p3_01

December 4, 2025

1 Práctica de aprendizaje automático (parte 1)

2 Teorema de Bayes y Naïve Bayes (1.5 puntos)

2.1 1. Teorema de Bayes

El **Teorema de Bayes**:

$$P(H_1 \mid D, I) = \frac{P(D \mid H_1, I) \times P(H_1 \mid I)}{P(D \mid I)} \quad \text{con} \quad P(D \mid I) = \sum_i P(D \mid H_i, I) \times P(H_i \mid I).$$

- H_1 = Hipótesis 1, D = datos, I = información.
- $P(H_1 \mid I)$: Probabilidad a *priori* de la hipótesis 1 (antes de observar los datos).
- $P(D \mid I)$: Evidencia de los datos.
- $P(D \mid H_1, I)$: *Verosimilitud* de la hipótesis 1 dados los datos.
- $P(H_1 \mid D, I)$: Probabilidad a *posteriori* de la hipótesis 1 (después de observar los datos).

2.1.1 Ejercicio 1 - Palabra sospechosa en correo

Vamos a calcular la probabilidad de que un correo sea **SPAM** dadas las palabras **"gratis"** y **"oferta"**. Supongamos que dado un nuevo correo, la probabilidad a priori de que sea **SPAM** es del **20%**.

Si aparece la palabra **"gratis"**: - P("gratis" | SPAM) = 0.4 - P("gratis" | ¬SPAM) = 0.05 - Si aparece la palabra **"oferta"**: - P("oferta" | SPAM) = 0.7 - P("oferta" | ¬SPAM) = 0.05

1. Calcula P(SPAM | "gratis").
2. ¿Qué ocurre si además ves la palabra **"oferta"** y asumes independencia? (Pista: multiplica verosimilitudes.)

Completa y ejecuta la celda de abajo. Además, justifica razonadamente el resultado que obtengas.

Apartado 1 Para calcular $P(SPAM|\text{"}gratis\text{"}, I)$, utilizamos la fórmula del Teorema de Bayes. De modo que llamamos H_1 a la hipótesis de SPAM. Y D es que aparezca la palabra "gratis". Luego

$$P(SPAM|''gratis'', I) = \frac{P(\text{"}gratis\text{"}|SPAM, I) \times P(SPAM, I)}{P(\text{"}gratis\text{"}|I)}$$

A su vez, sabemos que $P(\text{"}gratis\text{"}|I) = \sum P(\text{"}gratis\text{"}|H_i, I) \times P(H_1|I)$. Como en este caso, solo tenemos H_1 y H_2, siendo H_2 que el correo no sea SPAM, entonces

$$P(\text{"}gratis\text{"}, I) = P(\text{"}gratis\text{"}|SPAM, I) \times P(SPAM, I) + P(\text{"}gratis\text{"}|NO\_SPAM) \times P(NO\_SPAM, I)$$

Con todo lo anterior, calculamos la probabilidad.

Apartado 2 Por otro lado, en el caso de ver tambień en el correo la palabra "oferta", debemos tratarlas, como nos especifica el enunciado como variables independientes. Luego calcularemos la verosimilitud como la multiplicación de ambas:

$$P(\text{"}oferta\text{"},\text{"}\ gratis\text{"}|SPAM, I) = P(\text{"}oferta\text{"}|SPAM, I) \cdot P(\text{"}gratis\text{"}|SPAM, I)$$

Luego, la probabilidad a posteriori vendrá dada por:

$$P(SPAM|\text{"}oferta\text{"},\text{"}\ gratis\text{"}, I) = \frac{P(\text{"}oferta\text{"},\text{"}\ gratis\text{"}|SPAM, I) \times P(SPAM; I)}{P(\text{"}oferta\text{"},\text{"}\ gratis\text{"}, I)}$$

con

$$P(\text{"}oferta\text{"},\text{"}\ gratis\text{"}, I) = P(\text{"}oferta\text{"},\text{"}\ gratis\text{"}|SPAM, I) \cdot P(SPAM, I) + P(\text{"}oferta\text{"},\text{"}\ gratis\text{"}|NO\_SPAM) \cdot P(N$$

