Apuntes de la Clase

Apuntes Semana 6

Apuntes del 11 de setiembre de 2025

Sahid Rojas Chacón — 2018319311
Curso: Inteligencia Artificial
reds@estudiantec.cr

***Resumen*—En este documento, se resume la clase del 11 de setiembre de 2025, en la cuál se realizó primeramente un repaso de lo visto en la clase anterior. De manera general, este documento recopila información sobre verosimilitud en la regresión logística, la función de costo e información sobre un notebook de regresión logística compartido por el profesor.**

***Index Terms*—Verosimilitud, regresión logística, gradiente descendiente, función sigmoide, derivada.**

I. Nota sobre Tarea I

Se recuerda otorgar la debida importancia al informe escrito y a su documentación, por cuanto constituirán la base de la retroalimentación para entregas posteriores y para las etapas subsiguientes del proyecto. El informe deberá presentar con claridad los objetivos, la metodología y los resultados, asegurar la reproducibilidad (datos, código, semillas y versiones), e incluir instrucciones de ejecución suficientes y verificables, manteniendo coherencia entre texto, figuras y conclusiones.

II. Verosimilitud: idea y función para el dataset

II-A. Qué es verosimilitud

Verosimilitud es: dado un conjunto de datos y un modelo con parámetros, ¿qué tan probable es observar *esos* datos bajo *esos* parámetros? En binario, nuestro modelo $f_{w,b}(x)$ devuelve una probabilidad en $(0, 1)$ y la verosimilitud del dataset se construye multiplicando las probabilidades individuales.

II-B. Modelo y notación

Usaré $f_{w,b}(x) = \sigma(w^\top x + b)$, donde σ es la sigmoide:

$$\sigma(t) = \frac{1}{1 + e^{-t}}.$$

II-C. Verosimilitud del conjunto

Para datos $\{(x_i, y_i)\}_{i=1}^N$ con $y_i \in \{0, 1\}$:

$$L(w, b) = \prod_{i=1}^{N} f_{w,b}(x_i)^{\,y_i} \left(1 - f_{w,b}(x_i)\right)^{\,1-y_i}. \tag{1}$$

II-D. Ejemplo narrativo: calabaza/naranja

La misma fórmula (1) explica los dos casos:

$$y_i = 1 \Rightarrow f_{w,b}(x_i)^1 \big(1 - f_{w,b}(x_i)\big)^0 = f_{w,b}(x_i), \tag{2}$$

$$y_i = 0 \Rightarrow f_{w,b}(x_i)^0 \big(1 - f_{w,b}(x_i)\big)^1 = 1 - f_{w,b}(x_i). \tag{3}$$

Caso "no es naranja" (clase 0).: Si $wx + b = 1{,}458$, entonces $\sigma(1{,}458) \approx 0{,}81$; como aquí me interesa que *no sea naranja* (y_i=0), la probabilidad es $1 - \sigma(1{,}458) \approx 0{,}19$. Para el evento complementario (*no naranja* como etiqueta positiva en esa formulación), se reportó $0{,}81$; el punto es que el mismo f sirve para ambos casos cambiando y_i.

Caso "sí es naranja" (clase 1).: Si $wx + b = -1{,}32$, entonces $\sigma(-1{,}32) \approx 0{,}21$; la probabilidad de *sí ser naranja* (clase 1) se obtiene con $1 - \sigma(-1{,}32) \approx 0{,}79$. Estos números ilustran cómo interpretar $f(x)$ en los dos escenarios.

III. Por qué metemos logaritmos

Multiplicar muchas probabilidades puede complicar la derivada y además es numéricamente inestable. Usamos identidades de logaritmos:

$$\ln(a^n) = n \ln a, \qquad \ln(ab) = \ln a + \ln b,$$

para convertir el producto en suma. Aplicando $\ln$ a (1):

$$\ln L(w, b) = \sum_{i=1}^{N} \big[y_i \ln f_{w,b}(x_i) + (1 - y_i) \ln(1 - f_{w,b}(x_i))\big]. \tag{4}$$

Esto se llama *log-likelihood* y es mucho más amigable para derivar.

IV. De maximizar a minimizar: negando el log-likelihood

En entrenamiento solemos *minimizar*. Como maximizar (4) es equivalente a minimizar su opuesto, definimos la pérdida logística promedio:

$$\mathcal{L}(w, b) = -\frac{1}{N} \sum_{i=1}^{N} \Big[y_i \ln f_{w,b}(x_i) + (1 - y_i) \ln\big(1 - f_{w,b}(x_i)\big)\Big]. \tag{5}$$

Nota : esta pérdida es ≥ 0 en práctica; si te da negativa, probablemente hay un bug de signos o promedios. El objetivo es empujarla hacia cero.

