conmutativa: $f * g = g * f$
- Asociativa: $f * (g * h) = (f * g) * h$
- Distributiva: $f * (g + h) = (f * g) + (f * h)$
- Derivada: $\frac{d}{dt}(f * g) = \frac{df}{dt} * g = \frac{dg}{dt} * f$
- Relación con las transformadas de Laplace y Fourier: $\mathscr{L}\{f * g\} = \mathscr{L}\{f\} \cdot \mathscr{L}\{g\}$, o lo que suele ser más útil, $f * g = \mathscr{L}^{-1}\{\mathscr{L}\{f\} \cdot \mathscr{L}\{g\}\}$, de forma que se puede calcular la convolución en tiempo $\mathcal{O}(n \log(n))$ con el algoritmo FFT[3].

Si bien en las definiciones previas se ha tomado la integral y tanto $\mathbb{R}$ como $\mathbb{R}^2$ como dominios continuos sobre los que calcular la convolución, no se debe olvidar que las imágenes no dejan de ser matrices o funciones de dos variables con un dominio discreto, por lo que se debe presentar una definición adecuada a este caso[4].

$$(f * g)(u, v) = \sum_{i=-k}^{k}\sum_{j=-k}^{k} g(i, j)f(u-i, v-j)$$

En la expresión anterior, a la función $g(u, v)$ se le llama filtro o *kernel* de convolución. A continuación se muestra un ejemplo de cómo calcular una convolución.

$$\begin{pmatrix} \boxed{\begin{matrix}1&2&3\\5&6&7\\9&8&7\end{matrix}} & 4 & 5 \\ 5 & 4 & 3 & 2 & 1 \\ 1 & 2 & 3 & 4 & 5 \end{pmatrix} * \begin{pmatrix} \boxed{\begin{matrix}1&0&0\\0&1&0\\0&0&1\end{matrix}} \end{pmatrix} = \begin{pmatrix} 14 & 15 & 16 \\ 16 & 15 & 14 \\ 16 & 15 & 14 \end{pmatrix}$$

$$1 \cdot 1 + 2 \cdot 0 + 3 \cdot 0 + 5 \cdot 0 + 6 \cdot 1 + 7 \cdot 0 + 9 \cdot 0 + 8 \cdot 0 + 7 \cdot 1 = 14$$
$$2 \cdot 1 + 3 \cdot 0 + 4 \cdot 0 + 6 \cdot 0 + 7 \cdot 1 + 8 \cdot 0 + 8 \cdot 0 + 7 \cdot 0 + 6 \cdot 1 = 15$$
$$\vdots$$
$$7 \cdot 1 + 6 \cdot 0 + 5 \cdot 0 + 3 \cdot 0 + 2 \cdot 1 + 1 \cdot 0 + 3 \cdot 0 + 4 \cdot 0 + 5 \cdot 1 = 14$$

Aplicando diferentes kernels de convolución a una imagen se pueden extraer diferentes tipos de características de una imagen, como por ejemplo bordes. El filtro Sobel es capaz de hacer esto con los kernels que se muestran a continuación, pues se comportan como aproximaciones de las derivadas parciales de la imagen en un punto teniendo en cuenta los píxeles cercanos[5].

$$G_u = \frac{\partial f(u,v)}{\partial u} \approx \begin{pmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{pmatrix} \qquad G_v = \frac{\partial f(u,v)}{\partial v} \approx \begin{pmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{pmatrix}$$

