Tarea 1

Aprendizaje supervisado, funciones de pérdida y dimensión VC
Reconocimiento de Patrones y AA 26-01

Instrucciones: Resuelve los siguientes ejercicios de manera individual. La entrega es en un documento PDF. Puedes escribir tus respuestas en el documento mediante algún software o completamente a mano. Si lo haces a mano, cada paso debe ser legible, y las fotos que tomes se deben incorporar como parte del documento PDF.

1. [2.5 puntos] Dimensión VC

Para una clase de hipótesis de triángulos $\mathcal{H}_{\text{tri}}$:

a) Escribe una función indicadora para evaluar si un punto 2D pertenece o no a una hipótesis de un triángulo particular $h_{\boldsymbol{\theta}} \in \mathcal{H}_{\text{tri}}$.

Solución:

Usando coordenadas baricéntricas:

$$h_{\boldsymbol{\theta}}(x, y) = \begin{cases} 1 & \text{si } \lambda_1 \geq 0, \lambda_2 \geq 0, \lambda_3 \geq 0 \\ 0 & \text{en caso contrario} \end{cases}$$

donde:

$$\lambda_1 = \frac{(y_2 - y_3)(x - x_3) + (x_3 - x_2)(y - y_3)}{(y_2 - y_3)(x_1 - x_3) + (x_3 - x_2)(y_1 - y_3)}$$

$$\lambda_2 = \frac{(y_3 - y_1)(x - x_3) + (x_1 - x_3)(y - y_3)}{(y_2 - y_3)(x_1 - x_3) + (x_3 - x_2)(y_1 - y_3)}$$

$$\lambda_3 = 1 - \lambda_1 - \lambda_2$$

b) Debes escribir la función parametrizada por los parámetros $\boldsymbol{\theta}$ que tú consideres.

Solución:

$$\boldsymbol{\theta} = (x_1, y_1, x_2, y_2, x_3, y_3) \in \mathbb{R}^6$$

Función parametrizada:

$$h_{\boldsymbol{\theta}}(x, y) = \begin{cases} 1 & \text{si el punto } (x, y) \text{ está dentro del triángulo} \\ 0 & \text{en caso contrario} \end{cases}$$

Implementación:

```
def indicator_triangle(x, y, theta):
    x1, y1, x2, y2, x3, y3 = theta
    denom = (y2 - y3) * (x1 - x3) + (x3 - x2) * (y1 - y3)

    if abs(denom) < 1e-10:
        return 0

    lambda1 = ((y2 - y3) * (x - x3) + (x3 - x2) * (y - y3)) / denom
    lambda2 = ((y3 - y1) * (x - x3) + (x1 - x3) * (y - y3)) / denom
    lambda3 = 1 - lambda1 - lambda2

    return 1 if (lambda1 >= 0 and lambda2 >= 0 and lambda3 >= 0) else 0
```

c) Luego usarás dicha función en la práctica asociada a este ejercicio para demostrar exhaustivamente (mediante un programa en Python) que la dimensión VC de la clase $\mathcal{H}_{\text{tri}}$ de triángulos es 7.

Solución:

La dimensión VC es 7:

1. Existen 7 puntos que pueden ser shattered:

Toma 7 puntos en un círculo: $P_k = (\cos(2\pi k/7), \sin(2\pi k/7))$ para $k = 0, 1, \ldots, 6$.

Un triángulo es la intersección de 3 semiplanos, por lo que puede crear hasta 3 intervalos de puntos positivos sobre el círculo. Con 7 puntos, cualquier etiquetado tiene a lo más 6 cambios, realizable con 3 intervalos.

2. NO existen 8 puntos que puedan ser shattered:

Con 8 puntos en círculo y etiquetado alternante (+, -, +, -, ...), hay 8 transiciones que requieren 4 intervalos. Un triángulo solo puede generar 3 intervalos, por lo que este etiquetado no es realizable.

