Probabilidad y Estadística para Inteligencia Artificial
Examen

Lautaro Delgado

22 de agosto de 2020

1. Ejercicio 1

Se tiene una proporción de fósforos defectuosos aleatoria de un 1 %. Se quiere por lo tanto encontrar la máxima cantidad de fósforos N que se pueden ensamblar en una misma caja de manera que la probabilidad de tener uno o más fósforos defectuosos sea menor a 0,5.

En primer lugar, se reconoce que se puede aproximar el problema con una distribución binomial de parámetro $p = 0{,}01$ donde N es desconocido y X representa la cantidad de fósforos defectuosos. El problema plantea lo siguiente:

$$P(X >= 1) < 0{,}5 \tag{1}$$
$$P(X >= 1) = 1 - P(X < 1) \tag{2}$$
$$1 - P(X < 1) = 1 - P(X = 0) \tag{3}$$
$$P(X = 0) > 0{,}5 \tag{4}$$

ya que X toma valores en los naturales.

Haciendo uso de la función de distribución de probabilidad binomial y teniendo en cuenta que $p = 0{,}01$ y $k = 0$ se tiene:

$$0{,}5 = \binom{N}{k} p^k (1-p)^{N-k} \tag{5}$$
$$0{,}5 = \binom{N}{0} 0{,}01^0 (1 - 0{,}01)^N \tag{6}$$
$$0{,}5 = 0{,}99^N \tag{7}$$
$$\ln 0{,}5 = N \ln 0{,}99 \tag{8}$$
$$N = \frac{\ln 0{,}5}{\ln 0{,}99} \tag{9}$$
$$N = 68{,}968 \tag{10}$$
$$N = 68 \tag{11}$$

Se toma $N = 68$ ya que N debe pertenecer a los naturales. Ya con todos los parámetros de la distribución binomial se puede verificar lo establecido en el enunciado:

$$P(X >= 1) < 0{,}4951 \tag{12}$$
$$P(X >= 1) < 0{,}5 \tag{13}$$

El cálculo del valor esperado y el desvío estándar es bastante directo y se desprende de la distribución binomial. Se sabe que para este tipo de distribución se verifica que:

$$\mu = Np \tag{14}$$
$$\sigma = Np(1-p) \tag{15}$$

Por lo tanto, los valores teóricos son:

$$\mu = 0{,}68 \tag{16}$$
$$\sigma = 0{,}8205 \tag{17}$$

El detalle de la implementación de la simulación se puede encontrar en el Anexo I, pero principalmente la idea es crear un vector de muestras suficientemente grande donde la cantidad de fósforos defectuosos mantenga la proporción de la consigna. Luego se genera un vector de $N = 1 : 100$ sobre el que se itera y se toman aleatoriamente N muestras del vector inicial. Se mide luego la proporción de falla para ese valor de N y si supera el valor buscado de 0.5 se finaliza la ejecución. Debido a la aleatoriedad, se suelen obtener resultados distintos pero generalmente se obtiene el valor teórico de $N = 68$. En cuanto a la simulación de la media y el desvío estándar, al conocer ya el valor de N, se realiza una serie de experimentos y en cada uno de ellos se genera un vector aleatorio de distribución binomial con parámetros $p = 0{,}01$ y $N = 68$. Se obtiene la media y el desvío de cada uno de los vectores y luego se toma el promedio para todos los experimentos, En ese caso se obtienen los siguientes resultados que se aproximan bastante a los teóricos calculados:

$$\mu_{simulada} = 0{,}6789 \tag{18}$$
$$\sigma_{simulada} = 0{,}8187 \tag{19}$$

2. Ejercicio 2

Partiendo de la función de densidad de probabilidad de la variable aleatoria X, se sabe que es necesario que la misma integre en su rango a 1. Haciendo uso de ello se puede obtener el valor de K:

$$f(x) = kx, 0 \leq x \leq 3 \tag{20}$$

Se procede de la siguiente manera:

$$\int_0^3 kx\,dx = k\frac{x^2}{2}\Big|_0^3 \tag{21}$$
$$\int_0^3 kx\,dx = k\frac{9}{2} = 1 \tag{22}$$
$$k = \frac{2}{9} \tag{23}$$

