Statistical Rethinking: Lecture 02

with Python code

Tossing the globe example

The problem: to estimate the proportion of water covering the globe by "tossing it", i.e. sampling water and land observations. Imagine being away from the planet and being able only to send probes to get information of whether it landed on land "L" or water "W". Alternatively, imagine a globe replica being tossed at a crowd and taking note of where in the globe, water or land, the hand of the person who got it first touched the globe.

In a frequentist approach, we could use the relative frequencies of "W" or "L" as the estimate for the water and land proportions, respectively. Here we use a Bayesian approach: given a sample consisiting on W and L observations of water and land, what value of p, the proportion of water covering the globe, is more likely to reproduce the data?

Since p can take an infinite number of values in the range from 0 to 1, let us consider a 4-sided "globe", a tetrahedron or 4-sided die. This allows discretizing the problem. At the end we can take the limit of polyhedron sides reaching infinity, recovering a proper spherical globe.

Recall the fundamental principle in Bayesian analysis

> For each possible explanation of the sample, count the ways each explanation can reproduce the data. Explanations with the largest number of ways to reproduce the data are more likely.

And the basic workflow

1) Define a generative model of the sample (generate synthetic data)
2) Define a specific estimand
3) Design a way to produce the estimate (estimator)
4) Test 3) using 1) (use the estimator to get an estimate in controlled situation, where you know the answer)
5) Analyze sample (real data)

1) The generative model

Let us define a function to generate a sample of water "W" or land "L" observations. If p is the proportion of water covering the globe, then the probability of observing "W" is p and "L" is $1 - p$.

```
def toss_globe(proportion, N):
    return np.random.choice(["W", "L"], size=N, p=[proportion, 1 - proportion])
```

We can toss a globe covered 70% by water ten times by calling `toss_globe(p=0.7, N=10)`, which gives

```
array(['L', 'W', 'W', 'W', 'W', 'W', 'W', 'W', 'L', 'W'], dtype='<U1')
```

2) The estimand

Our estimand will be the number of ways each explanation can reproduce the sample. In this case, the number of ways each proportion of water p covering the globe can reproce a series of observations. That value of p with the largest number of ways to reproduce the data is more plausible to be the true value. Equivalently, instead of counting the ways each p can reproduce the data, we can normalize the count by the total number of ways of explaining the data with all the possible values of p. This way, we'll be working with a probability mass distribution, thus our estimand will be the probability distribution for the prorportion of water p.

3) The estimator

For a total of $N = W + L$ tosses, the expected number of water and land observations are given by:

$$W \quad \text{or} \quad L = (4p)^W (4 - 4p)^L$$

which count the ways each explanation – the proportion of water p – can reproduce the sample. This is our estimator. Let us code it.

Using the sample observed in the previous toss, we can calculate the number of ways each proportion (explanation) can reproduce the sample as follows

```
W, L = np.sum(sample=='W'), np.sum(sample=='L')
proportions = [0, 0.25, 0.5, 0.75, 1]
ways = [(4 * p)**W * (4 - 4 * p)**L for p in proportions]
ways
```

```
[0, 9.0, 1024.0, 6561.0, 0]
```

Normalizing by the total number of ways we can calculate the probabilities each proportion p has to reproduce the data. The plot is shown in Figure 1.

```
probabilities = np.array(ways)/np.sum(ways)
probabilities
```

```
array([0.        , 0.00118515, 0.1348433 , 0.86397156, 0.        ])
```

```
fig, ax = mplt.subplots()
ax.bar([str(prop) for prop in proportions], probabilities)
ax.set_xlabel('water proportions')
ax.set_ylabel('proportion probability')

mplt.show()
```

