Examen de RFA, MIARFID, DSIC, UPV, 2-11-2023

Apellidos: | **Nombre:**

Cuestiones (1 punto)

Marca cada recuadro con una única opción de entre las dadas o déjalo en blanco si no quieres contestar. Puntuación: $(A - E/3) \cdot 1/6$ con A aciertos y E errores; $A, E \in \{0, 1, \ldots, 6\}$, $A + E \leq 6$.

1 [4] Sea un problema de clasificación en C clases, $y \in \{1, 2, \ldots, C\}$, para objetos representados mediante vectores de D características. Los clasificadores naive Bayes son clasificadores generativos basados en la asunción naive Bayes. Indica cuál de las siguientes opciones expresa correctamente dicha asunción.

1. $p(y = c \mid \boldsymbol{x}, \boldsymbol{\theta}) = \sum_{d=1}^{D} p(y = c \mid x_d, \boldsymbol{\theta}_{dc})$
2. $p(y = c \mid \boldsymbol{x}, \boldsymbol{\theta}) = \prod_{d=1}^{D} p(y = c \mid x_d, \boldsymbol{\theta}_{dc})$
3. $p(\boldsymbol{x} \mid y = c, \boldsymbol{\theta}) = \sum_{d=1}^{D} p(x_d \mid y = c, \boldsymbol{\theta}_{dc})$
4. $p(\boldsymbol{x} \mid y = c, \boldsymbol{\theta}) = \prod_{d=1}^{D} p(x_d \mid y = c, \boldsymbol{\theta}_{dc})$

2 [4] Indica la afirmación incorrecta (o escoge la última opción si las tres primeras son correctas):

1. Análisis discriminante Gaussiano asume que las densidades condicionales de las clases son Gaussianas independientes, lo que resulta en discriminantes (log-posteriors) cuadráticas.
2. Análisis discriminante lineal asume que las densidades condicionales de las clases son Gaussianas de matriz de covarianzas común, lo que resulta en discriminantes (log-posteriors) lineales.
3. LDA coincide con regresión logística (multinomial) en términos de modelo, pero se diferencia mucho en entrenamiento.
4. Todas son correctas.

3 [3] Sea un problema de clasificación en dos clases, $y \in \{0, 1\}$, y sea $p(y \mid \boldsymbol{x}; \boldsymbol{\theta}) = \mathrm{Ber}(y \mid \sigma(\boldsymbol{w}^t \boldsymbol{x} + b))$ un modelo de regresión logística binaria con $\boldsymbol{w} = (1, 1)^t$ y $b = -1$. La probabilidad de que $\boldsymbol{x} = (1, 1)^t$ pertenezca a la clase 1, $p = p(y = 1 \mid \boldsymbol{x}; \boldsymbol{\theta})$, es:

1. $p < 0.25$
2. $0.25 \leq p < 0.5$
3. $0.5 \leq p < 0.75$ 0.7311
4. $0.75 \leq p$

4 [1] Sea un problema de clasificación en $C = 3$ clases, y sea $p(y \mid \boldsymbol{x}; \boldsymbol{\theta}) = \mathrm{Cat}(y \mid \mathcal{S}(\mathbf{W}^t \boldsymbol{x} + \boldsymbol{b}))$, un modelo de regresión logística multinomial con

$$\mathbf{W}^t = \begin{pmatrix} 1 & 1 \\ 1 & 0 \\ 0 & 1 \end{pmatrix} \in \mathbb{R}^{C \times D} \qquad \text{y} \qquad \boldsymbol{b} = \begin{pmatrix} 1 \\ -1 \\ 0 \end{pmatrix}$$

La probabilidad de que $\boldsymbol{x} = (1, 1)^t$ pertenezca a la clase 2, $p(y = 2 \mid \boldsymbol{x}; \boldsymbol{\theta})$, es:

1. $p < 0.25$ 0.042
2. $0.25 \leq p < 0.5$
3. $0.5 \leq p < 0.75$
4. $0.75 \leq p$

5 [4] Indica la afirmación incorrecta sobre perceptrones (redes) multicapa:

