

MÉTODOS COMPUTACIONALES

SEGUNDO TRABAJO PRÁCTICO

PRIMER SEMESTRE 2024

Introducción

La neumonía es una infección del sistema respiratorio que genera inflamación de los pulmones. Para su diagnóstico, los médicos se basan en los síntomas presentados y radiografías de tórax del paciente.



Figura 1: En la figura se observan radiografías de tórax en pacientes **sin** neumonía, **con** neumonía bacteriana y **con** neumonía viral.

Con esta información, el médico puede determinar con suficiente seguridad el diagnóstico del paciente y realizar el tratamiento correspondiente. El objetivo de este trabajo práctico es desarrollar un algoritmo utilizando los contenidos vistos en la materia, que permita diagnosticar pacientes con neumonía en función de radiografías de torax. Plantearemos la solución como un problema de regresión logística.

Método

El conjunto de datos a utilizar se puede descargar del siguiente enlace: https://data.mendeley.com/public-files/datasets/rscbjbr9sj/files/f12eaf6d-6023-432f-acc9-80c9d7393433/file_downloaded. Consiste en 5840 radiografías de Torax, cuenta con casos de pacientes sanos y pacientes con Neumonía

Les proponemos pensar el problema de la siguiente manera. Tenemos un conjunto de imágenes $\mathbf{i}_1, \dots, \mathbf{i}_N$ y sus diagnósticos asociados d_1, \dots, d_N . Cada \mathbf{i}_i es una matriz de píxeles y cada píxel es un número entre 0 y 255 que indica la tonalidad desde negro hasta blanco respectivamente. Los números d_i son el valor binario 0 ó 1 que indica si el paciente está sano o tiene neumonía. Nos gustaría encontrar alguna función f , que tome una imagen y nos devuelva un diagnóstico. Ahora bien, encontrar una función que cumpla eso exactamente puede ser demasiado ambicioso; por lo tanto nos conformamos con encontrar una función que nos de una aproximación:

$$f(\mathbf{i}_i) \approx d_i \quad \text{para cada } i. \quad (1)$$

O equivalentemente, queremos que la función cometa un error pequeño (es decir, un ϵ lo más chiquito posible) para cada imagen:

$$(f(\mathbf{i}_i) - d_i)^2 \leq \epsilon \quad \text{para cada } i. \quad (2)$$

A partir de esta ecuación podemos derivar que nuestro problema se resume en encontrar una función f^* que minimice la ecuación:

$$\sum_{i=1}^N (f^*(\mathbf{i}_i) - d_i)^2. \quad (3)$$

La pregunta que surge es ¿cómo hacemos para encontrar esta función?.

Implementación

Optimización

Dado que existen infinitas funciones posibles y no podemos probar todas, la idea principal de los algoritmos de regresión logística (desde los métodos lineales más sencillos a las redes neuronales más complejas) reside en limitar la búsqueda a alguna *familia* de funciones. Por ejemplo: funciones lineales, polinomios de grado 2, combinación lineal de funciones periódicas, etcétera. Proponemos entonces buscar la mejor solución dentro de las funciones $f : \mathbb{R}^K \rightarrow (0, 1)$ que tengan la forma:

$$f_{\mathbf{w},b}(\mathbf{i}) = \frac{\tanh(\mathbf{w} \cdot \mathbf{i} + b) + 1}{2}, \quad (4)$$

donde \mathbf{w} es un vector de \mathbb{R}^K , b un escalar, y \tanh la tangente hiperbólica. Nuestro problema entonces se reduce a encontrar \mathbf{w} y b tal que minimicen la Ecuación 3. Podemos reescribir el problema:

$$\operatorname{argmin}_{\mathbf{w},b} \mathcal{L}(\mathbf{w}, b) = \operatorname{argmin}_{\mathbf{w},b} \sum_{i=1}^N (f_{\mathbf{w},b}(\mathbf{i}_i) - d_i)^2. \quad (5)$$

