Hypothesis testing for leakage assessment in side-channel analysis

Tutorial workbook

June 15, 2023

Goals: This tutorial aims to give participants a solid technical understanding of hypothesis testing and how it is used for leakage assessment. Participants will perform hands-on leakage detection tests and draw appropriate conclusions from the data. We will discuss common pitfalls to improve our results' accuracy, unravel the mystery of the magic number 4.5, and learn how to interpret p-values. The tutorial is rich in hands-on exercises via interactive Jupyter notebooks and data sets. We start with simple examples using simulated data such that participants grasp the mechanics of hypothesis testing, 1 and move from there to real power traces, 2.

Before you start: The steps required for solving the assignment are the following:

1. Install the Python environment. The provided code snippets are written in a `jupyter notebook`. If you are unfamiliar with this concept, you can find out more here: https://realpython.com/jupyter-notebook-introduction/.We have tested the code on `python 3.` (you can select this with https://docs.anaconda.com/anaconda/install), `Jupyter Lab 2.2.6`. To run the notebooks, the following packages should be installed: `numpy`, `scipy`, `matplotlib`, and `seaborn`.

2. Download the provided files (data and code) as described in Table 1. Datasets are courtesy of CESCA lab.

1 A primer on Null-Hypothesis Significance Tests

Exercise 1 In this exercise, we will visualize the distribution of the **mean** test statistic. Open the notebook `Lake_water.ipynb`. Please consider the following questions:

1. **The natural variance of a test statistic**. Since we do not have the necessary skills or equipment to measure the salt concentration in water, we will simulate this activity. Imagine taking samples from the lake water and measuring the average salinity. The function `collect_glass_lake_water` simulates filling a glass of water. The function generates a fixed number of random values from a normal distribution. The function returns the salt concentration of the individual water

Table 1: Files to download for solving the first part of the tutorial

File name	Description
`Lake_water.ipynb`	[CODE] `jupyter`. Download here. E1(Q1,Q2,Q3), E2(Q1,Q2), E3(Q2)
`Magic_coins.ipynb`	[CODE] `jupyter`. Download here. E2(Q4) E3(Q1)
`PhD_vs_Faculty.ipynb`	[CODE] `jupyter`. Download here. E2(Q2)E3(Q3)

drops. Compute the sample mean three times. What do you notice?

2. **The number of samples and the test statistic**. Let us compare how the average value of different sample numbers affects the test statistic's variance. The function `sample_means_water` automates the computation of the mean value for many experiments. If the population mean is $\mu = 0.3$, is obtaining a value of 0.2 reasonable? Is there a difference between the values when we use five water drops in a glass vs. twenty water drops in a glass?

3. **Population distribution vs. test statistic distribution.** Let us compare the population distribution with the test statistic distribution. Change the value of variable `number_of_drops` successively to 7, 10, 20, and 30 and observe its effect on the estimate of the sample mean distribution. What do you notice?

Exercise 2. Understanding the null distribution. The textbook choice for the null-hypothesis distribution when testing sample means is the t distribution. But why is this the case? The goal of this exercise is to answer this question. We will estimate the null distribution using simulated data, which we then compare with the textbook t-distribution.

1. **Null-distribution for the one-sample test statistic.**

   Open the notebook `Lake_water.ipynb`. The code for this exercise is labeled Exercise 2, Question 1. Recall that the null distribution for our case is $H_0 : \mu = 0.3$. We will proceed as follows:

   a Generate several *mean* values ($\overline{x}$). The number of observed mean statistic values is controlled by the variable `number_of_experiments`. Change the null-distribution $H_0 : \mu = 0.2$ and $H_0 : \mu = 0.5$ by changing the value of the variable `null_hypothesis` and plot this relative to the observed sample mean stored in variable `observed_sample_mean`. What do you notice?

b A *t-score* roughly measures how far away the observed $\overline{x}$ is compared to the population mean μ, normalized by the standard deviation in our sample (s). To compute the t-score, we use the formula:

$$t = \frac{\overline{x} - \mu}{\frac{s}{\sqrt{n-1}}}$$

Experiment with the code, generate sample data for different populations, and observe the t-score values. For which cases do you notice a small t-score, and for which cases do you notice a large t-score?

c The final step. Let us normalize the distribution we obtained in the previous step and compare it with a t distribution. What do you notice?

2. **(Optional) Normal vs. t-distribution.** In this exercise, we compare the t-distribution to the normal distribution. Open the notebook `Lake_water.ipynb`, The cell labeled Exercise 2, Question 2, provides the code to plot the t-test and normal distributions. Choose different degrees of freedom and compare the plot of the t-distribution with the normal distribution. What do you notice?

