Capítulo 4

Modelo normal

Comenzaremos con un modelo en el que supondremos que cada variable es independiente de las demás. Este modelo es el más sencillo, ya que con esta suposición se simplifican los cálculos. Sin embargo, esta suposición es incorrecta, dado que la existencia de la fuente puntual hace que las variables cercanas a ella estén correlacionadas. En este trabajo exploraremos las diferencias entre suponer independencia y correlación, y cómo afectan a los resultados obtenidos.

Supongamos que tenemos un conjunto de N observaciones x_i continuas, con $i = 1, \ldots, N$. Cada x_i pertenece a una de las s zonas S_j, identificadas por sus coordenadas $lat(S_j)$ y $long(S_j)$. Es importante destacar que $s \leq N$, ya que cada zona contiene más de una observación. Para cada zona S_j, definimos $x_{s_j} = \sum_{x_i \in S_j} x_i$ como la suma de los valores observados en S_j, y n_{s_j} como el número de observaciones en S_j. Además, definimos la suma total $X = \sum x_i$ como la suma de todos los valores observados en el espacio.

En este enfoque, utilizaremos ventanas que consisten en círculos centrados en cada observación, con un radio que varía continuamente desde un límite inferior hasta un límite superior. El límite superior será el círculo que incluya como máximo el 50 % del total de observaciones. Se ignorarán los círculos que contengan una sola observación.

Para un círculo Z dado, definimos n_Z y $x_Z = \sum_{x_i \in Z} x_i$. Por las definiciones anteriores, N_Z representa el número de observaciones dentro del círculo y x_Z es la suma de los valores dentro del mismo.

4.1. Prueba de hipótesis

La hipótesis nula postula que tanto dentro como fuera de la zona, la media y la varianza son iguales, ya que no hay una fuente puntual presente. La hipótesis alternativa, por otro lado, sugiere la existencia de una fuente puntual, lo que implica que hay al menos una zona donde tanto la media como la varianza son diferentes tanto dentro como fuera de ella. En otras palabras:

$$H_0 : \mu_z = \mu_{z'} \quad \text{vs} \quad H_1 : \mu_z \neq \mu_{z'}, \quad \text{para } z \in Z$$

La función de verosimilitud para esta prueba es

$$L(\mu_z, \mu_{z'}, \sigma, Z|x_1, ..., x_n) = \prod_{i=1}^{N} f(x_i) = \prod_{x_i \in Z} f(x_i) \prod_{x_i \in Z'} f(x_i)$$

Donde Z' es el complemento del círculo. Al desarrollar los productos, obtenemos

$$\begin{aligned}\prod_{x_i \in Z} f(x_i) &= \prod_{x_i \in Z} \frac{1}{\sigma\sqrt{2\pi}} \exp\left\{-\frac{1}{2}\left(\frac{x_i - \mu_Z}{\sigma}\right)^2\right\} \\ &= \frac{1}{(\sigma\sqrt{2\pi})^{n_Z}} \exp\left\{\sum_{x_i \in Z} -\frac{1}{2}\left(\frac{x_i - \mu_z}{\sigma}\right)^2\right\}\end{aligned}$$

$$\begin{aligned}\prod_{x_i \in Z'} f(x_i) &= \prod_{x_i \in Z'} \frac{1}{\sigma\sqrt{2\pi}} \exp\left\{-\frac{1}{2}\left(\frac{x_i - \mu_{z'}}{\sigma}\right)^2\right\} \\ &= \frac{1}{(\sigma\sqrt{2\pi})^{n_{Z'}}} \exp\left\{\sum_{x_i \in Z'} -\frac{1}{2}\left(\frac{x_i - \mu_{z'}}{\sigma}\right)^2\right\}\end{aligned}$$