Esta función tendrá mínimos en aquellos lugares donde las derivadas se anulen. Es decir, buscamos que:

$$\frac{\partial}{\partial \mathbf{w}} \sum_{i=1}^N (f_{\mathbf{w},b}(\mathbf{i}_i) - d_i)^2 = 0 \quad (6)$$

$$\frac{\partial}{\partial b} \sum_{i=1}^N (f_{\mathbf{w},b}(\mathbf{i}_i) - d_i)^2 = 0 \quad (7)$$

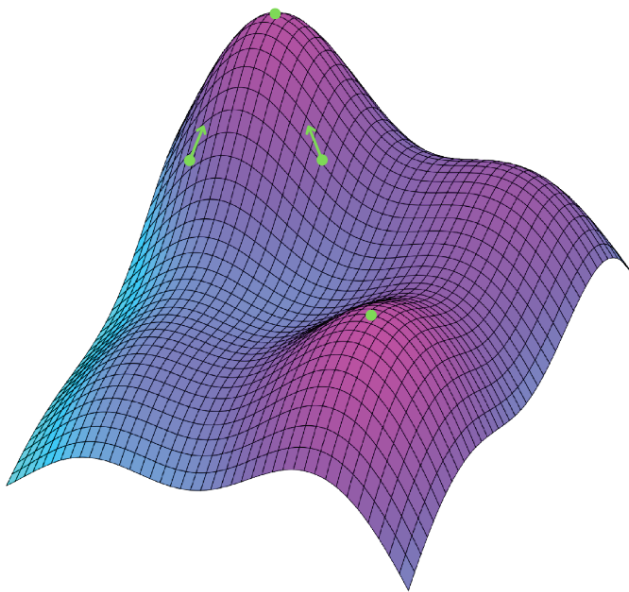
Para hallar estos mínimos utilizaremos el método de *descenso por gradiente*.

Descenso por gradiente

Supongamos que tenemos una función $g : \mathbb{R}^n \rightarrow \mathbb{R}$ diferenciable. El gradiente de g es el vector que contiene las derivadas parciales:

$$\nabla g = \left(\frac{\partial g}{\partial x_1}, \frac{\partial g}{\partial x_2}, \dots, \frac{\partial g}{\partial x_n} \right). \quad (8)$$

Recordemos que el gradiente indica la dirección en la que la función g crece **más rápidamente**:



En esta figura se puede observar como los gradientes de la función (las flechas verdes) apuntan hacia el punto máximo de la función. Los dos puntos verdes indican los máximos locales. En estos lugares, el gradiente es el vector nulo y por lo tanto, no tiene una dirección. Equivalentemente, el opuesto al gradiente apunta en la dirección en la que la función **decrece** más rápidamente. Esto da paso al algoritmo iterativo conocido como *descenso por gradiente*.

Partiendo desde algún punto inicial $x_1^{(0)}, \dots, x_n^{(0)}$. El descenso por gradiente consiste en actualizar iterativamente el punto moviéndose en la dirección opuesta al gradiente hasta alcanzar algún mínimo local de la función.

La **regla de actualización** para cada coordenada es:

$$(x_1^{(i+1)} \dots x_n^{(i+1)}) := (x_1^{(i)} \dots x_n^{(i)}) - \alpha \nabla g(x_1^{(i)}, \dots, x_n^{(i)}). \quad (9)$$

El parámetro α debe **elegirse manualmente**, y es un número pequeño positivo que determina el tamaño del paso en cada iteración. Es decir, cuánto vamos a movernos en la dirección del gradiente. Es sumamente importante elegir valores apropiados, ya que si elegimos un valor muy pequeño entonces el algoritmo tardará muchas iteraciones en alcanzar un mínimo local; y si elegimos un valor muy grande, corremos el riesgo de “pasarnos de largo” de los mínimos locales.

