Métodos Computacionales
Segundo Trabajo Práctico
Primer Semestre 2025

Introducción

El Parkinson es una enfermedad neurodegenerativa que afecta principalmente al control del movimiento. Uno de los síntomas más comunes es el temblor, lo que se refleja de manera clara en tareas motoras finas como dibujar una espiral. Este patrón se ha utilizado en investigaciones médicas como una forma sencilla de observar diferencias entre personas sanas y pacientes con Parkinson.

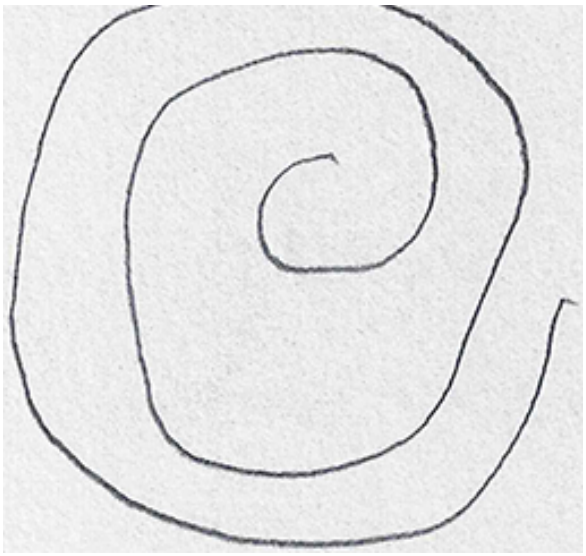


Figura 1: Paciente sano

Figura 2: Paciente con Parkinson

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 distinguir entre espirales de pacientes con Parkinson y espirales de personas sanas. Plantearemos la solución como un problema de clasificación binaria mediante una función no lineal simple.

Método

El conjunto de datos a utilizar se puede descargar en la sección de recursos del Trabajo Práctico Consiste en 1633 dibujos de espirales, cuenta con casos de personas con y sin Parkison.

Les proponemos pensar el problema de la siguiente manera. Tenemos un conjunto de imágenes $\mathbf{i}_1, \ldots, \mathbf{i}_N$ y sus diagnósticos asociados $d_1, \ldots, d_N$. Cada $\mathbf{i}_i$ es una matriz de píxeles RGB, es decir, una matriz con dimensiones [widthSize, heightSize, 3] donde cada "pixel"tiene 3 valores entre 0 y 255 que indica la cantida de Red, Green y Blue que deberia tener el pixel. Los números d_i son el valor binario 0 ó 1 que indica si el paciente esta sano o tiene parkinson. 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:

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

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

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

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

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

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 $f : \mathbb{R}^K \to (0, 1)$ que tengan la forma:

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

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:

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

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

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

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

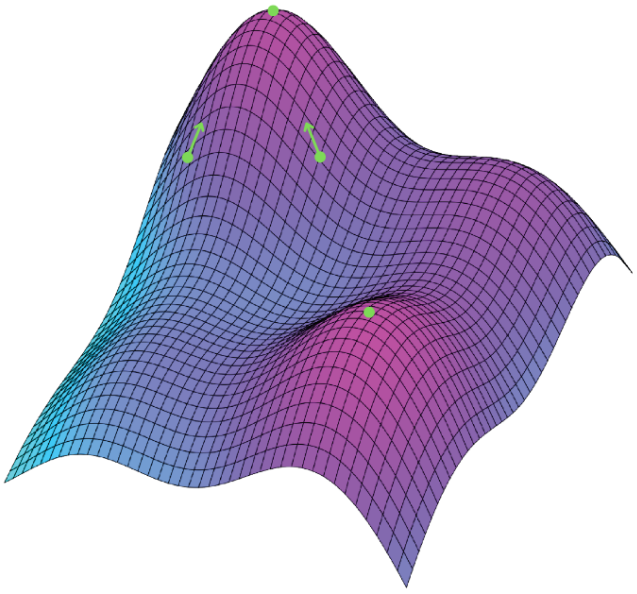
Para hallar estos mínimos utilizaremos el método de *descenso por gradiente*.