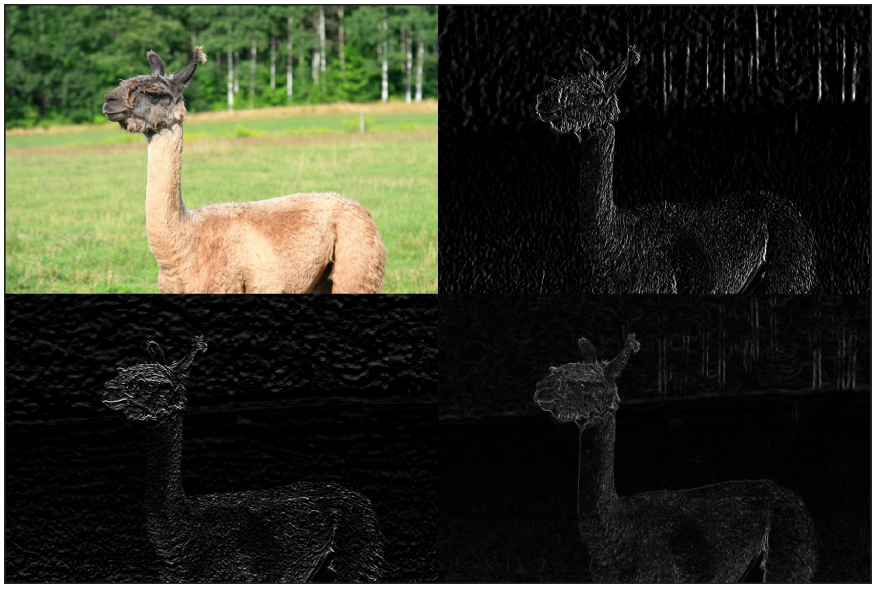

Figura 2.1: Detección de bordes aplicando los kernel Sobel

A continuación se va a calcular la convolución de la matriz del ejemplo anterior con G_u. Para comprobar los cálculos, se puede realizar esto con la función `convn` de MATLAB. Al realizar los cálculos a mano tal y como se ha mostrado en el ejemplo anterior, se obtiene como resultado la matriz A, mientras que MATLAB devuelve la matriz B como resultado de `convn(X,` G_u`, 'valid')`. ¿Qué acaba de suceder? ¿Está mal codificada la función de MATLAB? ¿Está mal realizado el ejemplo?

$$A = \begin{pmatrix} 4 & 4 & 4 \\ -4 & -4 & -4 \\ -4 & -4 & -4 \end{pmatrix} \qquad B = \begin{pmatrix} -4 & -4 & -4 \\ 4 & 4 & 4 \\ 4 & 4 & 4 \end{pmatrix}$$

La respuesta a estas preguntas se podría resumir en que se ha realizado una "pequeña trampa" a la hora de calcular la convolución manualmente en el ejemplo, ya que no se ha aplicado correctamente la definición dada. ¿Qué sentido tiene hacer esto? En visión artificial y tratamiento de imágenes, muchos autores y librerías llaman convolución a la operación que se ha mostrado en el primer ejemplo, cuando en realidad no lo es y trae lugar a confusión. Dicha operación se llama correlación cruzada, denotada por $(\star)$, y que es muy similar a la convolución, pues su principal diferencia es que en la convolución "real", el kernel se rota 180 grados antes de calcular la convolución "falsa" o correlación cruzada, es decir, $X * Y = X \star (R_{180} \cdot Y)$, donde R_{180} es la matriz de rotación de 180 grados. La correlación cruzada de dos imágenes (matrices) se define de la siguiente manera[4].

$$(f \star g)(u, v) = \sum_{i=-k}^{k} \sum_{j=-k}^{k} g(i, j) f(u + i, v + j)$$

Esta operación sí es con la que realmente se aplican los filtros a las imágenes y con la que se trabaja en general en el campo de la visión artificial. Es importante ver que ahora, al contrario que con la convolución, $f \star g \neq g \star f$. Algunas librerías de visión artificial tratan a la correlación cruzada como convolución debido al frecuente uso que tiene una sobre la otra y la forma similar que tienen de calcularse. Un ejemplo es OpenCV en la documentación de su función `filter2D`, donde se comenta que aplica una convolución cuando realmente aplica la correlación cruzada[6]. Finalmente, como se verá en próximos capítulos, las famosas redes neuronales convolucionales, no aplican convoluciones sino correlaciones cruzadas.