Implementación:

```
# Verificar que 7 puntos pueden ser shattered
puntos_7 = generar_7_puntos_en_circulo()
for etiquetado in all_labelings(2**7):
    assert existe_triangulo_separador(puntos_7, etiquetado)

# Verificar que 8 puntos NO pueden ser shattered
puntos_8 = generar_8_puntos_en_circulo()
etiquetado_alt = alternar_signos(8)  # + - + - + - + -
assert not existe_triangulo_separador(puntos_8, etiquetado_alt)
```

Por tanto, VC-dim = 7.

2. [2.5 puntos] Clases de hipótesis

Para una clase de hipótesis de elipses $\mathcal{H}_{\text{eli}}$ donde una elipse con centro en la coordenada (h, k) se describe con la siguiente ecuación:

$$\frac{(x-h)^2}{a^2} + \frac{(y-k)^2}{b^2} = 1 \tag{1}$$

donde $a > 0$ y $b > 0$ son los parámetros para los semiejes de la elipse.

a) Usa esta ecuación para escribir una función indicadora para evaluar si un punto en 2D pertenece o no a una hipótesis de una elipse particular $h_{\boldsymbol{\theta}} \in \mathcal{H}_{\text{eli}}$, donde $\boldsymbol{\theta} = \{h, k, a, b\}$.

Solución:

$$h_{\boldsymbol{\theta}}(x, y) = \begin{cases} 1 & \text{si } \frac{(x-h)^2}{a^2} + \frac{(y-k)^2}{b^2} \leq 1 \\ 0 & \text{en caso contrario} \end{cases}$$

b) Escribe a manera de pseudocódigo un algoritmo para encontrar los parámetros $\boldsymbol{\theta}' = \{h', k', a', b'\}$ de la elipse más pequeña que sólo permite un error menor o igual a un umbral α (relativamente pequeño; por ejemplo, 10 %) de clasificaciones incorrectas (para una de dos clases) del total de puntos $\mathcal{X} = \{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\}$.

Solución:

```
ALGORITMO: Elipse_Minimo_Con_Umbral_Alpha (búsqueda por escalamiento)

ENTRADA:
- X: conjunto de puntos {(x_i, y_i)}
- Y: etiquetas en {+1, -1} (interior = +1)
- alpha \in (0, 1): umbral de error permitido (p. ej., 0.1; puede ser > 0.1 s

1. Separar por clase y fijar presupuesto de error:
   X_pos = { (x_i, y_i) | Y_i = +1 }
   X_neg = { (x_i, y_i) | Y_i = -1 }
   n = |X|
   max_errores = floor(alpha * n)

2. Estimar centro y forma base con la clase positiva:
   h' = media(x | (x,y) en X_pos)
   k' = media(y | (x,y) en X_pos)
   eps = 1e-9
   a0 = desviacion_estandar(x | (x,y) en X_pos) + eps
   b0 = desviacion_estandar(y | (x,y) en X_pos) + eps

3. Precomputar distancias normalizadas respecto a (h', k', a0, b0):
   Para cada punto (x_i, y_i):
       d_i = sqrt( (x_i - h')^2 / a0^2 + (y_i - k')^2 / b0^2 )

4. Construir lista de escalas candidatas:
```

```
   S = valores_únicos_ordenados({ d_i : i = 1..n })
   # Clasificación con escala s: pred_i(s) = +1 si d_i <= s, \
   # en otro caso -1

5. Encontrar el menor s que cumple el presupuesto:
   mejor_s = None
   mejor_err = +infinito
   PARA cada s en S (en orden ascendente):
       errores = contar_errores_con_s(d, Y, s)
       SI errores < mejor_err:
           mejor_err = errores
       SI errores <= max_errores y (mejor_s es None):
           mejor_s = s   # es el más pequeño por el orden
           # opcional: si mejor_err == 0, romper
   SI mejor_s es None:
       # No hay s que cumpla; tomar el s con menor #errores (y el más \
       # pequeño en caso de empate)
       mejor_s = argmin_por((errores(s), s), s en S)

6. Definir la elipse mínima:
   a' = a0 * mejor_s
   b' = b0 * mejor_s
   theta' = {h', k', a', b'}
   RETORNAR theta'

FUNCIÓN contar_errores_con_s(d, Y, s):
    errores = 0
    PARA i = 1..n:
        pred = +1 si d_i <= s, en otro caso -1
        SI pred != Y_i: errores += 1
    RETORNAR errores
```

c) Ahora encuentra también a manera de pseudocódigo los parámetros del elipse más grande que no introduce nuevos errores. Puedes partir de los parámetros $\boldsymbol{\theta}'$ del inciso anterior.

