index

January 18, 2022

1 Welch's T-test - Lab

1.1 Introduction

Now that you've gotten a brief introduction to Welch's t-test, it's time to practice your NumPy and math programming skills to write your own Welch's T-test function!

1.2 Objectives

In this lab you will:

- Write a function to calculate Welch's t-score
- Calculate the degrees of freedom for a Welch's t-test
- Calculate p-values using Welch's t-test

1.2.1 Welch's t-test

Recall that Welch's t-Test is given by

$$t = \frac{\bar{X}_1 - \bar{X}_2}{\sqrt{\frac{s_1^2}{N_1} + \frac{s_2^2}{N_2}}} = \frac{\bar{X}_1 - \bar{X}_2}{\sqrt{se_1^2 + se_2^2}} \tag{1}$$

where $\bar{X}_i$, ${s_i}^2$, and N_i are the sample mean, sample variance, and sample size, respectively, for sample i.

Write a function for calculating Welch's t-statistic using two samples a and b. To help, 2 potential samples are defined below.

> **Important Note**: While the formula does not indicate it, it is appropriate to take the absolute value of the t-value.

[2]:
```
import numpy as np

np.random.seed(82)
control = np.random.normal(loc=10, scale=1, size=8)
treatment = np.random.normal(loc=10.5, scale=1.2, size=12)
```

[3]:
```
control
```

[3]:
```
array([10.8406504 ,  8.64285284, 11.28693651, 10.57347539, 10.57945015,
        9.97237817,  9.61844717,  9.69121804])
```

[4]:
```
treatment
```

[4]:
```
array([12.16530726, 12.5597993 , 11.76525366,  9.82399228, 11.03539891,
       12.8992533 , 10.78680718, 11.71126641, 10.2343344 ,  9.77839837,
        9.72938618, 10.39959928])
```

[5]:
```
def welch_t(a, b):

    """ Calculate Welch's t-statistic for two samples. """
    n_a = len(a)
    n_b = len(b)
    a_mean = a.mean()
    b_mean = b.mean()
    a_std = a.std(ddof = 1)
    b_std = b.std(ddof = 1)

    se = np.sqrt( a_std**2 / n_a + b_std**2 / n_b)
    t = np.abs(a_mean - b_mean) / se

    return t

welch_t(control, treatment)
# 2.0997990691576858
```

[5]:
```
2.0997990691576858
```

1.3 Degrees of freedom

Once you have the t-score, you also need to calculate the degrees of freedom to determine the appropriate t-distribution and convert this score into a p-value. The effective degrees of freedom can be calculated using the formula:

$$v \approx \frac{(\frac{s_1^2}{N_1} + \frac{s_2^2}{N_2})}{\frac{s_1^4}{N_1^2 v_1} + \frac{s_2^4}{N_2^2 v_2}} \tag{2}$$

N_i - sample size of sample i
$s_i{}^2$ - variance of sample i
v_i - degrees of freedom for sample i (equivalent to $N_i - 1$)

Write a second function to calculate degree of freedom for above samples:

[6]:
```
def welch_df(a, b):
    n_a = len(a)
    n_b = len(b)
```

```
    a_dof = n_a - 1
    b_dof = n_b - 1

    a_std = a.std(ddof = 1)
    b_std = b.std(ddof = 1)

    num = ( a_std**2 / n_a + b_std**2 / n_b)**2
    denum = a_std**4 / (a_dof * n_a**2) + b_std**4 / (b_dof * n_b**2)

    nu = num / denum


    """ Calculate the effective degrees of freedom for two samples. """
    return nu

welch_df(control, treatment)
# 17.673079085111
```

[6]: 17.673079085110995

Now calculate the welch t-score and degrees of freedom from the samples, a and b, using your functions.

[7]:
```
# Your code here; calculate t-score and the degrees of freedom for the two␣
 ↪samples, a and b
a = control
b = treatment

t = welch_t(a, b)
df = welch_df(a, b)
print(t, df)
# 2.0997990691576858 17.673079085111
```

```
2.0997990691576858 17.673079085110995
```

1.4 Convert to a p-value

Great! Now that you have the t-score and the degrees of freedom, it's time to convert those values into a p-value (for a one-sided t-test). Remember that the easiest way to do this is to use the cumulative distribution function associated with your particular t-distribution.

Calculate the p-value associated with this experiment.

[8]:
```
# Your code here; calculate the p-value for the two samples defined above
import scipy.stats as stats
p = 1 - stats.t.cdf(t, df)
print(p)
# 0.02519166225846454
```

```
0.025191666225846454
```

In this case, there is a 2.5% probability you would see a t-score equal to or greater than what you saw from the data. Given that alpha was set at 0.05, this would constitute sufficient evidence to reject the null hypothesis.

Building on this, you can also write a function that calculates the p-value for given samples with a two-sided test by taking advantage of the symmetry of the t-distribution to calculate only one side. The two-tailed p-value is simply twice the one-tailed value because you want the probability: $> t < -|t|$ and $t > |t|$, where t̂ is the t-statistic calculated from our data

With that, define a summative function `p_val_welch(a, b, two_sided=False)` which takes in two samples a and b, as well as an optional binary variable to allow you to toggle between a one and two-sided Welch's t-test.

> The default behavior should be set to one-sided as indicated above. If the parameter two_sided is set to True, the function should return the p-value for a two-sided t-test, as opposed to a one-sided t-test.

[9]:
```
def p_value(a, b, two_sided=False):
    t = welch_t(a, b)
    df = welch_df(a, b)
    if two_sided == False:
         p = 1 - stats.t.cdf(t, df)
    elif two_sided == True:
         p = 2*(1 - stats.t.cdf(t, df))

    return p
```

Now briefly test your function; no need to write any code just run the cells below to ensure your function is operating properly. The output should match the commented values.

[10]:
```
p_value(treatment, control)
# 0.025191666225846454
```

[10]:
```
0.025191666225846454
```

[11]:
```
p_value(treatment, control, two_sided=True)
# 0.05038333245169291
```

[11]:
```
0.05038333245169291
```

1.5 Summary

Nice work! In this lab, you practiced implementing functions for Welch's t-test when sample variances or sample sizes differ in your experimental groups. You also got to review converting t-scores into p-values. All of this should continue to build on your abilities to effectively design and carry out statistically valid hypothesis tests.

[ ]:
```

```