Capítulo 3

Aprendizaje automático

3.1. Perceptrón

Ya en el año 1958, el psicólogo Frank Rosenblatt propuso un modelo llamado perceptrón el cual estaba basado en el comportamiento y funcionamiento de las neuronas de un humano, y que podía aprender ponderando cada coeficiente de entrada a la neurona[7]. Hoy en día, tal y como se mostrará en esta sección, el perceptrón es la unidad fundamental de muchos modelos de *machine learing* y *deep learning*.

Como se verá durante esta sección, este modelo ayuda a solucionar problemas de clasificación supervisada. Se dispone de una serie de valores de entrada $x_1, x_2, \ldots, x_n$ y se tiene una serie de valores de salida $y_1, y_2, \ldots, y_m$ que representan a qué clase pertenece la entrada (2^m clases posibles). Esto se consigue mediante la ayuda de sus parámetros, que son una serie de pesos $w_1, w_2, \ldots, w_n$ y un sesgo o *bias* b; y sus hiperparámetros, entre los que se encuentra una función f de activación.

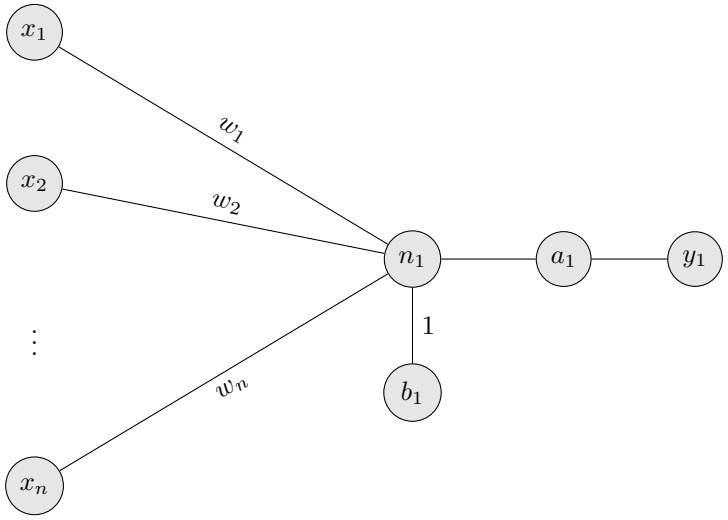

Figura 3.1: Arquitectura de un perceptrón

En la Figura 3.1 se muestra la arquitectura del caso más simple de un perceptrón. Se tienen n entradas y una única salida. La primera parte del diagrama representa que tal y como decía Rosenblatt, cada valor de entrada debe multiplicarse por un cierto peso, de tal forma que si se representa esto en función de sus valores en un instante k, lo que se computa en el nodo n_1 es la siguiente operación.

$$n_1(k) = b_1(k) + \sum_{i=1}^{n} x_i(k) w_i(k)$$

Una vez se ha realizado este cálculo, el valor pasa por una función de activación en el nodo a_1, pues esta arquitectura es común utilizarla para clasificar una entrada y es muy útil obtener una salida binaria donde se active únicamente la salida que represente la clase a la que pertenece la entrada dada. Aunque existen diferentes funciones de activación para las neuronas, al trabajar con un perceptrón, la función de activación por excelencia es la función escalón de Heaviside, donde $\mathcal{U} : \mathbb{R} \longrightarrow \{0, 1\}$ y su expresión analítica es

$$\mathcal{U}(x) = \begin{cases} 0 & \text{si} \quad x < 0 \\ 1 & \text{si} \quad x \geq 0 \end{cases} .$$

Combinando ambas expresiones, se puede resumir en que la salida del perceptrón es equivalente a la siguiente ecuación:

$$y_1(k) = \begin{cases} 0 & \text{si} \quad b_1(k) + \sum_{i=1}^{n} x_i(k) w_i(k) < 0 \\ 1 & \text{si} \quad b_1(k) + \sum_{i=1}^{n} x_i(k) w_i(k) \geq 0 \end{cases}$$

Para dar un ejemplo claro de cómo funciona el perceptrón, se pueden tomar una serie de observaciones que tengan dos valores de entrada y uno de salida. Además, se supondrá que existen dos clases. Esto a fin de cuentas es asignar un valor de 0 o 1 a cada punto de $\mathbb{R}^2$ tal y como se describe en la Figura 3.2.