3. **Null-distribution for the two-sample test statistic.** Open the notebook `PhD_vs_Faculty.ipynb`. The code in the cell is labeled Exercise 2, Question 3. Using the function `generate_gaussian_data` we simulate height measurements. Our null hypothesis states $H_0 : \mu_{phd} = \mu_{faculty}$ (same as $H_0 : \mu_{phd} - \mu_{faculty}$=0). To construct the null distribution, we need to describe the distance between our two groups:

   - The naive approach is to look at the difference between the two means: ($\overline{x}_{phd} - \overline{x}_{faculty}$); we will not use this one, but feel free to experiment with it.
   - The t-score, the normalized distance between the two sample means:

$$t = \frac{\overline{x}_{phd} - \overline{x}_{faculty}}{\sqrt{\frac{1}{n_1-1}s^2_{phd} + \frac{1}{n_2-1}s^2_{faculty}}}.$$

   s_{phd} and $s_{faculty}$ are the sample standard deviation for the phd group vs the faculty group and n_1, n_2 are the sample size for the two groups.

   (a) To generate different sample data for measurements of phd and faculty, modify the following variables in the notebook:
      - `height_group_phd` = `generate_gaussian_data(1.75, 0.2, 20)`
      - `height_group_faculty` = `generate_gaussian_data`(1.87, 0.2, 20)

   (b) Your task is to generate samples from (a) the same population $\mu_{phd} = \mu_{faculty}$ and (b) from different populations $\mu_{phd} \neq \mu_{faculty}$.

   (c) Compare the histograms for the two groups and observe the t-scores for both situations. What do you notice?

(d) To estimate the null distribution, the plan is straightforward:

- Generate many groups of phds and faculty;
- For each pair/group of phd and faculty, compute the distance in terms of t-score;
- Store the t-score.

We estimate the null distribution from the stored t-scores. Does this distribution match the estimated t-distribution (with the correct $df = n1 + n2 - 2$)?

4. (**Optional) Null distribution magic coin**. In this exercise, we generate the null distribution for ten coin flips. Open the notebook `Magic_coins.ipynb`. A binomial distribution best simulates a coin flip. Two parameters define the binomial distribution n, the number of trials, and p, the probability of success of a single trial. Our null hypothesis states: "Ileana does not have magic powers.", which means that no supernatural force will influence how the coins fall. We are now ready to generate the null distribution. To estimate the null distribution, we will generate 1000 experiments. Each experiment corresponds to throwing ten coins in the air and counting the number of heads after the coins fall:

   (a) Generate the null distribution for a fair coin ($p = 0.5$)

   (b) Generate the null distribution for a biased coin ($p = 0.7$)

   Compare the probability of obtaining 10 heads in the two cases.

Exercise 3. Estimating p-values. Using the results in the previous step, we can now compute the p-values for the data we generated. Intuitively, the p-value is a measure of surprise for observing a sample in a universe where the null hypothesis is true. Python does have a built-in library for this purpose. Still, to understand p-values, we will estimate them using our data[1]

1. **The p-value of magic coins.** Open the notebook `Magic_coins.ipynb`. The code in Exercise 3, Question 1 will generate the null distribution for flipping the 10 coins for a fair coin. The null hypothesis is H_0 : *"Ileana does not have magic powers."* What is the p-value for obtaining heads for 9 or more coins?

2. **The p-value for the one-sample t-test**. Open the notebook `Lake_water.ipynb` and go to the cell labeled Exercise 3, Question 2. The code in this cell produces Figure 1. The observed sample mean is $\overline{x} = 0.4$ Can you explain what the figure shows?

3. **(Optional)The p-value for two-sample t-test**.

[1] As a sanity check, it is always a good idea to compare the value we estimate with the one calculated by the Python libraries.

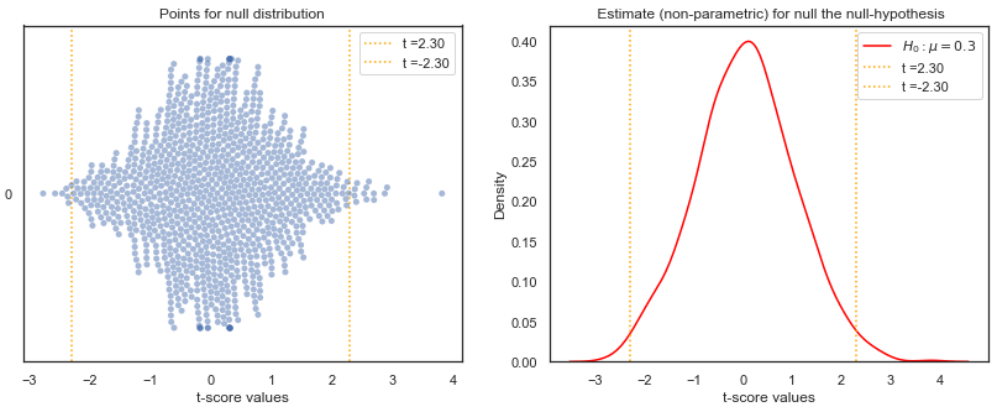

Figure 1: Mystery figure.