V. Qué parámetros sí podemos actualizar

Los parámetros *libres* del modelo son w (vector de pesos) y b (sesgo). Todo lo demás depende de ellos. Por eso, necesitamos $\partial\mathcal{L}/\partial w$ y $\partial\mathcal{L}/\partial b$ para poder aplicar descenso por gradiente.

VI. Composición de funciones y regla de la cadena

Primero reescribo el modelo de forma explícita como composición:

$$z(x) = w^\top x + b, \qquad a(z) = \sigma(z), \qquad f_{w,b}(x) = a(z(x)).$$

La pérdida *por muestra* (sin el promedio y con el signo negativo puesto) queda:

$$L = -\Big[y \,\ln a(z(x)) + (1-y)\,\ln\big(1 - a(z(x))\big)\Big].$$

Usamos regla de la cadena:

$$\frac{\partial L}{\partial w} = \frac{\partial L}{\partial a} \cdot \frac{\partial a}{\partial z} \cdot \frac{\partial z}{\partial w}, \tag{6}$$

$$\frac{\partial L}{\partial b} = \frac{\partial L}{\partial a} \cdot \frac{\partial a}{\partial z} \cdot \frac{\partial z}{\partial b}. \tag{7}$$

VI-A. Paso 1: derivada de L respecto a a

Derivando los logaritmos (regla de la cadena incluida):

$$\begin{aligned}\frac{\partial L}{\partial a} &= -\Big[\, y \cdot \frac{1}{a} \cdot (a)' \;+\; (1-y) \cdot \frac{1}{1-a} \cdot (1-a)' \,\Big] \\ &= -\Big[\, y \cdot \frac{1}{a} \cdot 1 \;+\; (1-y) \cdot \frac{1}{1-a} \cdot (-1) \,\Big] \\ &= -\frac{y}{a} + \frac{1-y}{1-a}. \end{aligned} \tag{8}$$

VI-B. Paso 2: derivada de a respecto a z

La derivada de la sigmoide es la famosa *forma cerrada*:

$$\frac{\partial a}{\partial z} = \sigma(z)\big(1 - \sigma(z)\big) = a(1-a). \tag{9}$$

VI-C. Paso 3: derivadas de z respecto a w y b

De $z(x) = w^\top x + b$:

$$\frac{\partial z}{\partial w} = x, \qquad \frac{\partial z}{\partial b} = 1. \tag{10}$$

VI-D. Juntando todo: primero $\partial L / \partial z$

Es útil agrupar $\frac{\partial L}{\partial z} = \frac{\partial L}{\partial a} \cdot \frac{\partial a}{\partial z}$ y luego propagar a w y b:

$$\begin{aligned}\frac{\partial L}{\partial z} &= \Big(-\frac{y}{a} + \frac{1-y}{1-a} \Big) \cdot a(1-a) \\ &= -y(1-a) + (1-y)a \\ &= a - y. \end{aligned} \tag{11}$$

VI-E. Gradientes finales de w y b

Usando (11) y (10):

$$\frac{\partial L}{\partial w} = (a-y)\,x, \tag{12}$$

$$\frac{\partial L}{\partial b} = (a-y). \tag{13}$$

Promediando sobre el dataset (dividiendo entre N y sumando en i) recuperamos las expresiones habituales de la pérdida promedio (5).

VII. Actualización de parámetros (descenso por gradiente)

Con *learning rate* $\alpha > 0$:

$$w \leftarrow w - \alpha \,\frac{1}{N} \sum_{i=1}^{N} \big(a_i - y_i\big) x_i, \tag{14}$$

$$b \leftarrow b - \alpha \,\frac{1}{N} \sum_{i=1}^{N} \big(a_i - y_i\big). \tag{15}$$

(*Nota :* si la curva de pérdida sube o se vuelve errática, probá reducir α.)

VIII. Código

En esta sección explico únicamente el *código fuente* y cómo implementa la teoría vista en clase: verosimilitud → log-likelihood → log-loss, derivadas por regla de la cadena y actualización de los parámetros w y b. Las Figuras 1–3 corresponden a tres capturas del archivo `main.py`.

VIII-A. Definiciones del modelo: clase, sigmoide y pérdida (`cap01`)

La Figura 1 muestra la clase `LogisticRegressionAI`. En el constructor (`__init__`) se fijan los **hiperparámetros** `lr` (tasa de aprendizaje) y `epochs` (épocas de entrenamiento), y se inicializan los **parámetros entrenables** $w \in \mathbb{R}^d$ y $b \in \mathbb{R}$, que son los únicos valores que el algoritmo ajusta.