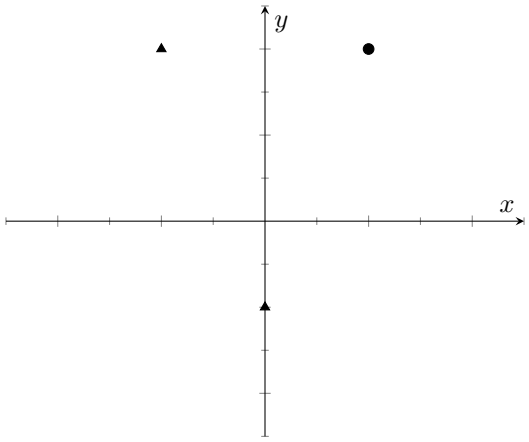

Figura 3.2: Puntos etiquetados en $\mathbb{R}^2$

Una solución rápida sería trazar una recta $r : ax + by + c = 0$ que separe $\mathbb{R}^2$ en dos regiones, de forma que todo punto que pertenezca a una región pertenece entonces a una misma clase, tal y como se observa en la Figura 3.3. Esta recta suele llamarse *decision boundary* o frontera de decisión. El problema entonces es hallar la recta r, pero se cumple que para este ejemplo es de la forma $w_1 x + w_2 y + b = 0$, siendo el problema encontrar los parámetros adecuados del modelo. La idea puede extrapolarse a diferente tamaño de entrada tomando un hiperplano de la forma $\mathbf{w}^t \mathbf{x} + b = 0$.

Las preguntas a resolver ahora son, ¿existen siempre dichos parámetros? ¿Cómo pueden hallarse? El propio Minsky se hizo estas preguntas en [8] y se dio cuenta de que dichos parámetros sí pueden hallarse en un número finito de pasos, siempre y cuando los puntos sean linealmente separables. Un ejemplo que no es linealmente separable es el de la función XOR tal y como se muestra en la Tabla 3.1 y Figura 3.4, pues no existe una recta r que separe $\mathbb{R}^2$ en dos regiones de tal forma que cada región contenga puntos de una única clase, sería necesaria una frontera de decisión no lineal.

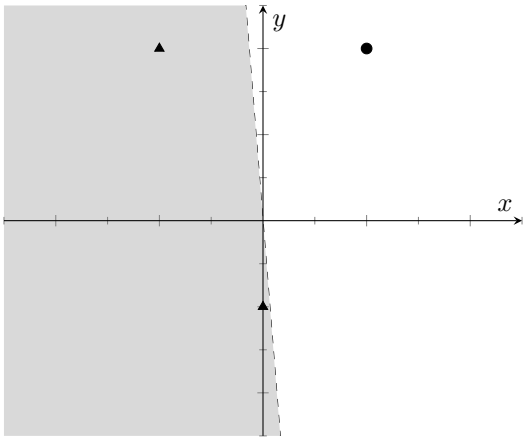

Figura 3.3: Puntos separados en $\mathbb{R}^2$

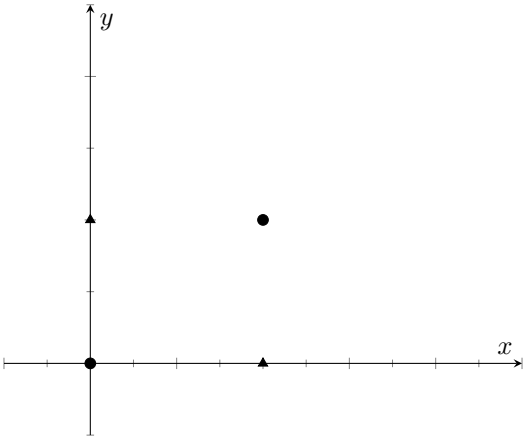

Figura 3.4: Valores de $x \oplus y$ en $\mathbb{R}^2$

x	y	$x \oplus y$
0	0	0
0	1	1
1	0	1
1	1	0

Tabla 3.1: Función XOR

En cuanto a la pregunta de cómo hallar los parámetros, se consideran las siguientes ecuaciones[9], donde $\mathbf{w}$ es el vector de pesos, t el valor esperado, y a la salida del perceptrón y se aplica el Algoritmo 3.1 para obtener los parámetros óptimos. En dicho algoritmo se supondrá que existe una matriz X de n filas que contiene los diferentes $\mathbf{x}$.

