Chris Gregg
CS 109

Tutorial #6
November 11, 2026

Tutorial 6: Information Theory, Bootstrapping, and P-Values

Warmup: *populations vs. samples*

What is the difference between the population variance, σ^2, and sample variance, S^2? What is the difference between sample variance, S^2, and variance of the sample mean, $\text{Var}(\bar{X})$?

- Population variance, σ^2: true variance of a population (or random variable).
- Sample variance, S^2: unbiased estimate of true variance based on a random subsample.
- Variance of sample mean, $\text{Var}(\bar{X})$: Amount of spread in the estimation of the true mean.

1 Song of the Quarter

In a past quarter in CS109 there were 167 songs that were voted on as the "song of the quarter". For each song, we have a list of votes where each vote is an integer in the set $\{1, 2, 3, 4, 5\}$. We assume all votes for a song are IID samples from the "true" distribution of CS109 opinion on the song.

For each song i we have m_i votes stored in a list `votes[i]` = $[x_1, x_2, \ldots, x_m]$. We have already calculated:

$$\bar{x}_i = \frac{1}{m_i}\sum_{j=1}^{m_i} x_j \qquad \text{using } \texttt{np.mean(votes[i])}$$

$$\text{var}_i = \frac{1}{m_i}\sum_{j=1}^{m_i} (x_j - \bar{x}_i)^2 \qquad \text{using } \texttt{np.var(votes[i])}$$

$$\text{svar}_i = \frac{1}{m_i - 1}\sum_{j=1}^{m_i} (x_j - \bar{x}_i)^2 \qquad \text{using } \texttt{np.var(votes[i], ddof=1)}$$

(a) Song 1 has $m_1 = 45$ votes. We have calculated:

$$\bar{x}_1 = 3.82 \qquad \text{var}_1 = 1.4 \qquad \text{svar}_1 = 1.5$$

Estimate the probability that the true average rating for song 1 is less than 3.

We can model the sample mean for the rating of song 1 using the Central Limit Theorem, since each vote is IID. Let $\bar{X}_1$ represent the sample mean of votes for song 1.

$$\bar{X}_1 \sim N(\mu_1, \frac{\sigma_1^2}{n})$$

where μ_1, σ_1^2 are the true mean and variance of votes for song 1. We can rearrange the equation:

$$\begin{aligned}
\bar{X}_1 - \mu_1 &\sim N(0, \frac{\sigma_1^2}{n}) \\
\mu_1 - \bar{X}_1 &\sim N(0, \frac{\sigma_1^2}{n}) \\
\mu_1 &\sim N(\bar{X}_1, \frac{\sigma_1^2}{n})
\end{aligned}$$

We can approximate σ_1^2 with $\text{svar}_1 = 1.5$, and we are given $\bar{X}_1 = 3.82$ from our sample where $n = m_1 = 45$. Substituting these values, we have:

$$\begin{aligned}
\mu_1 &\sim N(3.82, \frac{1.5}{45}) \\
P(\mu_1 < 3) &= \Phi(\frac{3 - 3.82}{\sqrt{1.5/45}}) \\
&\approx 0.00000354
\end{aligned}$$

(b) Song 1 has $m_1 = 45$ votes. Song 2 has $m_2 = 36$ votes. We have calculated:

$$\begin{array}{lllll}
\text{Song 1:} & \bar{x}_1 = 3.82 & \text{var}_1 = 1.4 & \text{svar}_1 = 1.5 \\
\text{Song 2:} & \bar{x}_2 = 3.79 & \text{var}_2 = 1.7 & \text{svar}_2 = 1.8
\end{array}$$

What is the probability that the true average of Song 1 is greater than the true average for Song 2?

We again use the CLT to determine distributions of sample means:

$$\bar{X}_1 \sim \mathcal{N}(\mu_1, \frac{\text{svar}_1}{m_1}) \qquad \bar{X}_2 \sim \mathcal{N}(\mu_2, \frac{\text{svar}_2}{m_2})$$

Then, to determine if the average for song 1 is greater than the average for song 2, we can recall that $a > b$ is equivalent to $a - b > 0$ and create a distribution for the difference in averages between the two songs. Note that the variances add because $\text{Var}(X - Y) = \text{Var}(X) + \text{Var}(Y)$.

$$\bar{X}_1 - \bar{X}_2 \sim \mathcal{N}(\mu_1 - \mu_2, \frac{\text{svar}_1}{m_1} + \frac{\text{svar}_2}{m_2})$$

$$P(\bar{X}_1 > \bar{X}_2) = P(\bar{X}_1 - \bar{X}_2 > 0) = 1 - P(\bar{X}_1 - \bar{X}_2 < 0)$$

$$= 1 - \Phi\left(\frac{0 - (3.82 - 3.79)}{\sqrt{\frac{1.5}{45} + \frac{1.8}{36}}}\right)$$