Dado que $n_Z + n_{Z'} = N$, entonces la función de verosimilitud es

$$L(\mu_Z, \mu_{Z'}, \sigma, Z) = \frac{1}{(\sigma\sqrt{2\pi})^N} \exp\left\{\sum_{x_i \in Z} -\frac{1}{2}\left(\frac{x_i - \mu_z}{\sigma}\right)^2 + \sum_{x_i \in Z'} -\frac{1}{2}\left(\frac{x_i - \mu_{z'}}{\sigma}\right)^2\right\}$$

Por lo tanto, la función de log verosimilitud es

$$l(\mu_Z, \mu_{Z'}, \sigma, Z) = -N\log(\sqrt{2\pi}) - N\log(\sigma) - \sum_{x_i \in Z} \frac{1}{2}\left(\frac{x_i - \mu_Z}{\sigma}\right)^2 - \sum_{x_i \in Z'} \frac{1}{2}\left(\frac{x_i - \mu_{Z'}}{\sigma}\right)^2$$

Una vez obtenida la función de log verosimilitud, el siguiente paso es encontrar los estimadores de máxima verosimilitud bajo H_0 y bajo H_1.

Comenzaremos primero con el caso más sencillo, que sería bajo H_0. Como $\mu_Z = \mu_{Z'}$, entonces la función de log verosimilitud es

$$l(\mu, \sigma) = -N\log(\sqrt{2\pi}) - N\log(\sigma) - \sum_{i=1}^{N} \frac{1}{2}\left(\frac{x_i - \mu}{\sigma}\right)^2$$

ahora el siguiente paso seria maximizar la función, por lo que haremos las derivadas parciales con respecto a μ y σ^2 e igualaremos a cero y despejaremos.

Para μ

$$\frac{\partial l(\mu, \sigma)}{\partial \mu} = -\frac{1}{2\sigma}\sum_{i=1}^{N}[-2x_i + 2\hat{\mu}] = 0$$

$$\hat{\mu} = \frac{X}{N}$$

Para σ^2

$$\frac{\partial l(\mu, \sigma)}{\partial \sigma^2} = -\frac{N}{2\hat{\sigma}^2} + \frac{1}{2\hat{\sigma}^4}\sum_{i=1}^{N}(x_i - \hat{\mu})^2 = 0$$

$$\hat{\sigma} = \frac{\sum(x_i - \hat{\mu})^2}{N}$$

Los estimadores son $\hat{\mu} = \frac{X}{N}$ y $\hat{\sigma} = \frac{\sum(x_i - \hat{\mu})^2}{N}$.

Ahora hagamos el caso mas tedioso que seria Bajo H_1, nuevamente maximizaremos pero ahora restringidos bajo H_1, donde sacaremos las derivadas parciales de μ_Z,μ'_Z y la varianza común σ_Z^2 y despejaremos nuevamente. y la varianza común
$\hat{\sigma}_z = \frac{1}{N}\left[\sum_{x_i \in Z} x_i^2 - 2x_i\mu_z + n_z\mu_z^2 + \sum_{x_i \notin Z} x_i^2 - 2(X - x_z)\mu_{z'} + (N - n_z)\mu_{z'}^2\right]$
Para μ_Z

$$\frac{\partial l(\mu_Z, \mu_{Z'}, \sigma, Z)}{\partial \mu_Z} = -\frac{1}{2\sigma}\sum_{x_i \in Z}[-2x_i + 2\hat{\mu}_Z] = 0$$

para μ'_Z

$$\frac{\partial l(\mu_Z, \mu_{Z'}, \sigma, Z)}{\partial \mu_{Z^c}} = -\frac{1}{2\sigma}\sum_{x_i \in Z'}[-2x_i + 2\hat{\mu}_{Z'}] = 0$$

los estimadores son

$$\hat{\mu}_Z = \frac{x_Z}{n_Z} \text{ y } \hat{\mu}_{Z'} = \frac{x_{Z'}}{n_{Z'}}$$