$$\begin{aligned} \mathbf{w}(k+1) &= \mathbf{w}(k) + e(k)\mathbf{x}(k) \\ b(k+1) &= b(k) + e(k) \\ e(k) &= t(k) - a(k) \\ a(k) &= \mathcal{U}(\mathbf{w}^t(k)\mathbf{x}(k)) \end{aligned} \tag{3.1}$$

A cada una de las iteraciones que realiza el bucle exterior se les denomina épocas o *epoch*, que consiste en realizar el proceso de entrenamiento sobre todo el conjunto de datos. En este caso se está suponiendo que no va a recibir casos que no sean linealmente separables, pero de lo contrario se puede añadir un contador

Algoritmo 3.1: Regla de aprendizaje del perceptrón

Datos: $X, \mathbf{t}$
Resultado: $\mathbf{w}, b$
$b \leftarrow 0$
$\mathbf{w} \leftarrow$ `random`
$k \leftarrow 0$
repetir
  `acabar` $\leftarrow$ `true`
  para $i \leftarrow k$ **hasta** $k + n - 1$ **hacer**
    $e(i) \leftarrow t(i) - a(i)$
    $\mathbf{w}(i+1) \leftarrow \mathbf{w}(i) + e(i)\mathbf{x}(i \pmod n)$
    $b(i+1) \leftarrow b(i) + e(i)$
    `acabar` $\leftarrow$ `acabar` $\wedge\, e(i) == 0$
  fin
  $k \leftarrow k + n - 1$
mientras $\neg$`acabar`

`max_epochs` y fijar un número máximo para no caer en un bucle infinito. No sería tarea fácil determinar dicho valor, pues aunque el algoritmo converge en los casos previamente explicados, no en todos lo hace de manera rápida. A continuación se muestra cómo obtener una solución para el caso de la Figura 3.2 aplicando el Algoritmo 3.1.

$$X = \begin{pmatrix} 1 & -1 & 0 \\ 2 & 2 & -1 \end{pmatrix} \quad \mathbf{t} = \begin{pmatrix} 1 & 0 & 0 \end{pmatrix} \quad \mathbf{w}(0) = \begin{pmatrix} 1 \\ 1 \end{pmatrix} \quad b(0) = 0$$

0.
- $a(0) = \mathcal{U}(\mathbf{w}^t(0)\mathbf{x}(0)) = \mathcal{U}\left(\begin{pmatrix} 1 & 1 \end{pmatrix}\begin{pmatrix} 1 \\ 2 \end{pmatrix} + 0\right) = 1$
- $e(0) = t(0) - a(0) = 1 - 1 = 0$
- $\mathbf{w}(1) = \mathbf{w}(0)$
- $b(1) = b(0)$

1.
- $a(1) = \mathcal{U}(\mathbf{w}^t(1)\mathbf{x}(1)) = \mathcal{U}\left(\begin{pmatrix} 1 & 1 \end{pmatrix}\begin{pmatrix} -1 \\ 2 \end{pmatrix} + 0\right) = 1$
- $e(1) = t(1) - a(1) = 0 - 1 = -1$
- $\mathbf{w}(2) = \mathbf{w}(1) + e(1)\mathbf{x}(1) = \begin{pmatrix} 1 \\ 1 \end{pmatrix} - \begin{pmatrix} -1 \\ 2 \end{pmatrix} = \begin{pmatrix} 2 \\ -1 \end{pmatrix}$
- $b(2) = b(1) + e(1) = -1$