Solución:

```
ALGORITMO: Elipse_Maximo_Sin_Nuevos_Errores (expansión segura)

ENTRADA:
- X, Y: puntos y etiquetas
- theta' = {h', k', a', b'}: la elipse mínima del inciso b)

1. errores_iniciales = contar_errores(X, Y, theta')

2. Para cada punto negativo (Y_i = -1), calcular
   d_i = sqrt( (x_i - h')^2 / a'^2 + (y_i - k')^2 / b'^2 )
```

```
3. Factor de expansión máximo que no introduce nuevos errores:
   # Al escalar a', b' por s > 1, las nuevas distancias son d_i / s.
   candidatos = { d_i : Y_i = -1 y d_i > 1 }
   SI candidatos no vacío:
       s_max = min(candidatos) - epsilon   # asegura que ningún negativo \
                                           # fuera cruce a d<=1
       s_max = max(s_max, 1.0)
   EN OTRO CASO:
       s_max = 1.0   # cualquier expansión introduciría nuevos FP si ya \
                     # hay negativos en el borde

4. Definir la elipse expandida:
   a'' = a' * s_max
   b'' = b' * s_max
   theta'' = {h', k', a'', b''}

5. Verificación:
   SI contar_errores(X, Y, theta'') > errores_iniciales:
       s_max = s_max * 0.999  # reducir ligeramente y volver a probar
       repetir paso 4 y 5 hasta cumplir

6. RETORNAR theta''

FUNCIÓN contar_errores(X, Y, {h, k, a, b}):
    errores = 0
    PARA cada (x_i, y_i), Y_i:
        d = sqrt( (x_i - h)^2 / a^2 + (y_i - k)^2 / b^2 )
        pred = +1 si d <= 1, en otro caso -1
        SI pred != Y_i: errores += 1
    RETORNAR errores
```

d) Finalmente propón como hipótesis para la función indicadora un elipse que esté entre ambos (de los incisos b y c), por ejemplo a la mitad del camino paramétricamente hablando.

Solución:

Elipse intermedio:

$$h_{\text{final}} = h' \tag{2}$$

$$k_{\text{final}} = k' \tag{3}$$

$$a_{\text{final}} = \frac{a' + a''}{2} \tag{4}$$

$$b_{\text{final}} = \frac{b' + b''}{2} \tag{5}$$

$$h_{\text{final}}(x, y) = \begin{cases} 1 & \text{si } \frac{(x-h')^2}{a_{\text{final}}^2} + \frac{(y-k')^2}{b_{\text{final}}^2} \leq 1 \\ 0 & \text{en caso contrario} \end{cases}$$

e) Implementarás esto en la práctica asociada a este ejercicio.

3. [2.5 puntos] Función de pérdida esperada

Dados los valores de una función de pérdida L_{kj}, el riesgo esperado

$$E[L] = \sum_k \sum_j \int_{R_j} L_{kj} p(x, \mathcal{C}_k) dx \tag{6}$$

se minimiza si para cada x escogemos la clase para minimizar la siguiente expresión (ya que $p(x)$ no contiene al parámetro de clase):

$$\arg\min_j \left(\sum_k L_{kj} p(\mathcal{C}_k|x) p(x) \right) = \arg\min_j \left(\sum_k L_{kj} p(\mathcal{C}_k|x) \right) \tag{7}$$

Verifica que si $L_{kj} = 1 - I_{kj}$, donde I_{kj} son los elementos de la matriz identidad, entonces la minimización anterior se reduce a escoger la clase i con la probabilidad más grande $p(\mathcal{C}_i|x)$.