Descenso por gradiente

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

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

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 $x_1^{(0)}, \cdots, x_n^{(0)}$. 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:

$$\left(x_1^{(i+1)} \ \cdots \ x_n^{(i+1)} \right) := \left(x_1^{(i)} \ \cdots \ x_n^{(i)} \right) - \alpha \ \nabla g(x_1^{(i)}, \ldots, x_n^{(i)}). \tag{9}$$

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 $x_1^{(0)}, \cdots, x_n^{(0)}$ 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).

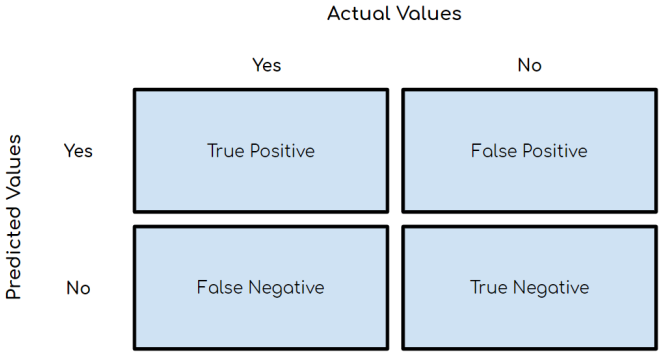
Evaluación del método

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 buscamos que el algoritmo funcione correctamente incluso con dibujos 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 dibujos que no fueron utilizadas durante la optimización. Para ello dividimos el análisis en cuatro casos:
  - Verdadero-Positivo: el porcentaje de dibujos de un paciente con parkison que el método clasifica como positivos correctamente.
  - Falso-Negativo: el contrario al caso anterior, porcentaje de pacientes con parkinson que el método clasifica como sanos.
  - Verdadero-Negativo: porcentaje de dibujos de personas sanas correctamente clasificadas como sanas.

- Falso-Positivo: el contrario al caso anterior, es decir el porcentaje de personas sanas clasificados erróneamente como pacientes con parkinson.

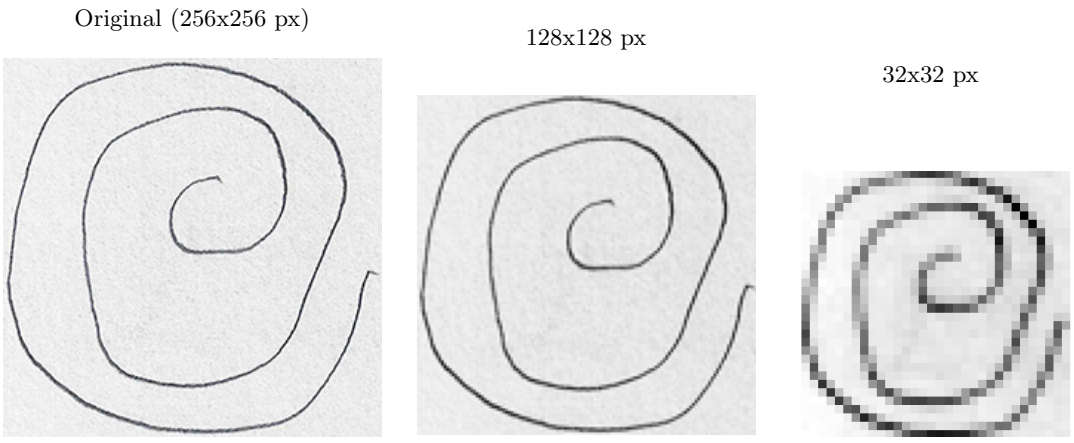
Teniendo estos cuatro casos, el objetivo del método es minimizar los casos *Falso-Negativo* y *Falso-Positivo*. Para poder visualizar podemos armar un cuadro como el siguiente:



Pre-procesamiento

Las imágenes que debemos utilizar tienen un tamaño variable, de promedio 256x256x3. Es decir, si queremos tratarlas como un arreglo unidimensional, cada dibujo es un vector de $\mathbb{R}^{196,608}$. Aunque esto no representa un problema insalvable, en la práctica es importante considerar que cargar todas las imágenes en memoria simultáneamente puede generar una carga considerable.