Para obtener el valor de x_1 que cumpla con la condición, debemos obtener la expresión de la $CDF(X)$ la cual nos será útil para los incisos siguientes:

$$F(x) = \int_0^x \frac{2}{9}t\,dt = \frac{2}{9}\frac{t^2}{2}\Big|_0^x \tag{24}$$
$$F(x) = \int_0^x \frac{2}{9}t\,dt = \frac{1}{9}x^2 \tag{25}$$

Por lo tanto, se tiene que:

$$P(X \leq x_1) = 0{,}1 \tag{26}$$
$$F(x_1) = 0{,}1 \tag{27}$$
$$\frac{1}{9}x_1^2 = 0{,}1 \tag{28}$$
$$x_1 = +\sqrt{0{,}9} \tag{29}$$

Para el siguiente inciso, se hace uso del cálculo anterior de la CDF. Se desea obtener x mediante:

$$x = F^{-1}(u) \tag{30}$$

donde $u \sim U(0,1)$ y F es la CDF de x. El rango donde la función F es biyectiva es $0 \leq x \leq 3$ por lo tanto se procede a calcular su inversa:

$$u = F(x) \tag{31}$$
$$u = \frac{1}{9}x^2 \tag{32}$$
$$x = \sqrt{3}u \tag{33}$$

El resultado de este inciso se utiliza para generar a partir de un vector de muestras uniformes $U(0,1)$, muestras de la distribución deseada. A partir de ello, se obtiene una aproximación de la *pdf* de la distribución haciendo uso de estimación de densidad de kernel con un kernel gaussiano. Nuevamente el detalle de la implementación en Python se puede observar en el Anexo I pero principalmente luego de generado el vector de la distribución, se genera un vector de variable independiente para el kernel gaussiano. Se suman las contribuciones del kernel en cada una de las muestras y así se obtiene la estimación.

En este caso en particular, se ha ensayado una estimación manual (sin implementación de librerías), realizando la suma de la contribución del kernel con un ancho de banda unitario. Se han comparado los resultados con lo obtenido por la librería de Python Scikit-Learn y luego con ésta última se ha intentado obtener cuál es el ancho de banda que minimiza el error cuadrático medio de la estimación de la PDF en función del ancho de banda.

En el caso de la estimación de kernel manual con $h = 1$ donde h es el ancho de banda se han obtenido el siguiente error y la curva que se puede observar en la Figura 1

$$MSE_{h=1} = 0{,}0312 \tag{34}$$

Para el caso de la estimación de kernel con la librería Scikit-Learn, se ha tomado un ancho de banda más cercano al óptimo, $h = 0{,}2$ y se ha obtenido la curva de la Figura 2 y el siguiente error:

$$MSE_{h=0,2} = 0{,}0081 \tag{35}$$

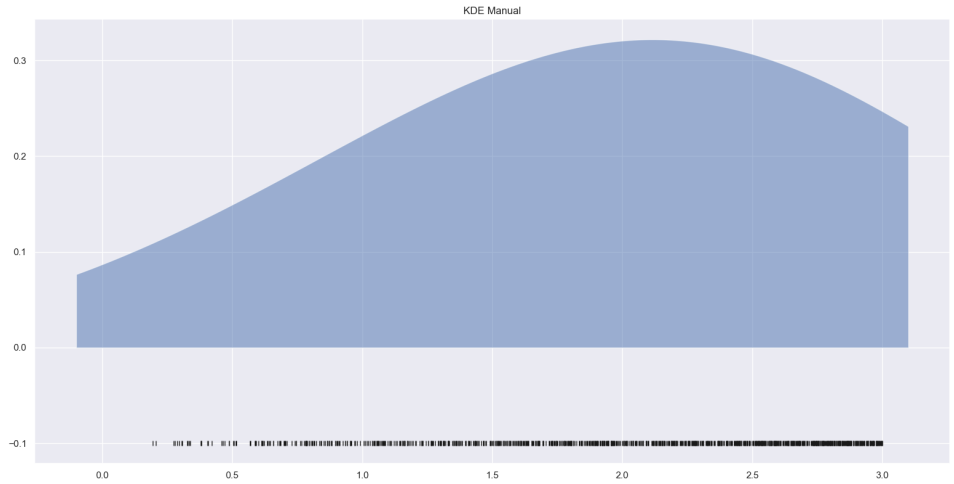

Figura 1: Estimación de Densidad de Kernel Manual con $h = 1$

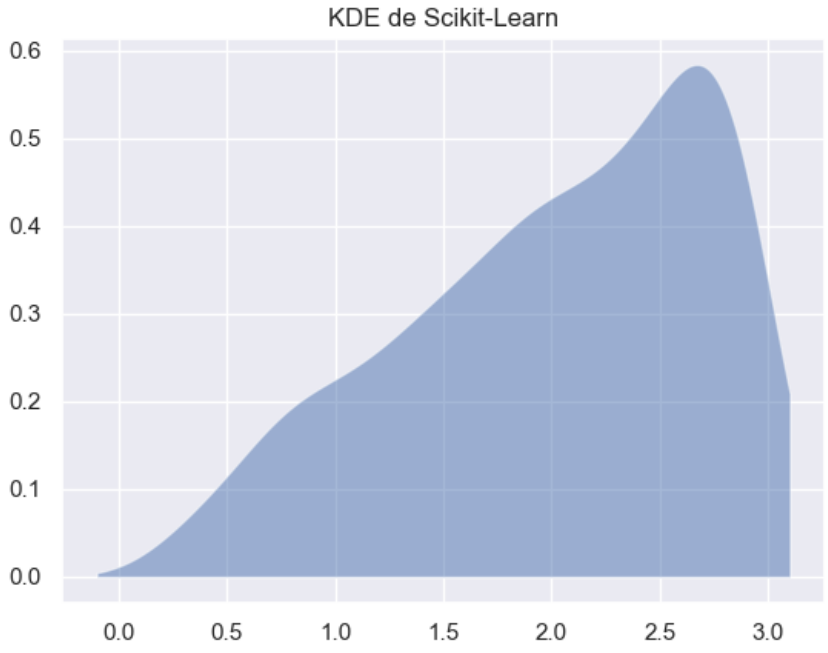

Figura 2: Estimación de Densidad de Kernel Scikit-Learn con $h = 0{,}2$

Por último, en la Figura 3, se observan los resultados de cómo evoluciona el error cuadrático medio en función de la elección del ancho de banda del KDE. En este caso el ancho de banda que miniza el

MSE y su error correspondiente son:

$$h_{optimo} = 0{,}1668 \tag{36}$$
$$MSE_{h_{opt}} = 0{,}0079 \tag{37}$$

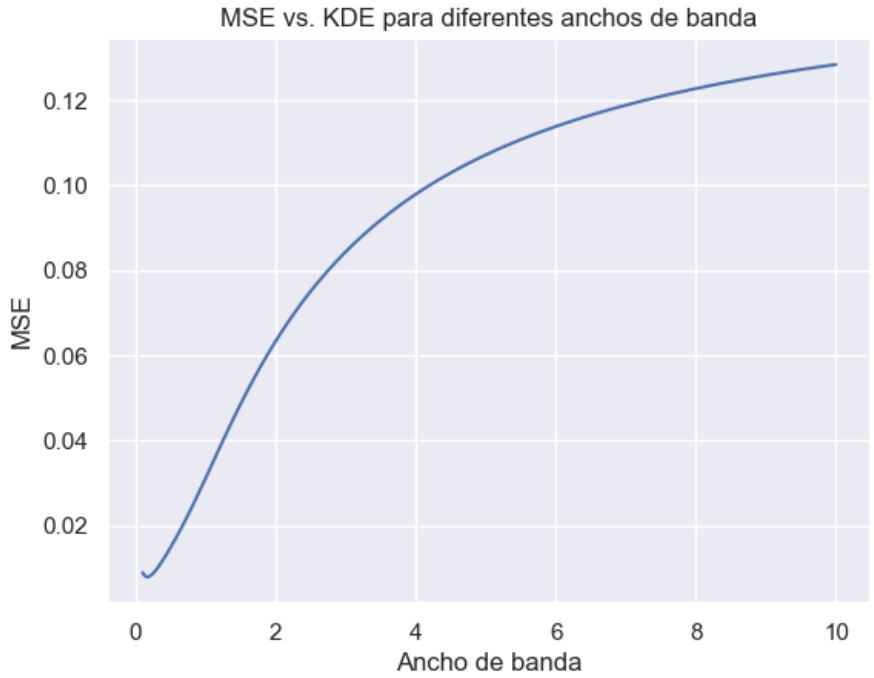

Figura 3: MSE de KDE en función de h

3. Ejercicio 3

Se tienen $N = 10$ muestras de una distribución normal con media μ desconocida, desvío estándar de $\sigma = 4$ y se obtiene una media muestral de $\hat{X} = 48$. Se desea obtener un intervalo de confianza del 95 % para μ. Por lo tanto se tiene los siguiente:

$$\hat{\mu} \sim N(\mu, \frac{\sigma^2}{N}) \tag{38}$$
$$\hat{\mu} \in (\mu_{min}, \mu_{max}) \tag{39}$$

Se normaliza para llevar el problema a una distribución normal de media 0 y desvío 1 (Z-score):

$$Z = \frac{\hat{\mu} - \mu}{\frac{\sigma}{\sqrt{N}}} \tag{40}$$
$$Z \sim N(0, 1) \tag{41}$$

Se conoce que los valores de Z que permiten obtener un intervalo de confianza bilateral del 95 % son $z_{min} = -1{,}96$ y $z_{max} = 1{,}96$. Por lo tanto, se plantea lo siguiente:

$$P(-1{,}96 < Z < 1{,}96) = 0{,}95 \tag{42}$$

$$P(\hat{\mu} - 1{,}96\frac{\sigma}{\sqrt{N}} < \mu < \hat{\mu} + 1{,}96\frac{\sigma}{\sqrt{N}}) = 0{,}95 \tag{43}$$

Con lo cual de la ecuación (43), reemplazando los datos del enunciado, se obtienen los siguientes valores límites de μ:

$$\mu_{min} = 45{,}5208 \tag{44}$$

$$\mu_{max} = 50{,}4792 \tag{45}$$

Para el segundo inciso, se plantea una hipótesis nula $H_0 : \mu = 45$ y una hipótesis alternativa $H_1 : \mu \neq 45$ y se pide justificar si se puede rechazar la hipótesis nula con un nivel de significancia del 5 %.

Nuevamente se normaliza con Z-Score y se planta que para no poder rechazar la hipótesis nula con el nivel de significancia indicado, se debe cumplir lo siguiente:

$$Z_{\mu_0} = \frac{\sqrt{N}(\hat{\mu} - \mu)}{\sigma} \in (-1{,}96, 1{,}96) \tag{46}$$

Reemplazando los datos del enunciado, se obtiene que $Z_{\mu_0} = 2{,}3717$ que está por fuera del intervalo que delimita la significancia y por lo tanto se rechaza la hipótesis nula.

En el Anexo I se encuentran los detalles de la implementación de la simulación de ambos incisos. En el caso del primer inciso, se generan un cierto número de experimentos y en cada uno de ellos se obtiene un vector de $N = 10$ variables aleatorias con distribución normal utilizando la media muestral y el desvío indicado. Se toma la proporción de muestras de cada experimento que se encuentran dentro del intervalo de confianza teórico y se calcula luego el promedio para todos los experimentos. De este modo, se obtiene el intervalo de confianza simulado. Con 100 experimentos, se obtiene un intervalo de confianza simulado de 0,945