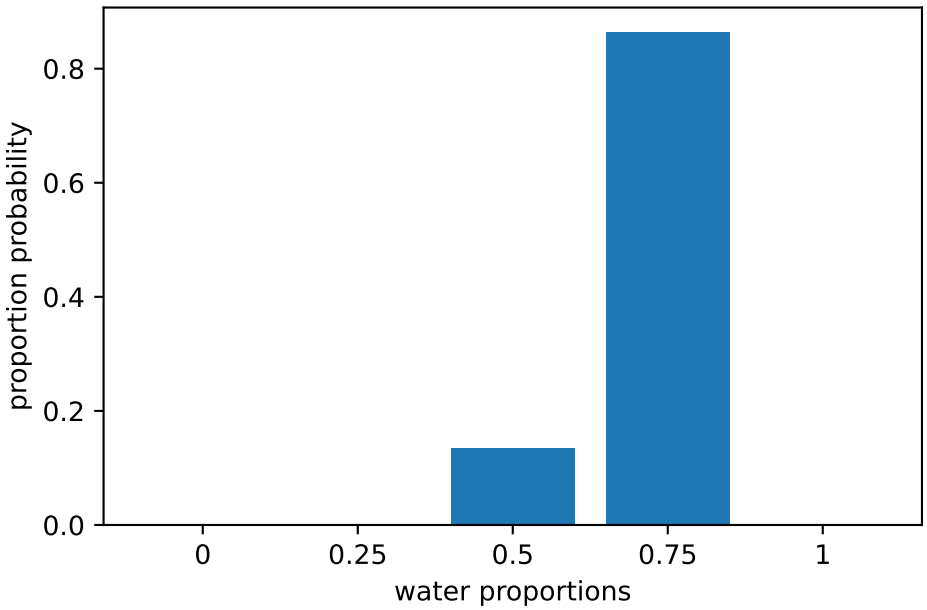


Figure 1: Bar plot for the plausability of each proportion of

This estimator calculates the estimate probability for each explanation p of the sample. In the Bayesian jargon, it computes the posterior distribution for p. We code it into a function for future convnience:

```
def compute_posterior(sample, proportions):
    W, L = np.sum(sample=='W'), np.sum(sample=='L')
    ways = [(4 * p)**W * (4 - 4 * p)**L for p in proportions]
    return np.array(ways)/np.sum(ways)
```

4) Testing

We have thus far succesfully designed the generative model, defined the estimand and designed the statistical estimator. We have also anticipated one estimate by applying the estimator to the first sample we got from the generative model. Now focus on testing if the estimand and estimator are valid.

Let us simulate extreme cases.

Water planet

Generate sample with water proportion equal to 1.

```
sample = toss_globe(proportion=1, N=10)
sample
```

```
array(['W', 'W', 'W', 'W', 'W', 'W', 'W', 'W', 'W', 'W'], dtype='<U1')
```

Compute the posterior

```
proportions = [0, 0.25, 0.5, 0.75, 1]
compute_posterior(sample, proportions)
```

```
array([0.00000000e+00, 9.01997925e-07, 9.23645876e-04, 5.32620755e-02,
       9.45813377e-01])
```

We see the most likely proportion is the correct $p = 1$.

Land planet

Generate sample with water proportion equal to 0.

```
sample = toss_globe(proportion=0, N=10)
sample
```

```
array(['L', 'L', 'L', 'L', 'L', 'L', 'L', 'L', 'L', 'L'], dtype='<U1')
```

Compute the posterior

```
proportions = [0, 0.25, 0.5, 0.75, 1]
compute_posterior(sample, proportions)
```

```
array([9.45813377e-01, 5.32620755e-02, 9.23645876e-04, 9.01997925e-07,
       0.00000000e+00])
```

We see the most likely proportion is the correct $p = 0$. Now lets play with large number of trials with $p = 0.5$ and see if we recover this frequency of water of observations

```
sample = [toss_globe(proportion=0.5, N=10) for _ in range(1_000)]
sample = np.concatenate(sample)  # reshape it 1D array
np.sum(sample=='W')/np.size(sample)  # compute the frequency of water
```

```
0.5019
```

Abstracting: Bayes theorem

Let's turn the concrete example we have worked on into some more abstract reasoning before continuing.

The whole idea behind Bayesian analysis is that explanations with the largest number of ways of explaining the data are more likely. Bayesian statistics is a counting problem. Where's Bayes theorem in it?

The counting problem, when normalized by the total number of possible outcomes, becomes a probability problem. The probability for observing W water and L land instances when tossing the globe for a given proportion p of water is given by the Biominal distribution

$$\Pr(W, L|p) = \frac{(W+L)!}{W!L!} p^W (1-p)^L$$

Saying that proportions with more ways to reproduce the sample are more plausible is equivalent to attribute higher plausability to p rendering the the highest probability $\Pr(p|W, L)$. If

you are familiar with Bayes theorem, you know you can calculate this "inverse" conditional as

$$\Pr(p|W,L) = \frac{\Pr(W,L|p)\Pr(p)}{\Pr(W,L)}$$

The $\Pr(W,L)$ term on the denominator is the normalization constant, equivalent to `np.sum(ways)` in our code. It is formally calculated in the most general case as

$$\Pr(W,L) = \mathbb{E}[\Pr(W,L|p)] = \int \Pr(W,L|p)\Pr(p)\mathrm{d}p$$

The main theme in Bayesian statistics, however, is not Bayes theorem. Bayes theorem derives from the basic rules of probabilities and can be used in non-bayesian analysis. The main distinction of Bayesian reasoning is practicing the philosophy of probabilities as measures of beliefs or creedances. $\Pr(W,L|p)$, is the likelihood function and $\Pr(p)$ is the prior distribution: the initial guess for the distribution of the explanations. Calculating $\Pr(p|W,L)$ can be seen as updating the initial belief on p given by the prior in face of the observed data (likelihood). If new observations come, the posterior is used as the new prior and the process is itrated, updating our belief about the distribution of p.