1. El problema XOR es un problema famoso del libro Perceptrons con datos 2d no linealmente separables. No puede resolverse con un perceptrón clásico (con activación Heaviside), pero sí con un perceptrón multicapa que combina tres perceptrones clásicos.
2. Los perceptrones multicapa clásicos han caído en desuso ya que emplean una función de activación no diferenciable. Actualmente se emplean redes multicapa con funciones de activación diferenciables como la sigmoide, tanh y ReLU.
3. La revolución del aprendizaje profundo se produce a partir de los 2010, gracias a una mayor disponibilidad de (grandes) conjuntos de datos, potencia de cálculo (en GPU sobre todo) y librerías especializadas de código abierto como tensorflow y pytorch.
4. Las redes con muchas capas (profundas) suelen ofrecer mejores resultados que las redes con pocas capas (superficiales). De hecho, se sabe que un perceptrón de una sola capa oculta se halla muy limitado como aproximador de funciones. (en teoría basta una capa oculta con suficientes unidades)

6 [3] Indica la afirmación incorrecta sobre el entrenamiento de redes con descenso por gradiente estocástico (o escoge la última opción si las tres primeras son correctas):

1. El ajuste del factor de aprendizaje es necesario para garantizar la convergencia a una buena solución.
2. Al entrenar modelos (muy) profundos, se suele presentar el problema de los gradientes que desaparecen y explotan. La explosión de gradientes puede evitarse con *gradient clipping*.
3. La desaparición de gradientes puede aliviarse modificando la arquitectura de la red con funciones de activación saturantes (en lugar de no saturantes), estandarización de activaciones (capas batchnorm) e inclusión de conexiones aditivas junto con las multiplicativas (capas con conexiones residuales).
4. Todas son correctas.

Problema (1.5 puntos)

Sea un perceptrón con una capa oculta,

$$\boldsymbol{h} = \boldsymbol{\sigma}(\boldsymbol{z}) \qquad \text{con} \qquad \boldsymbol{z} = \mathbf{W}\boldsymbol{x} + \boldsymbol{b}_1 \qquad \mathbf{W} = \begin{pmatrix} -1 & -1 \\ 1 & -1 \end{pmatrix} \qquad \boldsymbol{b}_1 = \begin{pmatrix} 1 \\ -1 \end{pmatrix}$$

y una capa de salida,

$$\hat{y} = \mathbf{V}\boldsymbol{h} + b_2 \qquad \text{con} \qquad \mathbf{V} = \begin{pmatrix} -1 & 1 \end{pmatrix} \qquad b_2 = 1$$

Dada la entrada $\boldsymbol{x} = (2, 2)^t$ y salida $y = -1$, se desea ajustar el modelo con el fin de minimizar la pérdida cuadrática,

$$\mathcal{L} = \frac{1}{2}(y - \hat{y})^2$$

Se pide:

1. (0.5 puntos) *Forward:* halla las pre-activaciones, activaciones y pérdida correspondientes al par entrada-salida dado.

```
import numpy as np; np.set_printoptions(precision=4)
sigmoid = lambda x: 1.0 / (1.0 + np.exp(-x));
x = np.array([2, 2]); y = -1
W = np.array([[-1, -1], [1, -1]]); b1 = np.array([1, -1])
V = np.array([1, -1]); b2 = 1
z = (W @ x + b1);                    print("z =", z)
h = sigmoid(z);                      print("h =", h)
haty = V @ h + b2;                   print("haty =", round(haty, 4))
loss = .5 * np.square(y-haty).sum(); print("loss =", round(loss, 4))
```

```
z = [-3 -1]
h = [0.0474 0.2689]
haty = 1.2215
loss = 2.4676
```

2. (0.75 puntos) *Backward:* usa los cálculos anteriores y halla las Jacobianas de la pérdida con respecto a . . .