La simulación del segundo inciso es relativamente más compleja. Se ensayan un gran número de experimentos y en cada uno de ellos se genera un vector aleatorio de tamaño $N = 10$ con una distribución normal del desvío indicado y media μ_0 correspondiente a la hipótesis nula. Se almacena la media muestral de cada uno de los experimentos y luego se busca la cantidad de ocurrencias de la media muestral $\hat{X} = 48$ y se pondera por el número de experimentos. Si la proporción es menor al 5 %, se rechaza la hipótesis nula. En el caso de 100.000 experimentos, se obtiene una proporción de $0{,}02104 < 0{,}05$ por lo que se rechaza la hipótesis.

```python
import numpy as np


def experiment():
    # Se generan las muestras con una probabilidad de falla p de 0.01
    n_samples = 10000
    samples = np.zeros((n_samples, 1))
    p = 0.01
    samples[:int(n_samples * p), :] = 1
    permute_ids = np.random.permutation(samples.shape[0])  # Se hace un shuffle de las muestras
    samples = samples[permute_ids]

    # Generamos un vector de N cantidad de fósforos
    n_vector = np.linspace(1, 100, 100)
    EPOCH = 1000  # Cantidad de experimentos
    p_vector = np.zeros_like(n_vector)  # Vector para almacenar probabilidad de obtener uno o más fósforos defectuosos
    fail_cases = np.zeros((EPOCH, 1))  # Vector para almacenar la cantidad de fósforos defectuosos por EPOCH

    for i, n in enumerate(n_vector):
        for j in range(EPOCH):
            idx = np.random.randint(0, n_samples, int(n))  # Obtenemos N índices aleatorios del vector de muestras
            fail_cases[j] = np.sum(samples[idx] == 1, axis=0)  # Cantidad de fósforos defectuosos
        p_vector[i] = np.sum(fail_cases > 0, axis=0) / fail_cases.shape[0] # Proporción de fósforos defectuosos
        if p_vector[i] >= 0.5:
            n_ok = i
            break
    return n_ok


if __name__ == '__main__':
    # Realizamos varios experimentos para reducir el efecto de la aleatoreidad
    TRIALS = 1
```

```
33      n_trials = []
34      for i in range(TRIALS):
35          n_trials.append(experiment())
36      n = np.array(n_trials)
37      n_unique = np.unique(n)
38      n_count = [n_trials.count(x) for x in n_unique]
39      result = np.array((n_unique, n_count))
40      print(result)
41
42      # Generamos N muestras bernoulli con p=0.01
43      EPOCHS = 1000
44      N = 68
45      p = 0.01
46      mean_epoch = []
47      stdv_epoch = []
48      for i in range(EPOCHS):
49          samples = np.random.binomial(N, p, 1000)
50          mean_epoch.append(np.mean(samples, axis=0))
51          stdv_epoch.append(np.std(samples, axis=0))
52      mean_epoch = np.array(mean_epoch)
53      stdv_epoch = np.array(stdv_epoch)
54      print("Media simulada:{media} Desvío simulado: {
    std}".format(media=np.round(np.mean(mean_epoch), 4),
55
                 std=np.round(np.mean(stdv_epoch), 4)))
56      print("Media teórica:{media} Desvío teórico: {std
    }".format(media=N * p, std=np.round((N * p * (1 - p
    )) ** 0.5, 4)))
57
```

```python
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
from scipy.stats import norm
from sklearn.neighbors import KernelDensity


def inverse_sampling(n_samples):
    n_uniform = np.random.uniform(0, 1, n_samples)
    return 3 * np.sqrt(n_uniform)


if __name__ == '__main__':
    sns.set()
    # Generar los datos para nuestra distribución mediante el método de la transformada inversa
    N = 1000
    x = inverse_sampling(N)

    # Estimación de densidad de kernel - Gaussiana Manual
    h = 1  # Coeficiente de dispersión de la gaussiana, se coloca aquí en 1 para facilitar el cálculo

    # Generamos el vector xd para las Gaussianas
    xd = np.linspace(-0.1, 3.1, 1000)
    # Obtenemos la pdf como la suma de los kernels centrados en cada punto y normalizados
    density = sum(norm(xi).pdf(xd) for xi in x) * (1 / x.shape[0])
    # Graphic
    plt.figure(1)
    plt.fill_between(xd, density, alpha=0.5)
    plt.plot(x, np.full_like(x, -0.1), '|k', markeredgewidth=1)
    plt.title("KDE Manual")
    # Cálculo del MSE
    pdf = (2/9)*xd
    mse_manual = np.sum((pdf - density) ** 2, axis=0) * (1 / density.shape[0])
    print("MSE Manual, h=1: {mse}".format(mse=mse_manual))