Para lidiar con este problema existen múltiples técnicas. La más sencilla, es pasar las imagenes a una escala de grises y finalmente escalar cada imagen para que tenga un tamaño mucho más reducido; por ejemplo, 32x32x1 píxeles, o 64x64x1, o 128x128x1. 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:



Parte 1: Descenso de gradiente

Ejercicio 1. Calcular las derivadas parciales de la función $\mathcal{L}$ con respecto a $\mathbf{w}$ y 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 imagenes de dibujos de personas con y sin Parkison.

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

Ejercicio 4. Compare el resultado anterior si se entrena el modelo con datos normalizados, es decir, con valores entre 0 y 1.

Ejercicio 5. 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. Graficar resultados

Ejercicio 6. ¿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 7. Para el valor de α que tenga mejor valor de convergencia, generar la matriz de confusión y analizar brevemente la efectividad del método.

Parte 2: Ascenso de gradiente

Otro enfoque diferente de aplicar el método de descenso por gradiente es no minimizar una función de pérdida, sino directamente maximizar la log-verosimilitud.

En este caso, en lugar de trabajar con la pérdida negativa (que convierte el problema en una minimización), podemos simplemente aplicar el gradiente de la log-verosimilitud y movernos en la dirección del gradiente ascendente, es decir, ajustar los parámetros $\mathbf{w}$ y b para aumentar el valor de la log-verosimilitud.

Es importante notar que, al minimizar la distancia entre los puntos y sus aproximaciones, se minimiza el error en un sentido geométrico. Por otro lado, al trabajar con la log-verosimilitud se considera la naturaleza probabilística del problema, lo que hace que este análisis resulte, generalmente, más adecuado en problemas de clasificación.

Podemos definir la log-verosimilitud total como:

$$\log \mathcal{L}(f) = \sum_{i=1}^{N} d_i \log(f(\mathbf{i}_i)) + (1 - d_i) \log(1 - f(\mathbf{i}_i))] \,.$$

Siendo,

$$f_{\mathbf{w},b}(\mathbf{i}) = \frac{1}{1 + e^{-(\mathbf{w} \cdot \mathbf{i} + b)}}, \tag{11}$$

donde $\mathbf{w}$ es un vector en $\mathbb{R}^K$, b es un escalar, y f es la conocida función sigmoide. Esta función devuelve valores entre 0 y 1, lo cual nos permite interpretarlos como probabilidades.

Nuestro problema se reduce entonces a encontrar los parámetros $\mathbf{w}$ y b que **maximizan** la log-verosimilitud, o equivalentemente, que **minimizan** la log-verosimilitud multiplicada por -1:

$$\underset{\mathbf{w},b}{\operatorname{argmin}}, \mathcal{L}(\mathbf{w}, b) = -\sum_{i=1}^{N} d_i \log(f_{\mathbf{w},b}(\mathbf{i} * i)) + (1 - d_i) \log(1 - f * \mathbf{w}, b(\mathbf{i}_i))] \,.$$

Ejercicios

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

Ejercicio 2. Implementar el método de *ascenso por gradiente* y optimizar los parámetros de la función f para el conjunto de datos de **entrenamiento**. *Nota:* Utilicen los vectores de las imágenes con el mejor tamaño de escala encontrado y normalizados.

Ejercicio 3. Calcular la exactitud (accuracy) 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 en la convergencia del método. Tomar los 2 mejores valores de α en la Parte 1 y compararlos.

Ejercicio 5. Analizar los resultados obtenidos del modelo en la Parte 1 y los resultados del modelo en la Parte 2 y compararlos. Utilizar los resultados y visualizaciones acordes.

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, visualizaciones con su propio análisis 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. `codigo:` Los archivos *.py* o *.ipynb* 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 7 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.
- En los ejercicios que siguen, **no está permitido utilizar librerías externas** (como `sklearn`, `tensorflow`, `pytorch`, entre otras) que ya implementen las funcionalidades solicitadas. La intención es que ustedes mismos construyan el código necesario a partir de los conceptos explicados en clase.
- 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.