[1]:
```
p_spam = 0.2
p_no_spam = 1 - p_spam

p_gratis_spam = 0.4
p_gratis_no_spam  = 0.05
p_oferta_spam = 0.7
p_oferta_no_spam  = 0.05

# Cálculo del apartado 1
p_gratis = p_gratis_no_spam * p_no_spam + p_gratis_spam*p_spam
posterior_gratis = (p_gratis_spam * p_spam)/p_gratis

# Cálculo del apartado 2
p_gratis_oferta_spam = p_oferta_spam * p_gratis_spam
p_gratis_oferta_no_spam = p_oferta_no_spam * p_gratis_no_spam
p_gratis_oferta = p_gratis_oferta_spam * p_spam + p_gratis_oferta_no_spam *␣
 ↪p_no_spam
posterior_gratis_oferta = (p_gratis_oferta_spam * p_spam)/p_gratis_oferta

print(f"P(SPAM | 'gratis') = {posterior_gratis:.6f}  (~{100*posterior_gratis:.
 ↪2f}%)")
print(f"P(SPAM | 'gratis' y 'oferta') = {posterior_gratis_oferta:.6f} ␣
 ↪(~{100*posterior_gratis_oferta:.2f}%)")
```

```
P(SPAM | 'gratis') = 0.666667  (~66.67%)
P(SPAM | 'gratis' y 'oferta') = 0.965517  (~96.55%)
```

2.2 Ejercicio 2 - Probabilidad de jugar profesionalmente al baloncesto

Ahora vamos a calcular la probabilidad de que una persona juegue profesionalmente al baloncesto dada su altura y su edad. La *probabilidad a priori* de que una persona sea **PRO** es del **1%**: $P(\text{PRO}) = 0.01$. Asumiremos **independencia condicional** entre *edad* y *altura* dada la clase (al asumir independencia estamos calculando "Naïve Bayes"). Además, consideraremos que las características se modelan con distribuciones **gaussianas**.

Estas son las verosimilitudes:

- Si es **PRO**
  edad $\sim \mathcal{N}(\mu{=}27,\ \sigma^2{=}3^2)$,
  altura (m) $\sim \mathcal{N}(\mu{=}2.00,\ \sigma^2{=}0.07^2)$.
- Si es **NO_PRO**
  edad $\sim \mathcal{N}(\mu{=}35,\ \sigma^2{=}10^2)$,
  altura (m) $\sim \mathcal{N}(\mu{=}1.72,\ \sigma^2{=}0.06^2)$.

Calcula $P(\text{PRO} \mid \text{edad}, \text{altura})$ para:

1) Persona A: **edad = 25**, **altura = 1.95 m**.

2) Persona B: **edad = 30**, **altura = 1.90 m**.

   Pista: como **asumimos independencia condicional**, la verosimilitud conjunta se obtiene **multiplicando** las densidades de cada característica. $\text{score}(\text{PRO}) = P(\text{PRO}) \cdot \mathcal{N}(\text{edad}) \cdot \mathcal{N}(\text{altura})$.
   $\text{Posterior} = \dfrac{\text{score}(\text{PRO})}{\text{score}(\text{PRO}) + \text{score}(\text{NO\_PRO})}$.

Completa y ejecuta la celda de abajo. Además, justifica razonadamente el resultado que obtengas.

Apartado 1 En este apartado tenemos que calcular $P(\text{PRO} \mid \text{edad, altura})$ para una persona A, con edad 25 años y altura 1.95 m. Para calcularla, utilizamos la fórmula de probabilidad a posteriori dada por:

$$P(\text{PRO} \mid \text{edad, altura}) = \frac{\text{score}(\text{PRO})}{\text{score}(\text{PRO}) + \text{score}(\text{NO\_PRO})}$$

Donde $\text{score}(\text{PRO}) = P(\text{PRO}) \cdot \mathcal{N}(\text{edad}) \cdot \mathcal{N}(\text{altura})$. Más en concreto, $\mathcal{N}(\text{X})$ es la probabilidad de la gaussiana correpondiente al suceso X de media y varianzas especificadas en el enunciado. Para obtener dicha probabilidad, hacemos uso de la función que nos facilita la práctica `gaussian_pdf(x,mu,var)` donde x es el suceso del que queremos saber la probabilidad, mu es la media y var la varianza la la distribución gaussiana.