For a standard normal variable Z, we use the symmetry property $1 - \Phi(-z) = \Phi(z)$.

$$P(\bar{X}_1 > \bar{X}_2) = \Phi\left(\frac{3.82 - 3.79}{\sqrt{\frac{1.5}{45} + \frac{1.8}{36}}}\right)$$

2 Entropy & Disease Testing

Choosing a Diagnostic Test with Information Theory

A doctor is deciding which diagnostic test to administer to a patient. There are nine mutually exclusive possibilities: diseases A, B, C, D, E, F, G, H, or having *no disease* (labeled None).

You are given:

- `prior`: A prior distribution $P(x)$ over all diseases $x \in A, B, C, D, E, F, G, H,$ None stored in a dictionary called prior.
- A function `prob_pos_given_disease(test, x)` that returns the probability that a given test yields a positive result if the patient truly has disease x. (For the "None" case, this value represents the false-positive rate.)
- `tests`: A list of available tests, stored as `tests`.

When a test is run, it will return either a + or − result. However, the probability that a test is positive can vary depending on which disease the patient actually has. For example, a test designed to detect disease A may also occasionally return a positive result if the patient has disease B (a cross-reactivity), even though the true disease is not A. So while we only run one test at a time and get one result, that result provides evidence that updates our belief about *all* diseases.

The goal is to determine which test to run by choosing the one that is expected to reduce our uncertainty about the patients condition the most. To do this, write code to compute the expected uncertainty for each test.

Let X be the (mutually exclusive) disease label which can be one of the values in D where $D = \{A, B, C, D, E, F, G, H, \text{None}\}$ and prior $P(x) = P(X = x)$. For test j, define the likelihood

$$\ell_j(x) \;\equiv\; P(+ \mid \text{test}_j,\; X = x) = \texttt{prob_pos_given_disease}(\text{test}_j, x).$$

For each test, we need to know the probability that it returns positive and negative results.

$$P(+ \mid \text{test}_j) = \sum_{x \in D} P(x)\, \ell_j(x), \qquad P(- \mid \text{test}_j) = 1 - P(+ \mid \text{test}_j).$$

We have access to the probability that a test returns positive given the patient truly has that disease, but we want it in the other direction: probability the patient truly has that disease given the test results. So we use Bayes' rule.

$$P(X = x \mid +, \text{test}_j) = \frac{P(x)\, \ell_j(x)}{\sum_{k \in D} P(k)\, \ell_j(k)}, \qquad P(X = x \mid -, \text{test}_j) = \frac{P(x)\, [1 - \ell_j(x)]}{\sum_{k \in D} P(k)\, [1 - \ell_j(k)]}.$$

Posterior entropies and expected entropy.

$$\mathrm{H}\big(X \mid +, \text{test}_j\big) = -\sum_{x \in D} P(X = x \mid +, \text{test}_j)\, \log_2 P(X = x \mid +, \text{test}_j),$$

$$\mathrm{H}\big(X \mid -, \text{test}_j\big) = -\sum_{x \in D} P(X = x \mid -, \text{test}_j)\, \log_2 P(X = x \mid -, \text{test}_j),$$

$$\mathbb{E}\big[\mathrm{H}(X) \mid \text{test}_j\big] = P(+ \mid \text{test}_j)\, \mathrm{H}\big(X \mid +, \text{test}_j\big) + P(- \mid \text{test}_j)\, \mathrm{H}\big(X \mid -, \text{test}_j\big).$$