In this exercise, you can practice calculating p-values using the built-in Python function `stats.ttest_ind` (part of the `scipy.stats` library), which returns the t-score and p-values for a two-sample t-test. Compute the two-sample t-score and p-value for different sample sizes, such as $n \in \{20, 30, 50, 200\}$ for the following cases

- Case 1: Groups which have different means: $N(1.75, 0.2)$, $N(1.87, 0.2)$
- Case 2: Groups which have same means: $N(1.75, 0.2)$, $N(1.75, 0.2)$
- Case 3: Groups with different sample standard deviations $N(1.75, 0.2)$, $N(1.75, 0.5)$. Welch's t-test compares the mean of two groups and can be used when the standard deviation of the two groups is not equal. Set the flag `equal_var=True` when dealing with unequal variance.

2 TVLA on real traces

Time to move to real side channel traces. To get started, download the files Table 2

Table 2: Files for solving the second part of the tutorial.

File name	**Description**
`TVLA_Tiny_AES.ipynb`	[CODE] `jupyter`. Download here.
`fix_tiny_aes.npy`	[DATASET]. Download it here.
`rand_tiny_aes.npy`	[DATASET]
`TVLA_Xoodoo.ipynb`	[CODE] `jupyter`. Download it here.
`fix_xoodoo.npy`	[DATASET]. Download it here.
`rand_xoodoo.npy`	[DATASET].

Exercise 4. Leakage detection for an unprotected AES implementation Open the file `TVLA_Tiny_AES.ipynb`. The notebook contains the TVLA leakage assessment of an unprotected AES implementation. Follow the code in the notebook and answer the following questions:

1. How many traces are in the `fix_tiny_aes.npy` dataset? How many traces are in the `random_tiny_aes.npy` dataset?
2. How many samples are in one trace?
3. Visually inspect two random traces. Do they appear to be well-aligned?
4. We are ready to compute the *t-scores* and *p-values*. Please note that in function `ss.ttest_ind` we set the flag `equal_var=False`. Why is that?
5. How many sample points have an *absolute* t-score value larger than 4.5?
6. We notice 248 t-score values produced large p-values (> 0.05). What does this tell us?

Exercise 5. Leakage detection for an unprotected Xoodoo implementation Leakage detection for an unprotected Xoodoo implementation. The notebook `TVLA_xoodoo.ipynb` contains the code for reading the power traces. Write the code and answer the following questions:

1. How many traces are in the `fix_xoodoo.npy` dataset? How many traces are in the `random_xoodoo.npy` dataset?
2. How many samples are in one trace?
3. Visually inspect two random traces. Do they appear to be well-aligned?
4. How many sample points have an *absolute* t-score value larger than 4.5?
5. How many samples produce large p-values (> 0.05)? What does this tell us?
6. Change the critical value to $t_{crit} = 5$. To which critical values this corresponds?

Exercise 6. (Optional) Protected AES implementation.

Table 3: Files for solving the optional exercise.

File name	Description
`TVLA_masked_AES.ipynb`	[CODE] `jupyter`. Download here.
`fix_masked_aes.npy`	[DATASET]. Download here.
`rand_masked_aes.npy`	[DATASET].

First-order analysis. Even if we are dealing with a protected implementation, our first task is to check for any first-order leakage. Do you notice any first-order leaks?

Second-order Analysis. For a second-order assessment, we must combine the samples. By combining the traces, we obtain a new second-order trace. The following formula can calculate the size of the new trace:

$$^{n}C_{k} = \frac{n!}{k!(n-k)!} \tag{1}$$

n is the number of samples and k is the number of combinations ($k = 2$ for second-order) As the length of the second-order trace could increase dramatically, we will apply the combining method for a small window of samples. We suggest you combine the 50 samples around sample number 10 000 (from 9 975 to 10 025). Combine these samples by the multiplication function to create a new power trace.

$$New\_Trace\_sample_{x} = sample[i] \times sample[j] \tag{2}$$

$i = 9\,975 \;\; to \;\; 10\,025 \qquad j = (i+1) \; to \;\; 10\,025 \qquad x = 0 \;\; to \;\; 1\,225$

Apply the TVLA test on the new power trace, compare the result with first-order TVLA, and answer the following questions.

1. Implement the second-order TVLA test.
2. Plot the TVLA graph with the threshold lines (+/- 4.5).
3. Count the number of leaky points with respect to the threshold value of 4.5.