Apartado 2 Este apartado es hacer lo mismo que en el anterior, pero para una persona B con edad 30 años y altura 1.9 metros.

[36]:
```
import numpy as np

# Prior
p_pro = 0.01
```

```python
p_no_pro = 1 - p_pro

# Parámetros (medias y varianzas)
mu_edad_pro, var_edad_pro = 27, 3**2
mu_alt_pro,  var_alt_pro  = 2.00, 0.07**2

mu_edad_no, var_edad_no = 35, 10**2
mu_alt_no,  var_alt_no  = 1.72, 0.06**2

# Candidatos
edad_A, alt_A = 25, 1.95
edad_B, alt_B = 30, 1.90

def gaussian_pdf(x, mu, var):
    return 1/np.sqrt(2*np.pi*var) * np.exp(-(x-mu)**2/(2*var))


# Cáclulo de apartado 1
score_proA = p_pro * gaussian_pdf(edad_A, mu_edad_pro, var_edad_pro) *␣
 ↪gaussian_pdf(alt_A, mu_alt_pro, var_alt_pro)
score_no_proA = p_no_pro * gaussian_pdf(edad_A, mu_edad_no, var_edad_no) *␣
 ↪gaussian_pdf(alt_A, mu_alt_no, var_alt_no)
posterior_personaA = score_proA / (score_proA + score_no_proA)


# Cálculo de apartado 2

score_proB = p_pro * gaussian_pdf(edad_B, mu_edad_pro, var_edad_pro) *␣
 ↪gaussian_pdf(alt_B, mu_alt_pro, var_alt_pro)
score_no_proB = p_no_pro * gaussian_pdf(edad_B, mu_edad_no, var_edad_no) *␣
 ↪gaussian_pdf(alt_B, mu_alt_no, var_alt_no)
posterior_personaB = score_proB / (score_proB + score_no_proB)

print(f"P(PRO | A) = {posterior_personaA:.6f} (~{100*posterior_personaA:.2f}%)")
print(f"P(PRO | B) = {posterior_personaB:.6f} (~{100*posterior_personaB:.2f}%)")
```

```
P(PRO | A) = 0.978639 (~97.86%)
P(PRO | B) = 0.391573 (~39.16%)
```

2.3 2. Clasificador Gaussian Naïve Bayes

Ahora vamos a implementar un clasificador basado en Naïve Bayes, que solo acepta características continuas. Para ello, la única librería que emplearemos será `numpy`. El clasificador **Gaussian Naïve Bayes** asume que las características son **independientes entre sí dada la clase** (por esa razón se llama "naïve") y modela cada característica con una **Gaussiana**. > **Nota:** Existen variantes de Naïve Bayes que emplean otras distribuciones según el tipo de dato (p. ej., Multinomial, Bernoulli), y también para datos continuos pueden usarse alternativas a la gaussiana.

El conjunto de datos de entrada es $\mathcal{D} = \{\mathbf{X}, \mathbf{y}\}$, donde $\mathbf{X} = \{\mathbf{x}_1, ..., \mathbf{x}_N\}$, $\mathbf{y} = \{y_1, \dots, y_N\}$, N es el número de datos, $\mathbf{x}_i = (x_{i,1}, \dots, x_{i,D})$ y D es el número de características.

2.3.1 `fit(X, y)`

Método para ajustar el clasificador (realiza el entrenamiento). Primero estima los **priores** de clase $P(y{=}k)$ como la frecuencia de cada clase en el conjunto de entrenamiento. Después, para cada clase k y característica j, calcula la **media** $\mu_{j,k}$ y la **varianza** $\sigma^2_{j,k}$ usando únicamente los datos de esa clase.

- **Prior (frecuencia relativa):**

$$P(y{=}k) = \frac{n_k}{N}.$$

  (donde n_k es el número de ejemplos de la clase k y N el total).

  Nota: añade un ε pequeño a cada $\sigma^2_{j,k}$ para evitar varianzas nulas.

- **Salida:** el clasificador entrenado con las clases, los priors $\{P(y{=}k)\}$ y las matrices de medias μ y varianzas σ^2 de tamaño $(K \times D)$.

