Estadística No Paramétrica y Pruebas Robustas - Día 5

Change Status

pending 40 min

Learning Objectives

1. Aplicar pruebas no paramétricas cuando fallan los supuestos tradicionales
2. Usar técnicas de remuestreo (bootstrap) para estimación confiable
3. Evaluar robustez de conclusiones estadísticas con métodos alternativos

Activities and Learning

Task 1: Pruebas No Paramétricas para Comparación de Grupos (10 minutos)

Las pruebas no paramétricas no requieren supuestos distribucionales estrictos, siendo útiles cuando los datos no son normales o tienen outliers.

Prueba U de Mann-Whitney: Comparación de Dos Grupos Independientes

Alternativa no paramétrica a la prueba t: Compara medianas en lugar de medias.

Hipótesis:

H_0: Las distribuciones de ambos grupos son idénticas
H_1: La distribución de un grupo está desplazada respecto al otro

Ventajas:

Robusta a outliers: Usa rangos en lugar de valores absolutos
No requiere normalidad: Funciona con cualquier distribución continua
Eficiente: Similar poder estadístico que t-test cuando se cumplen supuestos

```
from scipy import stats

# Datos de dos grupos
grupo_a = np.random.normal(100, 15, 50)
grupo_b = np.random.normal(110, 20, 45)  # Media diferente, varianza diferente

# Prueba Mann-Whitney
u_stat, p_value = stats.mannwhitneyu(grupo_a, grupo_b, alternative='two-sided')

print(f"Estadístico U: {u_stat:.1f}")
print(f"Valor p: {p_value:.4f}")
print(f"Mediana grupo A: {np.median(grupo_a):.2f}")
print(f"Mediana grupo B: {np.median(grupo_b):.2f}")
```

Prueba de Kruskal-Wallis: Comparación de Múltiples Grupos

Alternativa no paramétrica al ANOVA: Extensión de Mann-Whitney para k grupos.

Lógica: Combina todos los datos, los ordena, y compara cómo se distribuyen los rangos entre grupos.

```
# Tres grupos con distribuciones diferentes
grupo_1 = np.random.normal(100, 15, 40)
grupo_2 = np.random.exponential(20, 35) + 80  # Distribución diferente
grupo_3 = np.random.normal(105, 25, 38)

# Prueba Kruskal-Wallis
h_stat, p_value = stats.kruskal(grupo_1, grupo_2, grupo_3)

print(f"Estadístico H: {h_stat:.3f}")
print(f"Valor p: {p_value:.4f}")
print(f"Medianas: G1={np.median(grupo_1):.2f}, G2={np.median(grupo_2):.2f}, G3={np.median(grupo_3):.2f}")
```

Task 2: Correlaciones No Paramétricas (8 minutos)

Cuando las relaciones no son lineales o los datos no son normales, las correlaciones tradicionales fallan.

Correlación de Spearman: Relaciones Monótonas

Coeficiente rho de Spearman: Mide relaciones monótonas (crecientes o decrecientes) usando rangos.

Fórmula: $\rho = 1 - (6 \times \Sigma(d_i^2)) / (n(n^2 - 1))$ Donde d_i es la diferencia de rangos para cada observación.

Ventajas:

Captura relaciones no lineales: Mientras sean monótonas
Robusta a outliers: Usa rangos, no valores absolutos
Funciona con datos ordinales: Ideal para encuestas Likert

```
# Datos con relación no lineal pero monótona
x = np.random.uniform(0, 10, 100)
y = np.log(x + 1) + np.random.normal(0, 0.2, 100)  # Relación logarítmica

# Correlación de Pearson (pobre para relación no lineal)
pearson_r, _ = stats.pearsonr(x, y)

# Correlación de Spearman (mejor para relaciones monótonas)
spearman_rho, _ = stats.spearmanr(x, y)

print(f"Correlación Pearson: {pearson_r:.3f}")
print(f"Correlación Spearman: {spearman_rho:.3f}")
```

Tau de Kendall: Correlación por Concordancia

Más conservadora que Spearman: Cuenta pares concordantes/discordantes.

Interpretación: Proporción de pares que siguen la misma dirección menos pares que van en direcciones opuestas.

Uso: Datos con muchos empates o cuando se quiere una medida más robusta.

Task 3: Bootstrap para Estimación Robusta (12 minutos)

Bootstrap estima la variabilidad de estadísticos remuestreando los datos observados muchas veces.

Principio del Bootstrap

Idea fundamental: Los datos observados representan la población. Al remuestrear con reemplazo, podemos estimar la distribución muestral del estadístico.

Pasos:

Tomar muestra bootstrap: n observaciones con reemplazo de los datos originales
Calcular estadístico: Media, mediana, correlación, etc.
Repetir B veces: Típicamente B = 1000-10000
Analizar distribución: Media, desviación estándar, intervalos de confianza
Intervalos de Confianza Bootstrap

Percentil bootstrap: Usa percentiles de la distribución bootstrap.

```
def bootstrap_confidence_interval(data, statistic_func, n_bootstrap=1000, confidence=0.95):
    """
    Calcula intervalo de confianza bootstrap
    """
    bootstrap_stats = []

    for _ in range(n_bootstrap):
        # Muestra bootstrap con reemplazo
        sample = np.random.choice(data, size=len(data), replace=True)
        # Calcula estadístico
        stat = statistic_func(sample)
        bootstrap_stats.append(stat)

    # Intervalo de confianza
    alpha = 1 - confidence
    lower = np.percentile(bootstrap_stats, alpha/2 * 100)
    upper = np.percentile(bootstrap_stats, (1 - alpha/2) * 100)

    return np.mean(bootstrap_stats), lower, upper

# Ejemplo: IC bootstrap para la mediana
data = np.random.exponential(2, 100)  # Distribución no normal

media_bootstrap, ci_lower, ci_upper = bootstrap_confidence_interval(
    data, np.median, n_bootstrap=1000, confidence=0.95
)

print(f"Mediana observada: {np.median(data):.3f}")
print(f"IC Bootstrap 95%: ({ci_lower:.3f}, {ci_upper:.3f})")
```

Bootstrap para Pruebas de Hipótesis

Prueba de permutación: Método exacto que no requiere supuestos distribucionales.

```
def permutation_test(group1, group2, n_permutations=1000):
    """
    Prueba de permutación para diferencia de medias
    """
    # Estadístico observado
    observed_diff = np.mean(group1) - np.mean(group2)

    # Combinar datos
    combined = np.concatenate([group1, group2])
    n1, n2 = len(group1), len(group2)
```

```
    # Generar distribución nula
    null_diffs = []
    for _ in range(n_permutations):
        # Permutación aleatoria
        np.random.shuffle(combined)
        perm_group1 = combined[:n1]
        perm_group2 = combined[n1:]
        null_diffs.append(np.mean(perm_group1) - np.mean(perm_group2))

    # Valor p
    p_value = np.mean(np.abs(null_diffs) >= np.abs(observed_diff))

    return observed_diff, p_value

# Ejemplo con datos no normales
group_a = np.random.exponential(1, 50)
group_b = np.random.exponential(1.3, 45)

diff_observed, p_perm = permutation_test(group_a, group_b, n_permutations=1000)

print(f"Diferencia observada: {diff_observed:.3f}")
print(f"Valor p (permutación): {p_perm:.4f}")
```