Para σ_Z^2

$$\frac{\partial l(\mu_Z, \mu_{Z'}, \sigma, Z)}{\partial \sigma_Z^2} = -\frac{N}{2\hat{\sigma}^2} + \frac{1}{2\hat{\sigma}^4}\left[\sum_{x_i \in Z}(x_i - \hat{\mu}_Z)^2 + \sum_{x_i \in Z'}(x_i - \hat{\mu}_{Z'})^2\right] = 0$$

EL estimador es

$$\hat{\sigma}_z = \frac{1}{N}\left[\sum_{x_i \in Z} x_i^2 - 2x_i\mu_z + n_z\mu_z^2 + \sum_{x_i \notin Z} x_i^2 - 2(X - x_z)\mu_{z'} + (N - n_z)\mu_{z'}^2\right]$$

Por lo que solo nos faltaría un estimador para Z, por lo que tendríamos una función que depende de Z

$$\log L(Z) = -N\log(\sqrt{2\pi}) - N\log(\sqrt{\sigma_z}) - \frac{N}{2}$$

Tenemos que

$$\max_z \frac{L_Z}{L_0}$$

es equivalentemente a

$$\max_z \log\left(\frac{L_Z}{L_0}\right) = \max_z \left[\log(L_Z) - \log(L_0)\right]$$

Código Python en LaTeX

4.1.1. importación

El primer paso es importar las librerias necesarias

```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
from tabulate import tabulate
import math
from scipy.optimize import minimize
import seaborn as sns
import time
import random
```

El segundo paso es importar los datos

```
df = pd.read_excel("C:/Users/HP/Desktop/Desk/jupyter/database/
   pole.xls", header=None)
valor1 = df[2].tolist()
valor2 = df[3].tolist()
x = df[0].tolist()
y = df[1].tolist()
```

4.1.2. Funciones

Cálculo de la distancia euclidiana entre dos puntos

La distancia euclidiana entre dos puntos (x_1, y_1) y (x_2, y_2) se puede calcular utilizando la siguiente fórmula:

$$\text{distancia} = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

En Python, esta fórmula se puede implementar con la siguiente función:

```
def distancia(x1, y1, x2, y2):
    return math.sqrt((x1 - x2)**2 + (y1 - y2)**2)
```

Ejemplo

Supongamos que queremos calcular la distancia entre los puntos $(1, 2)$ y $(4, 6)$. Los valores son:

$$(x_1, y_1) = (1, 2)$$

$$(x_2, y_2) = (4, 6)$$

Sustituyendo estos valores en la fórmula obtenemos:

$$\text{distancia} = \sqrt{(1-4)^2 + (2-6)^2} = 5$$

$$distancia(1, 2, 4, 6) \longrightarrow 5$$

Por lo tanto, la distancia entre los puntos $(1, 2)$ y $(4, 6)$ es 5.

Selección de elementos en una lista con índices específicos

La función `list_selec` toma dos argumentos: `valor` y `puntos`. El argumento `valor` es una lista de la cual queremos seleccionar ciertos elementos, y el argumento `puntos` es una lista de índices que indica qué elementos de `valor` debemos seleccionar. La función crea una nueva lista `X` que contiene los elementos de `valor` en las posiciones especificadas por `puntos`.

```
def list_selec(valor, puntos):
    X = []

    for i in puntos:
        X.append(valor[i])

    return X
```

Ejemplo

Supongamos que tenemos la lista `valor = [10, 20, 30, 40, 50]` y queremos seleccionar los elementos en las posiciones `1`, `3` y `4`. Los índices que queremos usar están en la lista `puntos = [1, 3, 4]`.

$$list\_select(valor, puntos) \longrightarrow [20, 40, 50]$$

Por lo tanto, con `valor = [10, 20, 30, 40, 50]` y `puntos = [1, 3, 4]`, obtenemos la lista `[20, 40, 50]`.