2.3.2 `predict(X)`

Método de predicción. Para cada nuevo dato x_i, calcula la *probabilidad a posteriori* de cada clase $P(y{=}k \mid x_i)$ usando el prior y el producto de densidades Gaussianas por característica. Termina prediciendo la clase con mayor probabilidad.

- **Puntuación por clase:**

$$\text{score}(k) = P(y{=}k) \prod_{j=1}^{D} \mathcal{N}(x_j;\, \mu_{j,k},\, \sigma^2_{j,k}).$$

- **Salida:** la predicción final es la clase que tenga mayor score (probabilidad).

$$y = \arg\max_k \ \text{score}(k).$$

[118]:
```
# Cargamos librerias
import numpy as np
from sklearn.datasets import load_iris
```

[119]:
```
# 1) Cargar datos
X_all, y_all = load_iris(return_X_y=True)
np.random.seed(0)

# Dividimos el dataset entre train-test y validación (75-25)
indices = np.random.permutation(len(y_all))
cut = int(0.75 * len(y_all))
indices_train, indices_validation = indices[:cut], indices[cut:]
X_train, y_train = X_all[indices_train], y_all[indices_train]
X_validation, y_validation = X_all[indices_validation],␣
 ↪y_all[indices_validation]
```

[143]:
```
class GaussianNaiveBayes:

    def __init__(self, eps=1e-8):
        self.eps = eps

    def fit(self, X, y):
        """
        Función que establece los datos dado un set de datos de entrenamiento
        Args:
            X: set de datos
            y: set de labels
        """
        # Vemos qué clases distintas hay
        self.classes = np.unique(y)

        # Guardamos los datos del tamaño, num_obs es el número de␣
↪observaciones, num_data número de datos y
        # num_classes es el número de clases distintas
        num_obs, num_data = X.shape
        num_classes = len(self.classes)

        self.priors = np.zeros(num_classes)
        self.mu = np.zeros((num_classes, num_data))
        self.var  = np.zeros((num_classes, num_data))

        # Calculamos la probabilidad, como frecuencia del set de datos de␣
↪entrenamiento
        # A su vez, con los datos de la etiqueta i, calculamos la media y la␣
↪varianza.
        for i, cl in enumerate(self.classes):
            X_i = X[y==cl]
            self.priors[i]  = len(X_i)/num_obs
            self.mu[i]      = X_i.mean(axis=0)
            self.var[i]     = X_i.var(axis=0) + self.eps

        return self

    def gaussian_pdf(self, x, mu, var):
        """
        Función que devuelve la probabilidad de x dada una distribución␣
↪gaussiana de media mu y varianza var
        """
        return 1/np.sqrt(2*np.pi*var) * np.exp(-(x-mu)**2/(2*var))

    def predict(self, X):
        """
        Función que predice las etiquetas de unos datos X
```

```
        """
        num_obs, _ = X.shape
        num_classes = len(self.classes)
        scores = np.zeros((num_obs, num_classes))

        # Iteramos por todas las observaciones
        for i in range(num_obs):

            # Luego por todas las etiquetas posibles, para calcular el score
            for j in range(num_classes):

                # Como pone en el enunciado, el score del dato x para la␣
↪etiqueta j es la probabilidad a priori de la etiqueta i
                # por el producto de las probabilidades de la distribución␣
↪gaussiana de cada elemento x_i dada la media y varianza
                # del elemento i observado en los datos de entrenamiento
                probabilities = np.prod(self.gaussian_pdf(X[i], self.mu[j],␣
↪self.var[j]))
                scores[i][j] = self.priors[j] * probabilities

        # Por último, devolvemos aquel label que maximize las probabilidades␣
↪para cada elemento de los datos
        return self.classes[np.argmax(scores, axis=1)]
```

[144]:
```
# Entrenar y validar
gnb = GaussianNaiveBayes().fit(X_train, y_train)
y_pred = gnb.predict(X_validation)
print("Precisión (validación en el 25% restante):", (y_pred == y_validation).
↪mean())
```

```
Precisión (validación en el 25% restante): 0.9473684210526315
```