```

```
36      # Scikit-Learn Kernel Density Estimation
37
38      # Instanciamos el modelo y hacemos el fit
39      kde = KernelDensity(bandwidth=0.201, kernel='
    gaussian')
40      kde.fit(x[:, None])
41
42      # Obtenemos el logarítmo de las probabilidades
    evaluado en xd
43      logprob = kde.score_samples(xd[:, None])
44
45      # Gráfica
46      plt.figure(2)
47      plt.fill_between(xd, np.exp(logprob), alpha=0.5)
48      plt.title("KDE de Scikit-Learn")
49
50      # Cálculo del MSE
51      mse_skl = np.sum((pdf - np.exp(logprob)) ** 2,
    axis=0) * (1 / pdf.shape[0])
52      print("MSE Manual, h=0.2: {mse}".format(mse=
    mse_skl))
53
54      # MSE en función del ancho de banda
55      bandwidths = 10 ** np.linspace(-1, 1, 100)
56      mse_bandwidths = np.zeros(bandwidths.shape)
57      for i, bandwidth in enumerate(bandwidths):
58          kde = KernelDensity(bandwidth=bandwidth,
    kernel='gaussian')
59          kde.fit(x[:, None])
60          logprob = kde.score_samples(xd[:, None])
61          mse_bandwidths[i] = np.sum((pdf - np.exp(
    logprob)) ** 2, axis=0) * (1 / pdf.shape[0])
62
63      plt.figure(3)
64      plt.plot(bandwidths, mse_bandwidths)
65      plt.title("MSE vs. KDE para diferentes anchos de
    banda")
66      plt.xlabel("Ancho de banda")
67      plt.ylabel("MSE")
68
69      # Ancho de banda con menor MSE KDE
70      min_idx = np.argmin(mse_bandwidths)
71      min_mse = np.min(mse_bandwidths)
72      best_n_bin = bandwidths[min_idx]
```

```
    print("Ancho de banda mínimo: ")
    print(best_n_bin)
    print("MSE mínimo")
    print(min_mse)

    plt.show()

```

```python
import numpy as np

if __name__ == '__main__':
    # Datos de entrada
    n = 100  # Cantidad de simulaciones
    N = 10  # Cantidad de muestras tomadas
    mu = 48  # Media muestral
    sigma = 4  # Desvío

    # Intervalo de confianza teórico
    mu_min = mu - 1.96 * sigma / (N ** 0.5)
    mu_max = mu + 1.96 * sigma / (N ** 0.5)

    trust = 0
    for i in range(n):
        # Generamos valores aleatorios de la distribución
        x = (sigma / (N ** 0.5)) * np.random.randn(N) + mu
        trust = trust + (1 / (n * N)) * np.sum(np.logical_and(x >= mu_min, x <= mu_max), axis=0)

    real_trust = 0.95
    print("Confianza teorica: {t}".format(t=real_trust))
    print("Confianza simulada: {t}".format(t=np.round(trust, 4)))

    n_exp = 100000  # Cantidad de experimentos para el test de hipótesis
    mu_o = 45
    alpha = 0.05
    medias_muestrales = []
    for i in range(n_exp):
        x = np.random.normal(mu_o, sigma, N)
        medias_muestrales.append(np.mean(x, axis=0))
    medias_muestrales = np.array(medias_muestrales)
    prob_media = np.sum(np.round(medias_muestrales) == mu) / n_exp
    print("Probabilidad simulada de obtener una media muestral X=48: {}".format(prob_media))
    if prob_media < alpha:
        print("Se rechaza la hipótesis nula")
    else:
```

```
37          print("No se puede rechazar la hipótesis nula
    ")
38
```