CH7 - Hypothesis and Inference

October 2, 2025

1 Hypothesis Testing and Inference

We often want to test whether a certain hypothesis is likely to be true. Hypotheses are typically assertions like “this coin is fair’ or”data scienctist love python” that can be translated into statistics about data. As we estabtilished before we can approximate a binomial distribution with a normal apporaximation such that

$$\mu = np \quad \sigma = \sqrt{np(1-p)}$$

[26]:
```
from typing import Tuple
import math

def normal_approximatiom_to_binomial(n: int, p: float) -> Tuple[float, float]:
    """ Returns the mean and standard deviation corresponding to a␣
 ↪Binomial(n,p) """
    mu = n*p
    sigma = math.sqrt(n*p*(1-p))

    return mu, sigma

normal_approximatiom_to_binomial(1000,0.2)
```

[26]:
```
(200.0, 12.649110640673518)
```

When a random variable follows a normal distribution, we can use normal_cdf to figure out the probability that its realised value lies within or outside a particular interval. As default normal_cdf() returns the area under the curve from a point and below.

[2]:
```
from helpers import normal_cdf

normal_probability_below = normal_cdf
print(f"The Probabtility below Z=1: {normal_probability_below(1)}")

#The probability that is is above the threshold
def normal_probability_above(lo: float, mu: float=0, sigma: float=1) -> float:
    """ Returns Probability = P(Z <= z) """
    return 1 - normal_cdf(lo,mu,sigma)
```

```
print(f"The Probabtility above Z=1: {normal_probability_above(1)}")

def normal_probability_between(lo: float, hi: float, mu: float=0, sigma:␣
 ↪float=1) -> float:
    return normal_cdf(hi, mu,sigma) - normal_cdf(lo, mu, sigma)

print(f"The Probabtility inbetween z=[-1,1]:␣
 ↪{normal_probability_between(-1,1)}")

def normal_probability_outside(lo: float, hi: float, mu: float=0, sigma:␣
 ↪float=1) -> float:
    return 1-normal_probability_between(lo,hi,mu,sigma)

normal_probability_outside(-1,1)
print(f"The Probabtility outside z=[-1,1]: {normal_probability_outside(-1,1)}")
```

```
The Probabtility below Z=1: 0.8413447460685429
The Probabtility above Z=1: 0.15865525393145707
The Probabtility inbetween z=[-1,1]: 0.6826894921370859
The Probabtility outside z=[-1,1]: 0.31731050786291415
```

We can do the reverse again. Say we are given a certain level of likelihood and we want to know what region thaat pertains to. For example, if we want to find the interval centered at the mean and containing 60% probability, then we find the cutoffs where the upper and lower tails each contain 20% of the probability (given symmmetry).

Show the three intervals with a chart showing the area and bounds we want to compute.

[31]:
```
from typing import Tuple
from helpers import inverse_normal_cdf

def normal_upper_bound(probability: float, mu: float=0, sigma: float=1) ->␣
 ↪float:
    """Returns the z for which P(Z <= z) = probabilty """
    return inverse_normal_cdf(probability,mu, sigma)

probability= 0.8413447460685429
print(f"The z, for which P(Z <= z) = {probability} is z =␣
 ↪{normal_upper_bound(probability)}") #we should get 1 looking at the results␣
 ↪from above where The Probabtility below Z=1: 0.8413447460685429

def normal_lower_bound(probability: float, mu: float=0, sigma: float=1) ->␣
 ↪float:
    """ Returns the z for which P(Z >= z) = probability """
    return inverse_normal_cdf(1-probability, mu, sigma)
```

```
print(f"The z, for which P(Z >= z) = {probability} is z =␣
 ↪{normal_lower_bound(probability)}") #we should get -1 looking at the results␣
 ↪from above where The Probabtility below Z=1: 0.8413447460685429


def normal_two_sided_bounds(probability: float, mu: float=0, sigma:float=1) ->␣
 ↪Tuple[float, float]:
    """ Returns the symmetric (about the mean) bounds that contain the␣
 ↪specified prob """
    tail_probability = (1-probability)/2
    upper_bound = normal_lower_bound(tail_probability, mu, sigma)
    lower_bound = normal_upper_bound(tail_probability, mu, sigma)

    return lower_bound, upper_bound

print(f"The z, for which P(z_1 <= Z <= z_2) = {probability} is z =␣
 ↪{normal_two_sided_bounds(probability)}")
```

```
The z, for which P(Z <= z) = 0.8413447460685429 is z = 0.9999847412109375
The z, for which P(Z >= z) = 0.8413447460685429 is z = -0.9999847412109375
The z, for which P(z_1 <= Z <= z_2) = 0.8413447460685429 is z =
(-1.4096832275390625, 1.4096832275390625)
```

1.1 Example of Hypothesis test: is a coin fair?

Say that we will flip a coin $n = 1000$ times. If our hypothesis of fairness is true (that the coin is fair and therefore gives heads 50% of the time and tails 50% of the time), X shoulder be distributed approximately normally with mean $\mu = 500 (= np = 1000 * 0.5)$ and standard deviation, $\sigma = 15.8 (= \sqrt{np(1-p)} = \sqrt{250})$

[25]:
```
num_of_flips = 1000
probability = 0.5
mu_0, sigma_0 = normal_approximatiom_to_binomial(num_of_flips, probability)
print(f"mean = {mu_0} , sigma = {sigma_0}")
```

```
mean = 500.0 , sigma = 15.811388300841896
```

But the question remains, how willing are we to make a type 1 error ("false positive"), in which we reject the hypothesis even though it is true. For example, lets say we run the simulation 1000 times and we get a head 350 times, how can we know if we should take this to mean the coin is unfair?

To do this, we must make a decision about significance. Lets define H_0 as the null hypothesis, that represents some default position and some alternative hypothesis H_1. We typically set the significance to 5% or 1% and say that we will reject the null hypothesis H_0 if it falls in outside a set of bounds. If we choose 5%, this is to say that our confidence interval is 95. Therefore, if the experiment produces values withinn this range we accept H_0, otherwise reject.

Assume probability p really equals 0.5, there is a 5% chance we observe an X that lies outside this interval.

[32]:
```
#given significance at 5%, we can compute the bounds that give us 95%␣
 ↪probability - which is the confidence interval.
prob=0.95
lower, upper = normal_two_sided_bounds(prob, mu_0, sigma_0)
print(f"{prob*100}% Confidence interval ({lower}, {upper})")
```

```
95.0% Confidence interval (469.011020350622, 530.988979649378)
```

Imagein we get 530 heads after running the experiment. We can compute the p-values, which is the probability - assuming H_0 is true, that we would see a value at least as extreme as the one we actually observed. We use the Continuity correction of $z \pm 0.5$

[39]:
```
def two_sided_p_value(x: float, mu: float=0, sigma: float=1) -> float:
    """ How likely are we to see a value at least as extreme as x """
    if x >= mu:
        return 2 * normal_probability_above(x, mu, sigma)
    else:
        return 2 * normal_probability_below(x, mu, sigma)

two_sided_p_value(529.5, mu_0, sigma_0)


import random

extreme_value_count = 0
num_of_experiments = 1000
for _ in range(num_of_experiments):
    num_heads = sum(1 if random.random() < 0.5 else 0 for _ in range (1000))
    if num_heads >=530 or num_heads <= 470:
        extreme_value_count +=1

print(f"How many times did we get an extreme value: {extreme_value_count}")
```

```
How many times did we get an extreme value: 59
```

[ ]:
```

```