Cálculo de distancias desde un punto específico a todos los demás puntos

La función `distancias_al_punto` toma tres argumentos: `x` y `y`, que son listas de coordenadas x e y de los puntos, y `j`, que es el índice del punto específico del cual queremos calcular las distancias a los demás puntos.

```
def distancias_al_punto(x, y, j):
    d = []
    N = len(x)

    for i in range(0, N):
        d.append([distancia(x[j], y[j], x[i], y[i]), i])

    return d
```

Ejemplo

Supongamos que tenemos las siguientes listas de coordenadas:

- `x = [1, 4, 5]`
- `y = [2, 6, 3]`

Queremos calcular las distancias desde el punto en el índice `1` (punto (4, 6)) a todos los demás puntos.

A continuación se muestra cómo se realizan los cálculos a mano:

- Distancia del punto (4, 6) al punto (1, 2):

$$\text{distancia} = \sqrt{(4-1)^2 + (6-2)^2} = \sqrt{3^2 + 4^2} = \sqrt{9+16} = 5{,}0$$

- Distancia del punto (4, 6) al punto (4, 6) (sí mismo):

$$\text{distancia} = \sqrt{(4-4)^2 + (6-6)^2} = \sqrt{0+0} = 0{,}0$$

- Distancia del punto (4, 6) al punto (5, 3):

$$\text{distancia} = \sqrt{(4-5)^2 + (6-3)^2} = \sqrt{(-1)^2 + 3^2} = \sqrt{1+9} = 3{,}162$$

Utilizando la función obtendríamos

$$\textit{distancias\_al\_punto}(x, y, 1) \longrightarrow \texttt{[[5.0, 0], [0.0, 1], [3.162, 2]]}$$

Graficar una Circunferencia

La función `graficar_circunferencia` se encarga de graficar una circunferencia en un plano cartesiano. La función también permite graficar un conjunto de puntos específicos en el mismo gráfico.

```
def graficar_circunferencia(x1, y1, h, k, radio):
    # Crear un conjunto de  ngulos  (en radianes) desde 0 hasta 2
       *pi
    theta = np.linspace(0, 2*np.pi, 100)

    # Calcular las coordenadas (x, y) de la circunferencia
    x = radio * np.cos(theta) + h
    y = radio * np.sin(theta) + k

    # Crear el gr fico de la circunferencia
    plt.figure(figsize=(9, 9))
    plt.xlim(min(x), max(x))
    plt.ylim(min(y), max(y))
    plt.plot(x1, y1, "o", color="red")
    plt.plot(x, y, label='Circunferencia')
    # Especificar los l mites en los ejes x e y
    plt.axis('scaled')

    # Mostrar el gr fico
    plt.show()
```

Determinar puntos dentro o fuera de un circulo

La función `dentro_fuera` devuelve dos listas: `dentro` con los índices de los puntos que están dentro del radio y `fuera` con los índices de los puntos que están fuera del radio.

```
def dentro_fuera(punto, r, X, Y):
    dentro = []
    fuera = []
    N = len(X)
    Distancia = distancias_al_punto(X, Y, punto)

    for i in range(0, N):
        if Distancia[i][0] <= r:
            dentro.append(Distancia[i][1])
        else:
            fuera.append(Distancia[i][1])

    return [dentro, fuera]
```

Ejemplo

Supongamos que tenemos un punto de referencia en el índice `0`, un radio de `1`, y coordenadas de puntos como sigue:

- `X = [0, 1, 2]`
- `Y = [0, 0, 0]`

Queremos determinar qué puntos están dentro o fuera de un circulo con centro en el punto (0, 0) y radio 1.