El proceso iterativo consta de los siguientes pasos:

1. **Elegir** aleatoriamente un **punto inicial** $x_1^{(0)}, \dots, x_n^{(0)}$ donde comenzar la búsqueda de un mínimo.
2. **Repetir hasta la convergencia** (o por un número fijo de iteraciones):
 - a) **Calcular** el **gradiente** de la función en el punto actual.
 - b) Mover el punto siguiendo la **regla de actualización** (Ecuación 9).

Evaluación del método

Para analizar el método haremos principalmente dos evaluaciones:

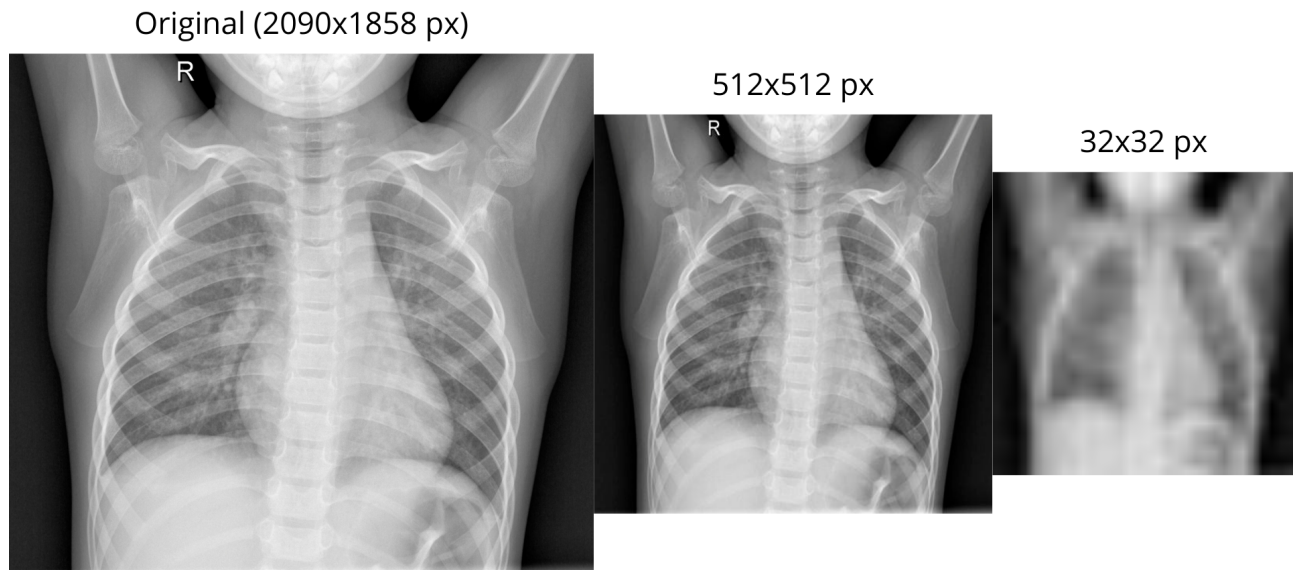
- **Convergencia:** el primer indicio lo basaremos en la convergencia del método.
- **Matriz de confusión:** esta evaluación consiste en analizar qué tan bien el método realiza la clasificación. Como no nos interesa que el algoritmo funcione únicamente para los datos originales; sino que busquemos que el algoritmo funcione correctamente incluso con radiografías que no han sido usadas durante la optimización. De lo contrario su utilidad sería muy limitada. Por este motivo, el *dataset* contiene datos de **entrenamiento** utilizados para realizar las iteraciones de descenso por gradiente, y datos de **testing**, utilizados para evaluar la capacidad del método sobre radiografías que no fueron utilizadas durante la optimización. Para ello dividimos el análisis en cuatro casos:
 - Verdadero-Positivo: el porcentaje de radiografías de un paciente con neumonía que el método clasifica como positivos correctamente.
 - Falso-Negativo: el contrario al caso anterior, porcentaje de pacientes enfermos que el método clasifica como sanos.
 - Verdadero-Negativo: porcentaje de radiografías de pacientes sanos correctamente clasificadas como sanas.
 - Falso-Positivo: el contrario al caso anterior, es decir el porcentaje de pacientes sanos clasificados erróneamente como pacientes enfermos.