$$u = \frac{\partial \mathcal{L}}{\partial \hat{y}} = \hat{y} - y$$ la predicción (activación de la capa de salida)

$$\boldsymbol{g}_{\mathbf{V}} = u \frac{\partial \hat{y}}{\partial \mathbf{V}} = \dots$$ la transformación lineal de la capa de salida

$$g_{b_2} = u \frac{\partial \hat{y}}{\partial b_2} = \dots$$ el sesgo de la capa de salida

$$\boldsymbol{u}^t = u \frac{\partial \hat{y}}{\partial \boldsymbol{h}} = u\mathbf{V}$$ la activación de la capa oculta

$$\boldsymbol{u}^t = \boldsymbol{u}^t \frac{\partial \boldsymbol{h}}{\partial \boldsymbol{z}} = \boldsymbol{u}^t \operatorname{diag}(\boldsymbol{\sigma}'(\boldsymbol{z}))$$ la pre-activación de la capa oculta

$$\boldsymbol{g}_{\mathbf{W}} = \boldsymbol{u}^t \frac{\partial \boldsymbol{z}}{\partial \mathbf{W}} = \dots$$ la transformación lineal de la capa oculta

$$\boldsymbol{g}_{\boldsymbol{b}_1} = \boldsymbol{u}^t \frac{\partial \boldsymbol{z}}{\partial \boldsymbol{b}_1} = \dots$$ el sesgo de la capa oculta

$$u = \frac{\partial \mathcal{L}}{\partial \hat{y}} = \hat{y} - y$$ la predicción (activación de la capa de salida)

$$\boldsymbol{g}_{\mathbf{V}} = u \frac{\partial \hat{y}}{\partial \mathbf{V}} = \boldsymbol{h}u$$ la transformación lineal de la capa de salida

$$g_{b_2} = u \frac{\partial \hat{y}}{\partial b_2} = u$$ el sesgo de la capa de salida

$$\boldsymbol{u}^t = u \frac{\partial \hat{y}}{\partial \boldsymbol{h}} = u\mathbf{V}$$ la activación de la capa oculta

$$\boldsymbol{u}^t = \boldsymbol{u}^t \frac{\partial \boldsymbol{h}}{\partial \boldsymbol{z}} = \boldsymbol{u}^t \operatorname{diag}(\boldsymbol{\sigma}'(\boldsymbol{z}))$$ la pre-activación de la capa oculta

$$\boldsymbol{g}_{\mathbf{W}} = \boldsymbol{u}^t \frac{\partial \boldsymbol{z}}{\partial \mathbf{W}} = \boldsymbol{x}\boldsymbol{u}^t$$ la transformación lineal de la capa oculta

$$\boldsymbol{g}_{\boldsymbol{b}_1} = \boldsymbol{u}^t \frac{\partial \boldsymbol{z}}{\partial \boldsymbol{b}_1} = \boldsymbol{u}^t$$ el sesgo de la capa oculta

```
J_haty = haty-y;                          print("J_haty =", round(J_haty, 4))
J_V = np.outer(h, J_haty);                print("J_V =", J_V)
J_b2 = J_haty;                            print("J_b2 =", round(J_b2, 4))
J_h = J_haty * V;                         print("J_h =", J_h)
J_z = J_h * sigmoid(z) * sigmoid(-z);     print("J_z =", J_z)
J_W = np.outer(x, J_z);                   print("J_W =", J_W)
J_b1 = J_z;                               print("J_b1 =", J_b1);
```

```
J_haty = 2.2215
J_V = [[0.1054]
 [0.5975]]
J_b2 = 2.2215
J_h = [-2.2215  2.2215]
J_z = [-0.1004  0.4368]
J_W = [[-0.2007  0.8736]
 [-0.2007  0.8736]]
J_b1 = [-0.1004  0.4368]
```

3. (0.25 puntos) *Actualización de parámetros:* actualiza las transformaciones lineales y sesgos del modelo a partir de las Jacobianas halladas, con factor de aprendizaje $\eta = 1$.

```
V  = V  - 1.0 * J_V.T; print("V =", V)
b2 = b2 - 1.0 * J_b2;  print("b2 =", b2)
W  = W  - 1.0 * J_W.T; print("W =", W)
b1 = b1 - 1.0 * J_b1;  print("b1 =", b1)
```

```
V = [[-1.1054  0.4025]]
b2 = -1.2215
W = [[-0.7993 -0.7993]
 [ 0.1264 -1.8736]]
b1 = [ 1.1004 -1.4368]
```