El método `sigmoid` implementa la activación logística

$$\sigma(z) = \frac{1}{1 + e^{-z}}, \quad z = w^\top x + b,$$

que mapea la combinación lineal z a una probabilidad $a = \sigma(z) \in (0, 1)$.

La función `binary_cross_entropy_loss` codifica la **log-loss** (negativa del log-likelihood promedio):

$$\begin{aligned}\mathcal{L}(w,b) &= -\frac{1}{N} \sum_{i=1}^{N} \big[\, y_i \ln a_i + (1-y_i) \ln(1-a_i) \big], \\ a_i &= \sigma\big(w^\top x_i + b\big). \end{aligned} \tag{16}$$

Antes de aplicar los logaritmos, el código realiza *clipping* con $\varepsilon = 10^{-15}$ para evitar $\log(0)$ y mejorar la estabilidad numérica. Minimizar (16) es equivalente a maximizar la verosimilitud $\prod_i a_i^{y_i}(1-a_i)^{1-y_i}$.

VIII-B. Entrenamiento vectorizado y predicción (`cap02`)

La Figura 2 concentra el corazón del aprendizaje: el método `fit` (bucle de entrenamiento) y `predict` (inferencia).

Forward (vectorizado).: Con $X \in \mathbb{R}^{N \times d}$ y $w \in \mathbb{R}^d$, se calcula

$$z = Xw + b\mathbf{1}, \qquad a = \sigma(z),$$

usando multiplicación matricial de `numpy` (`np.dot`). Este paso implementa la composición $x \to z \to a$ vista en clase.

```python
class LogisticRegressionAI:

    def __init__(self, lr=0.0001, epochs=1000):
        self.epochs = epochs
        self.lr = lr
        self.w = None
        self.b = None

    # activación sigmoide
    def sigmoid(self, x):
        return 1 / (1 + np.exp(-x))

    # función de costo (log-loss)
    def binary_cross_entropy_loss(self, y_true, y_pred):
        epsilon = 1e-15  # evita log(0)
        y_pred = np.clip(y_pred, epsilon, 1 - epsilon)
        loss = (-1 / len(y_true)) * np.sum(
            y_true * np.log(y_pred) + (1 - y_true) * np.log(1 - y_pred)
        )
        return loss
```

Figura 1. **cap01:** Clase `LogisticRegressionAI`. Hiperparámetros (`lr`, `epochs`), parámetros w, b, sigmoide y pérdida (16).

```python
    def fit(self, X, y):
        n_samples, n_features = X.shape
        # z = Xw + b
        self.w = np.random.rand(n_features)
        self.b = 0.0

        for epoch in range(self.epochs):
            # forward
            z = np.dot(X, self.w) + self.b
            a = self.sigmoid(z)

            # gradientes promedio
            dw = (1 / n_samples) * np.dot(X.T, (a - y))
            db = (1 / n_samples) * np.sum(a - y)

            # actualización
            self.w = self.w - self.lr * dw
            self.b = self.b - self.lr * db

            # logging (espaciado para que no sature la terminal)
            if epoch % 1000 == 0:
                loss = self.binary_cross_entropy_loss(y, a)
                print(f"[fit] epoch={epoch:4d}  loss={loss:.4f}")

    def predict(self, X):
        z = np.dot(X, self.w) + self.b
        a = self.sigmoid(z)
        return (a >= 0.5).astype(int)
```

Figura 2. **cap02:** `fit` (forward $z \to a$, gradientes `dw`/`db`, actualización) y `predict` (umbral 0,5).

Gradientes (regla de la cadena).: De la derivación teórica se obtiene

$$\frac{\partial \mathcal{L}}{\partial z} = a - y, \quad \frac{\partial \mathcal{L}}{\partial w} = \frac{1}{N} X^\top (a - y), \quad \frac{\partial \mathcal{L}}{\partial b} = \frac{1}{N} \mathbf{1}^\top (a - y).$$

En el código aparecen como `dw = (1/n_samples) * X.T @ (a - y)` y `db = (1/n_samples) * np.sum(a - y)`.

Actualización (descenso por gradiente).: Con tasa $\alpha =$ `lr`:

$$w \leftarrow w - \alpha \frac{\partial \mathcal{L}}{\partial w}, \qquad b \leftarrow b - \alpha \frac{\partial \mathcal{L}}{\partial b},$$

que en el código se implementa como `w = w - lr * dw` y `b = b - lr * db`. Al disminuir (16) se incrementa la verosimilitud del modelo.