$$dentro\_fuera(0, 1, X, Y) \longrightarrow \texttt{[[0,1],[2]]}$$

Obtener Círculos con Número de Puntos Dentro en un Rango

La función `Círculos` encuentra todos los círculos que tienen un número de puntos dentro del circulo que está entre un límite inferior y un límite superior.

```
def Circulos(limite_inf, limite_sup, x, y):
    x_sin_repetidos = []
    circulos = []
    n = len(x)

    for i in range(0, n):
        d = distancias_al_punto(x, y, i)
        d = sorted(d)
        for j in range(0, n):
            if round(d[j][0]) > 0:
                q = len(dentro_fuera(i, d[j][0], x, y)[0]) / n
                if q <= limite_sup and limite_inf <= q:
                    circulos.append([i, round(d[j][0], 3)])

    for elemento in circulos:
        if elemento not in x_sin_repetidos:
            x_sin_repetidos.append(elemento)

    return x_sin_repetidos
```

La función devuelve una lista de círculos con el centro en el índice del punto y el radio del círculo, que cumplen con el número de puntos dentro en el rango especificado.

Cálculo de la Log-Verosimilitud

La función `L_Z` calcula la log-verosimilitud para una zona dada

```python
def L_Z(punto, r, x, y, valor):
    dentroFuera = dentro_fuera(punto, r, x, y)
    fuera = dentroFuera[1]
    dentro = dentroFuera[0]
    N = len(valor)
    v_fuera = len(list_selec(valor, fuera)) * np.var(list_selec(
       valor, fuera))
    v_dentro = len(list_selec(valor, dentro)) * np.var(list_selec
       (valor, dentro))
    sigma_z = (v_fuera + v_dentro) / N

    return -N/2 * math.log(2*math.pi) - N/2 * math.log(sigma_z) -
        N/2
```

Obtención del Círculo con Máxima Log-Verosimilitud

La función `max_L_Z` encuentra el círculo que maximiza la log-verosimilitud entre un conjunto dado de círculos.

implementación

```python
def max_L_Z(x, y, valor, circulos):
    N = len(valor)
    L = []

    L0 = -N * math.log(math.sqrt(2 * math.pi)) - N * math.log(np.
       std(valor)) - N / 2

    for i in range(0, len(circulos)):
        Lamda = L_Z(circulos[i][0], circulos[i][1], x, y, valor)
           - L0
        L.append([Lamda, circulos[i][0], circulos[i][1]])

    m = sorted(L)
    return m[len(m) - 1]
```

Funcionamiento

El proceso para encontrar el círculo con máxima log-verosimilitud se realiza en los siguientes pasos:

1. **Inicialización**:
    - Se calcula L_0, que representa una log-verosimilitud base. Esta se utiliza como punto de referencia para comparar los círculos.

2. **Iteración sobre los círculos**:
    - Para cada círculo en el conjunto dado:
        a) Se calcula su log-verosimilitud utilizando la función `L_Z`.
        b) Se resta L_0 para obtener una medida relativa de ajuste.
        c) Se almacena esta información junto con las coordenadas del círculo.
3. **Ordenamiento**:
    - Se ordena la lista de círculos basándose en sus valores de log-verosimilitud.
4. **Selección**:
    - Se selecciona el último elemento de la lista ordenada, que corresponde al círculo con la máxima log-verosimilitud.

La función devuelve el círculo con el valor máximo de log-verosimilitud basado en los datos proporcionados.

Simulación Monte-Carlo

```
import numpy as np
import random

estadistico = []
numero_de_simulaciones = 999

for i in range(0, numero_de_simulaciones):
    conjunto_de_datos = np.random.normal(loc=np.mean(valor2),
       scale=np.std(valor2), size=len(valor2))
    random.shuffle(conjunto_de_datos)
    Estimacion_simulacion = max_L_Z(x, y, conjunto_de_datos,
       circulos)
    estadistico.append(Estimacion_simulacion[0])

estadistico = sorted(estadistico)
```

Significancia

```
estadistico.append(Estimaciones[0])
\# Ordenar estad sticos de mayor a menor
estadistico = sorted(estadistico, reverse=True)

\# Calcular la significancia
for i in range(0, len(estadistico)):
```

```
    if estadistico[i] == Estimaciones[0]:
        c = i

significancia = (c + 1) / (numero_de_simulaciones + 1)

print("La significancia es", significancia)
```