Teniendo estos cuatro casos, el objetivo del método es minimizar los casos *Falso-Negativo* y *Falso-Positivo*. Para poder visualizar podemos armar un cuadro como el siguiente:

		Actual Values	
		Yes	No
Predicted Values	Yes	True Positive	False Positive
	No	False Negative	True Negative

Pre-procesamiento

Las imágenes que debemos utilizar tienen un tamaño variable, de promedio 1380x1666px. Es decir, si queremos tratarlas como un arreglo unidimensional, **cada radiografía es un vector de $\mathbb{R}^{2299080}$** . Esto genera un problema de índole práctico para nuestra implementación, ya que para poder realizar el descenso por gradiente (de manera performante) deberíamos cargar todas las imágenes en memoria en simultaneo lo que implicaría una carga espacial de más de 23GB. Para lidiar con este problema existen múltiples técnicas. La más sencilla, es escalar cada imagen para que tenga un tamaño mucho más reducido; por ejemplo, 32x32 píxeles, o 64x64, o 512x512. Vale mencionar que cuanto mayor sea esta reducción, más veloz será el algoritmo en realizar iteraciones. Sin embargo, al reducir el tamaño de la imagen perdemos calidad e información. Es decir, si la reducción es muy agresiva corremos el riesgo de que el método no logre distinguir satisfactoriamente entre pacientes enfermos y sanos. En la siguiente figura podemos observar este fenómeno:



Ejercicios

Ejercicio 1. Calcular las derivadas parciales de la función \mathcal{L} con respecto a \mathbf{w} y \mathbf{b} (pueden asistirse mediante www.matrixcalculus.org)

Ejercicio 2. Implementar el método de *descenso por gradiente* y optimizar los parámetros de la función f para el conjunto de datos de **entrenamiento**. Para esto les recomendamos que trabajen con un subconjunto de los datos que tenga una cantidad parecida de imágenes con y sin neumonía

Ejercicio 3. Calcular el error cuadrático durante la optimización para el conjunto de **entrenamiento** y para el conjunto de **testing**. Generar las visualizaciones correspondientes.

Ejercicio 4. Analizar el impacto del parámetro α (Ecuación 9) en la convergencia del método. Tomar un rango de 5 valores posibles y analizar la convergencia para el conjunto de **testing** para los distintos valores de α .

Ejercicio 5. ¿Cómo impacta el tamaño del escalado de las imágenes en la efectividad del método? ¿Y en el tiempo de cómputo?. Realizar los experimentos y gráficos acordes para estudiar estas limitaciones.

Ejercicio 6. Para el valor de α que tenga mejor valor de convergencia, generar la matriz de confusión y analizar brevemente la efectividad del método.

Condiciones de Entrega

La entrega del trabajo debe consistir en los siguientes archivos:

1. `informe.pdf`: un informe o reporte en el que se describan las tareas realizadas, cómo fueron resueltas, los resultados de sus experimentos y conclusiones. Puede ser escrito en programas del tipo Word, Markdown o \LaTeX , controlando que al exportar el archivo no haya errores de fórmulas ni de formato.
2. `matrizRala.py`: El archivo modificado con la implementación

Además, se deben tener en cuenta las siguientes consideraciones:

- El trabajo debe realizarse en grupos de 3 alumnos.
- La fecha límite de entrega es el 1 de Julio.
- La entrega debe ser realizada a través del campus por sólo 1 de los integrantes del grupo (no entregar por duplicado).
- En cada uno de los archivos entregados deben especificarse, arriba de todo, los nombres y apellidos completos de los integrantes del grupo.
- Las consignas sirven como guía de trabajo, se valorará el uso de técnicas vistas en clase como operaciones de vectores, matrices, y demás.
- Mantener el mayor nivel de prolijidad posible, tanto en los desarrollos como en el código. Las resoluciones deben estar escritas de forma clara y precisa.