2.
- $a(2) = \mathcal{U}(\mathbf{w}^t(2)\mathbf{x}(2)) = \mathcal{U}\left(\begin{pmatrix} 2 & -1 \end{pmatrix}\begin{pmatrix} 0 \\ -1 \end{pmatrix} - 1\right) = 1$
- $e(2) = t(2) - a(2) = 0 - 1 = -1$
- $\mathbf{w}(3) = \mathbf{w}(2) + e(2)\mathbf{x}(2) = \begin{pmatrix} 2 \\ -1 \end{pmatrix} - \begin{pmatrix} 0 \\ -1 \end{pmatrix} = \begin{pmatrix} 2 \\ 0 \end{pmatrix}$
- $b(3) = b(2) + e(2) = -2$
- `acabar` $=$ `true` $\wedge$ `false` $\wedge$ `false` $=$ `false`

3.
- $a(3) = \mathcal{U}(\mathbf{w}^t(3)\mathbf{x}(3 \pmod 3)) = \mathcal{U}\left(\begin{pmatrix} 2 & 0 \end{pmatrix}\begin{pmatrix} 1 \\ 2 \end{pmatrix} - 2\right) = 1$
- $e(3) = t(3) - a(3) = 1 - 1 = 0$
- $\mathbf{w}(4) = \mathbf{w}(3)$
- $b(4) = b(3)$

4.
- $a(4) = \mathcal{U}(\mathbf{w}^t(4)\mathbf{x}(4 \pmod 3)) = \mathcal{U}\left(\begin{pmatrix}2 & 0\end{pmatrix}\begin{pmatrix}-1\\2\end{pmatrix} - 2\right) = 0$
- $e(4) = t(4) - a(4) = 0 - 0 = 0$
- $\mathbf{w}(5) = \mathbf{w}(4)$
- $b(5) = b(4)$

5.
- $a(5) = \mathcal{U}(\mathbf{w}^t(5)\mathbf{x}(5 \pmod 3)) = \mathcal{U}\left(\begin{pmatrix}2 & 0\end{pmatrix}\begin{pmatrix}0\\-1\end{pmatrix} - 2\right) = 0$
- $e(5) = t(5) - a(5) = 0 - 0 = 0$
- $\mathbf{w}(6) = \mathbf{w}(5)$
- $b(6) = b(5)$
- $\texttt{acabar} = \texttt{true} \wedge \texttt{true} \wedge \texttt{true} = \texttt{true}$

La ejecución del algoritmo finaliza tras dos épocas, donde en la primera de ellas va ajustando los pesos y el sesgo de manera adecuada, y en la segunda verifica que todas las observaciones han sido clasificadas de manera correcta. La frontera de decisión obtenida es la recta $r : 2x - 2 = 0$, que es una recta vertical. Una observación a realizar es que $\mathbf{x}(1) \in r$, por tanto ¿a qué clase pertenece? Esto depende de la función de transferencia empleada, al utilizar $\mathcal{U}$ la clasificación es correcta, pero al cambiarla por otra, podría no serlo y necesitaría de más épocas para realizar correctamente la clasificación.

La última cuestión que queda por tratar respecto al perceptrón es el porqué se verifica que en el caso de que los puntos dados sean linealmente separables, en un número finito de pasos, el Algoritmo 3.1 terminará su ejecución y con el $\mathbf{w}$ óptimo, tal y como se enunciaba en el **Teorema de Convergencia del Perceptrón**. A continuación se realiza su demostración basándose en la que se encuentra en [9] pero de forma más clara y simple. Para comenzar con la demostración será necesario definir una serie de elementos, comenzando por $\Omega(k)$ y $\mathbf{z}(k)$. Se entiende que se están concatenando vectores, no definiendo "vectores dentro de vectores".

$$\Omega(k) = \begin{pmatrix}\mathbf{w}(k)\\b(k)\end{pmatrix} \quad \mathbf{z}(k) = \begin{pmatrix}\mathbf{x}(k)\\1\end{pmatrix} \quad \Omega(0) = \begin{pmatrix}0\\0\\\vdots\\0\end{pmatrix}$$