Doing this math for all of the diseases for all of tests by hand would take forever. Good thing we are computer scientists!! To the code:

```
import math

# note, the functions on the next line would depend on actual datasets!! This
    problem is just to practice writing the code. You will practice with real
    data on your next pset!!

from utils import load_from_data, prob_pos_given_disease

prior, tests = load_from_data()

def normalize_pmf(pmf):
    total = sum(pmf.values())
    for x in pmf:
        pmf[x] /= total
    return pmf
```

```
def compute_uncertainty(disease_pmf):
    H = 0.0
    for x in disease_pmf:
        p = disease_pmf[x]
        if p > 0:
            H += -p * math.log2(p)
    return H

def compute_prob_pos(disease_pmf, test):
    # P(+ | test) = sum_x P(x) * P(+ | test, x)
    p_pos = 0.0
    for x in disease_pmf:
        p_pos += disease_pmf[x] * prob_pos_given_disease(test, x)
    return p_pos

def compute_posterior(disease_pmf, test, outcome_is_positive):
    posterior_unnorm = {}
    for x in disease_pmf:
        like = prob_pos_given_disease(test, x)
        like = like if outcome_is_positive else (1 - like)
        posterior_unnorm[x] = disease_pmf[x] * like
    return normalize_pmf(posterior_unnorm)

def compute_expected_uncertainty(disease_pmf, test):
    # E[H | test] = P(+) H(P(.|+,test)) + P(-) H(P(.|-,test))
    p_pos = compute_prob_pos(disease_pmf, test)
    p_neg = 1 - p_pos

    pmf_pos = compute_posterior(disease_pmf, test, True)  # outcome: +
    pmf_neg = compute_posterior(disease_pmf, test, False) # outcome: -

    H_pos = compute_uncertainty(pmf_pos) if p_pos > 0 else 0.0
    H_neg = compute_uncertainty(pmf_neg) if p_neg > 0 else 0.0

    return p_pos * H_pos + p_neg * H_neg

def chose_best_test(disease_pmf, tests):
    best_test = None
    best_uncertainty = None
    for test in tests:
        expected_uncertainty = compute_expected_uncertainty(disease_pmf, test
            )
        if best_test is None or expected_uncertainty < best_uncertainty:
            best_test = test
            best_uncertainty = expected_uncertainty
    return best_test, best_uncertainty
```

3 Variance of Hemoglobin Levels

A medical researcher treats patients with dangerously low hemoglobin levels. She has formulated two slightly different drugs and is now testing them on patients. First, she administered drug A to one

group of 50 patients and drug B to a separate group of 50 patients. Then, she measured all the patients' hemoglobin levels post-treatment. For simplicity, assume that all variation in the patient outcomes is due to their different reactions to treatment.

The researcher notes that the sample mean is similar between the two groups: both have mean hemoglobin levels around 10g/dL. However, drug B's group has a **sample variance** that is 3 $(g/dL)^2$ **greater** than drug A's group. The researcher thinks that patients respond to drugs A and B differently. Specifically, she wants to make the scientific claim that drug A's patients will end up with a significantly different spread of hemoglobin levels compared to drug B's.

You are skeptical. It is possible that the two drugs have practically identical effects and that the observed different in variance was a result of chance and a small sample size, i.e. the **null hypothesis**. Calculate the probability of the null hypothesis using bootstrapping. Here is the data. Each number is the level of an independently sampled patient:

Hemoglobin Levels of Drug A's Group (S^2 = 6.0): 13, 12, 7, 16, 9, 11, 7, 10, 9, 8, 9, 7, 16, 7, 9, 8, 13, 10, 11, 9, 13, 13, 10, 10, 9, 7, 7, 6, 7, 8, 12, 13, 9, 6, 9, 11, 10, 8, 12, 10, 9, 10, 8, 14, 13, 13, 10, 11, 12, 9

Hemoglobin Levels of Drug B's Group (S^2 = 9.1): 8, 8, 16, 16, 9, 13, 14, 13, 10, 12, 10, 6, 14, 8, 13, 14, 7, 13, 7, 8, 4, 11, 7, 12, 8, 9, 12, 8, 11, 10, 12, 6, 10, 15, 11, 12, 3, 8, 11, 10, 10, 8, 12, 8, 11, 6, 7, 10, 8, 5

How would this calculation be different if you were interested in looking at the statistical significance of the difference in sample mean? Or the 95th percentile?

```
def bootstrap(sample1, sample2):
   # make the universal population
   totalSample = copy.deepcopy(sample1)
   totalSample.extend(sample2)

   # Run a bootstrap experiment
   countDiffGreaterThanObserved = 0
   print(f"Starting␣Bootstrap")
   for i in range(50000):
      # resample and recalculate the statistic
      resample1 = resample(totalSample, len(sample1))
      resample2 = resample(totalSample, len(sample2))
      resampleStat1 = calcSampleVariance(resample1)
      resampleStat2 = calcSampleVariance(resample2)
      diff = abs(resampleStat2 - resampleStat1)
      # count how many times the statistic is more extreme
      if diff >= 3: countDiffGreaterThanObserved += 1

   # compute the p-value
   p = float(countDiffGreaterThanObserved) / 50000
   print(f"p-value:␣{p}")
```

For this data, the two-tailed (e.g. using absolute value) test returns a null hypothesis probability **p = 0.12**. There is a pretty decent chance that the observed difference in sample variance was from random chance – and it doesn't fall under what scientists often call "statistically significant."