Predicción.: El método `predict` repite $z = Xw + b$ y $a = \sigma(z)$ y umbraliza con 0,5 para devolver etiquetas binarias $\hat{y} \in \{0, 1\}$, consistente con decidir por la clase más probable.

```python
acc_manual = accuracy_score(y_test, y_hat)
print("\n--- RESULTADOS MODELO DESDE CERO ---")
print(f"Accuracy: {acc_manual:.2f}")
print(classification_report(y_test, y_hat, zero_division=0))

acc_sk = accuracy_score(y_test, y_pred)
print("\n--- RESULTADOS SKLEARN ---")
print(f"Accuracy: {acc_sk:.2f}")
print(classification_report(y_test, y_pred, zero_division=0))
```

Figura 3. **cap03:** Flujo completo: imports, generación del dataset con `make_classification`, división entrenamiento/prueba, entrenamiento del modelo propio y bloque análogo con `sklearn`.

VIII-C. Bloque principal: imports, dataset y flujo del entrenamiento (`cap03`)

Como se muestra en la Figura 3, se realiza la importación de librerías necesarias (`numpy` y módulos de `sklearn`). Luego, se crea un **dataset de clasificación** con `make_classification`, que genera datos sintéticos controlados para problemas binarios. En este caso, se indica:

$$\texttt{n\_samples} = 1000, \quad \texttt{n\_features} = 2, \quad \texttt{n\_informative} = 2,$$

Esto produce dos *features* informativas sin redundancia y un solo clúster por clase, coherente con los ejemplos del curso.

A continuación, se realiza la **división entrenamiento/prueba** con `train_test_split`, manteniendo la proporción de clases mediante `stratify=y` y fijando `random_state` para reproducibilidad. Seguidamente, se **instancia y entrena** el modelo implementado desde cero:

```
modelo_manual = LogisticRegressionAI(lr=0.001, epo
```

ejecutando el bucle explicado en la subsección anterior (forward, gradientes y actualización). Finalmente, el bloque incluye `LogisticRegression` de `sklearn` para **replicar el mismo enfoque** con la librería estándar, lo que sirve como referencia y valida que la implementación manual respeta la teoría.

VIII-D. Resumen código ↔ teoría

Verosimilitud → log-loss. El código minimiza (16), que es $-\log L$; así, "minimizar la pérdida" equivale a "maximizar la verosimilitud".
Regla de la cadena. La ruta $x \to z \to a \to \mathcal{L}$ da $\frac{\partial \mathcal{L}}{\partial z} = a - y$, base de los gradientes vectorizados `dw` y `db`.
Parámetros actualizables. Solo w y b cambian; el resto (sigmoide, datos) son transformaciones/entradas fijas conforme a la formulación del modelo.

IX. Conclusiones

La clave para no perderse es mirar la composición $x \to z \to a \to L$ y empujar las derivadas con la regla de la cadena. El uso de logaritmos cambia productos por sumas y, al negar el log-likelihood, pasamos a minimizar una función estable y derivable. Con los gradientes compactos $(a - y)x$ y $(a - y)$, actualizar w y b se vuelve mecánico con descenso por gradiente.

APÉNDICE: FÓRMULAS RELEVANTES

Sigmoide y derivada:

$$\sigma(t) = \frac{1}{1+e^{-t}}, \qquad \sigma'(t) = \sigma(t)\big(1-\sigma(t)\big).$$

Verosimilitud y log-likelihood:

$$L = \prod_i f(x_i)^{y_i}\big(1-f(x_i)\big)^{1-y_i}, \quad \ln L = \sum_i \big[y_i \ln f(x_i) + (1-y_i)\ln(1-f(x_i))\big].$$

Pérdida logística promedio (lo que minimizo):

$$\mathcal{L}(w,b) = -\frac{1}{N}\sum_i \Big[y_i \ln f(x_i) + (1-y_i)\ln\big(1-f(x_i)\big)\Big].$$

Gradientes (por muestra):

$$\frac{\partial L}{\partial a} = -\frac{y}{a} + \frac{1-y}{1-a}, \quad \frac{\partial a}{\partial z} = a(1-a), \quad \frac{\partial z}{\partial w} = x, \quad \frac{\partial z}{\partial b} = 1,$$

$$\frac{\partial L}{\partial z} = a - y, \quad \frac{\partial L}{\partial w} = (a-y)x, \quad \frac{\partial L}{\partial b} = (a-y).$$