Con estos elementos y el cálculo de $a(k)$ en la Ecuación (3.1) es fácil ver que $n(k) = \Omega^t(k)\mathbf{z}(k)$. Recordando los posibles valores de $t(k)$, se deseaba que si dicho valor era 1, entonces $n(k) \geq 0$; y en caso de que valiese 0 entonces se deseaba tener $n(k) < 0$, en resumen, que $t(k) - a(k) = 0$. Otra forma de ver esto es afirmar que en el caso en que $t(k) \neq a(k)$, entonces $\Omega(k)$ debe actualizarse de acuerdo a la Ecuación (3.1). De esta forma $\Omega(k+1) = \Omega(k) + e(k)\mathbf{z}(k)$. Ahora se considerará un vector Ω^* de forma que

$$\forall k \exists \Omega^* \ \mathcal{U}({\Omega^*}^t \mathbf{z}(k)) = t(k),$$

es decir, Ω^* es el vector de pesos óptimo. Además, por comodidad se normalizarán todas las distancias del problema, de manera que $\|\Omega^*\| = 1$ y $\|\mathbf{z}(k)\| \leq 1$. El último elemento a considerar será δ, que será definido como

$$\delta = \text{mín}\{{\Omega^*}^t \mathbf{z}(i)\},$$

tomando además que $\delta > 0$ pues otra manera de definirlo es la distancia al punto más cercano a la frontera de decisión óptima. Con estos elementos se puede comenzar la demostración. Como la regla de actualización es $\Omega(k+1) = \Omega(k) + e(k)\mathbf{z}(k)$, al vector de pesos en un determinado instante (clasificación fallida) se le suma o resta $\mathbf{z}(k)$ y a priori no se sabe cuántas veces se va a repetir esto, por lo que la idea de la demostración será ver si la norma del vector de pesos tiene una cota superior e inferior, es decir, se para de sumar o restar otros vectores $\mathbf{z}(i)$, de manera que el algoritmo terminaría. Para obtener esto basta con comparar el comportamiento de $\Omega^t(k)\Omega^*$ frente a $\Omega^t(k)\Omega(k)$ (es decir, $\|\Omega\|^2$).

Con el primero de los términos, al tratar de corregir un error se verifica que

$$\Omega^t(k+1)\Omega^* = (\Omega(k) + e(k)\mathbf{z}(k))^t\Omega^* = \Omega^t(k)\Omega^* + e(k)\Omega^{*^t}\mathbf{z}(k).$$

Además, por la manera en la que se ha definido δ, el segundo sumando pertenece al intervalo $(-\infty, -\delta] \cup [\delta, \infty)$, pudiendo deducir la siguiente desigualdad, por lo que en una actualización el término sólo varía por lo menos en δ unidades.

$$\Omega^t(k+1)\Omega^* \geq \Omega^t(k)\Omega^* + \delta \tag{3.2}$$

De la misma manera que se ha analizado el comportamiento de $\Omega^t(k)\Omega^*$ se procede con la actualización de $\Omega^t(k)\Omega(k)$.

$$\begin{aligned}\Omega^t(k+1)\Omega(k+1) &= (\Omega(k) + e(k)\mathbf{z}(k))^t(\Omega(k) + e(k)\mathbf{z}(k)) \\ &= \Omega^2(k) + (e(k)\mathbf{z}(k))^2 + 2e(k)\Omega^t(k)\mathbf{z}(k) \\ &= \Omega^t(k)\Omega(k) + e^2(k)\mathbf{z}^t(k)\mathbf{z}(k) + 2e(k)\Omega^t(k)\mathbf{z}(k) \\ &= \Omega^t(k)\Omega(k) + e^2(k)\mathbf{z}^t(k)\mathbf{z}(k) + 2e(k)n(k)\end{aligned}$$

En el segundo sumando se tiene que siempre será menor o igual que 1, pues por definición $0 \leq \mathbf{z}^t(k)\mathbf{z}(k) = \|\mathbf{z}(k)\|^2 \leq 1$. Además, el tercer sumando siempre será cero o negativo, pues en todos los casos posibles en los que $e(k) \neq 0$ se cumple que $e(k)n(k) < 0$:

- Si $t(k) = 1 \wedge n(k) < 0$, entonces $a(k) = 0 \wedge e(k) > 0$
- Si $t(k) = 0 \wedge n(k) \geq 0$, entonces $a(k) = 1 \wedge e(k) < 0$

De esta situación se puede deducir la siguiente desigualdad, por lo que en una actualización el término sólo varía en como máximo una unidad.

$$\Omega^t(k+1)\Omega(k+1) \leq \Omega^t(k)\Omega(k) + 1 \tag{3.3}$$

Una vez se ha observado cómo varían estos términos al actualizarlos, se puede observar qué pasaría con ellos al hacer m actualizaciones. Con el resultado obtenido en las Ecuaciones (3.2) y (3.3) se pueden deducir las desigualdades $\delta m \leq \Omega^t(m)\Omega^*$ y $\Omega^t(m)\Omega(m) \leq m$.

Ahora al aplicar la desigualdad de Cauchy–Schwarz[1], el valor de $\|\Omega^*\|$, y la propiedad transitiva, se obtiene una cota superior y otra inferior (ambas recuadradas) para $\|\Omega^t(m)\|$, lo que demuestra que el número de actualizaciones es finito y que por tanto el algoritmo converge.

$$\begin{aligned}\boxed{\delta m} &\leq \Omega^t(m)\Omega^* = \|\Omega^t(m)\Omega^*\| \\ &\leq \|\Omega^t(m)\|\|\Omega^*\| = \|\Omega^t(m)\| = \sqrt{\Omega^t(m)\Omega(m)} \\ &\leq \boxed{\sqrt{m}}\end{aligned}$$

$$\delta m \leq \|\Omega(m)\| \leq \sqrt{m} \implies m \leq \frac{1}{\delta^2} \qquad \square$$

Esta última desigualdad muestra cómo existe una relación entre el número de iteraciones del algoritmo y la distancia de los datos de entrenamiento a la frontera de decisión óptima (depende únicamente de esto). Cuanto más cerca estén, mayor será el número de iteraciones necesarias.

[1] Esta desigualdad afirma que $\|\mathbf{uv}\| \leq \|\mathbf{u}\|\|\mathbf{v}\|$.

Bibliografía

[1] R. Szeliski, *Computer Vision: Algorithms and Applications, 2nd ed.* Springer International Publishing, 2022, ISBN: 978-3-030-34371-2. DOI: 10.1007/978-3-030-34372-9. dirección: https://szeliski.org/Book/.

[2] López y H. I, "Método alternativo para calcular la convolución de señales en tiempo continuo," 2009.

[3] A. Lucía, "FFT: Transformada Rápida de Fourier,"

[4] I. Goodfellow, Y. Bengio y A. Courville, *Deep Learning*. MIT Press, 2016, págs. 326-330. dirección: http://www.deeplearningbook.org.

[5] W. Gao, L. Yang, X. Zhang y H. Liu, "An improved Sobel edge detection," *Proceedings - 2010 3rd IEEE International Conference on Computer Science and Information Technology, ICCSIT 2010*, vol. 5, págs. 67-71, 2010. DOI: 10.1109/ICCSIT.2010.5563693.

[6] *OpenCV: Image Filtering*. dirección: https://docs.opencv.org/4.x/d4/d86/group__imgproc__filter.html.

[7] A. Abeliuk y C. Gutiérrez, *Historia y evolución de la inteligencia artificial*.

[8] M. Minsky y S. A. Papert, *Perceptrons: An Introduction to Computational Geometry*. The MIT Press, ene. de 2017, ISBN: 9780262343930. DOI: 10.7551/MITPRESS/11301.001.0001.

[9] H. Demuth y B. D. Jesús, *Neural Network Design 2nd Edition*. dirección: https://hagan.okstate.edu/NNDesign.pdf.

Universidad de Alcalá
Escuela Politécnica Superior

ESCUELA POLITECNICA
SUPERIOR

Universidad
de Alcalá