Solución:

Con $L_{kj} = 1 - I_{kj}$:

$$L_{kj} = \begin{cases} 0 & \text{si } k = j \\ 1 & \text{si } k \neq j \end{cases}$$

La expresión a minimizar:

$$\sum_k L_{kj} p(\mathcal{C}_k|x) = \sum_k (1 - I_{kj}) p(\mathcal{C}_k|x) \tag{8}$$

$$= \sum_k p(\mathcal{C}_k|x) - \sum_k I_{kj} p(\mathcal{C}_k|x) \tag{9}$$

$$= 1 - p(\mathcal{C}_j|x) \tag{10}$$

Minimizar $[1 - p(\mathcal{C}_j|x)]$ equivale a maximizar $p(\mathcal{C}_j|x)$.

Por tanto, la regla óptima es:

$$i^* = \arg\max_j p(\mathcal{C}_j|x)$$

4. [2.5 puntos] Función de pérdida relajada

En los ejercicios 1 y 2 obtuviste funciones indicadoras para evaluar la pertenencia o no de puntos a hipótesis concretas.

a) Debes extender la función indicadora para crear una nueva función que no evalúe de manera estrictamente "dura" la pertenencia a la hipótesis sino que permita considerar puntos cercanos a la frontera de la hipótesis de manera penalizada.

Solución:

Función de pérdida suave para la elipse:

$$L_{\text{soft}}(x,y) = \begin{cases} 0 & \text{si } d(x,y) \leq 1 \\ \frac{d(x,y)-1}{\delta} & \text{si } 1 < d(x,y) \leq 1+\delta \\ 1 & \text{si } d(x,y) > 1+\delta \end{cases}$$

donde $d(x,y) = \sqrt{\frac{(x-h)^2}{a^2} + \frac{(y-k)^2}{b^2}}$ y $\delta > 0$ controla el ancho de transición.

b) Tú debes definir una manera de extender la frontera hasta cierta distancia máxima más allá de la frontera original. Por ejemplo: englobando a la hipótesis dentro de un círculo con cierto radio, o definiendo una distancia Δd más allá de la frontera de la hipótesis.

Solución:

Zona de transición con ancho δ:

- Frontera original: $d(x,y) = 1$
- Zona de transición: $1 < d(x,y) \leq 1+\delta$
- Región exterior: $d(x,y) > 1+\delta$

La zona forma una cáscara elíptica entre:

- Elipse original: $\frac{(x-h)^2}{a^2} + \frac{(y-k)^2}{b^2} = 1$
- Elipse expandida: $\frac{(x-h)^2}{(a(1+\delta))^2} + \frac{(y-k)^2}{(b(1+\delta))^2} = 1$

c) En la región donde extiendas la frontera la función de pérdida debe penalizar en lugar de evaluar a cero.

Solución:

$$L_{\text{relajada}}(x,y) = \begin{cases} 0 & \text{si } d(x,y) \leq 1 \\ \frac{d(x,y)-1}{\delta} & \text{si } 1 < d(x,y) \leq 1+\delta \\ 1 & \text{si } d(x,y) > 1+\delta \end{cases}$$

Penalización:

- Interior ($d \leq 1$): Sin pérdida
- Transición ($1 < d \leq 1+\delta$): Penalización lineal de 0 a 1
- Exterior ($d > 1+\delta$): Pérdida máxima

d) Puedes elegir para este ejercicio una de las hipótesis de los ejercicios 1 y 2.

Solución:

Elección: Elipse del Ejercicio 2

$$L_{\text{elipse}}(x,y,h,k,a,b,\delta) = \begin{cases} 0 & \text{si } d \leq 1 \\ \frac{d-1}{\delta} & \text{si } 1 < d \leq 1+\delta \\ 1 & \text{si } d > 1+\delta \end{cases}$$

donde $d = \sqrt{\frac{(x-h)^2}{a^2} + \frac{(y-k)^2}{b^2}}$

e) Implementarás esto en la práctica asociada a este ejercicio.

Solución:

Implementación:

```
def soft_loss_ellipse(x, y, h, k, a, b, delta):
    dist = np.sqrt((x - h)**2 / a**2 + (y - k)**2 / b**2)

    if dist <= 1:
        return 0
    elif dist <= 1 + delta:
        return (dist - 1) / delta
    else:
        return 1
```