Grid approximation

What if we allow our 4-sided globe to have instead 11 sides? Well, now the possible proportions of water are

```
proportions11 = np.linspace(0, 1, 11, endpoint=True)
proportions11
```

```
array([0. , 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1. ])
```

Let's assume $p = 0.7$ and toss the globe 10 times. Let's calculate the posterior and repeat this process for a 21-sided globe.

```
sample11 = toss_globe(proportion=0.7, N=10)
sample21 = toss_globe(proportion=0.7, N=10)

post11 = compute_posterior(proportions=proportions11, sample=sample11)

proportions21 = np.linspace(0, 1, 21, endpoint=True)
```

```
post21 = compute_posterior(proportions=proportions21, sample=sample21)
```

We plot the results in Figure 2.

```
fig, (ax, ay) = mplt.subplots(1, 2, figsize=(10, 5))
data = [f'{prop:.1f}' for prop in proportions11]
ax.bar(data, post11)
ax.set_title('11-sides')
ax.set_xticklabels(labels=data, rotation=90)
ax.set_ylabel('proportion probability')

data = [f'{prop:.2f}' for prop in proportions21]
ay.set_xticklabels(labels=data, rotation=90)
ay.bar(data, post21)
ay.set_title('21-sides')
fig.supxlabel('water proportions')
fig.tight_layout()
mplt.show()
```

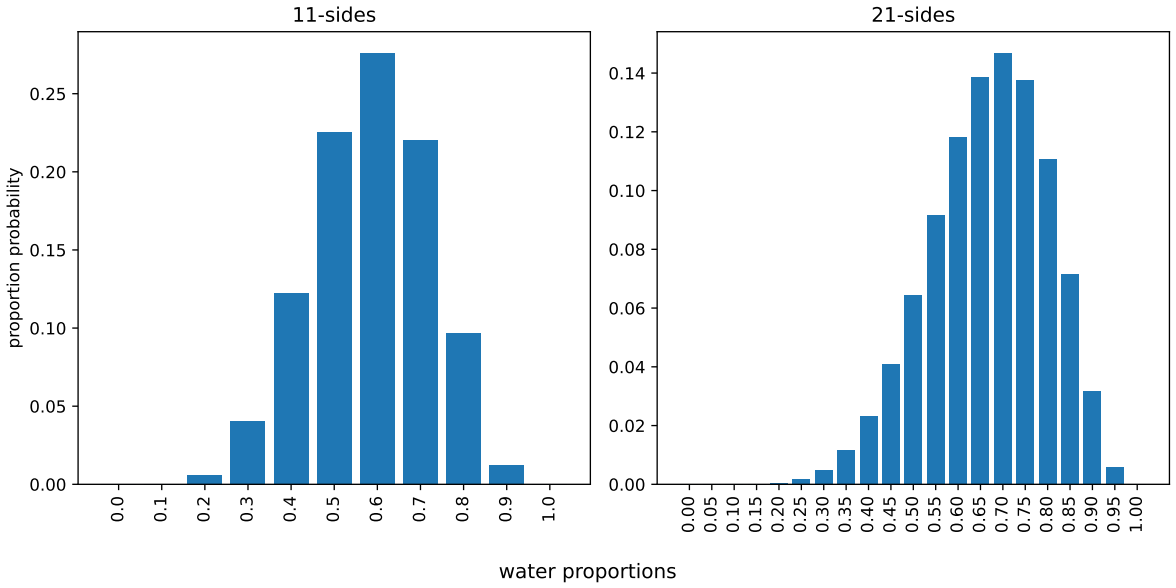


Figure 2: Distribution of p for 11- and 21-sided globes

Increasing the number of sides makes our polygonal dice closer to a proper globe. Approximating the posterior of the globe with increasing number of sides is an example of the *grid approximation*. The grid is the number of sides.

Continuum limit

NRecall that

$$\Pr(W, L|p) = \frac{1}{Z} p^W (1-p)^L$$

where the "partition function" reads $Z = \sum_p p^W (1-p)^L = \text{W!L!/(W+L)!}$. As the number of sides approach infinity, the sum approaches the integral

$$Z = \int p^W (1-p)^L \mathrm{d}p = \mathrm{B}(W+1, L+1) = \frac{W!L!}{(W+L+1)!}, \quad W, L \in \mathbb{Z}$$

where $\mathrm{B}(W+1, L+1)$ is the Beta function. We recognize the Beta distribution

$$\Pr(W, L|p) = \mathrm{Beta}(W+1, L+1) = \frac{(W+L+1)!}{W!L!} p^W (1-p)^L$$

Posterior: updating the prior

Before any observations, we assume any proportion p is equally plausible. So the probabiltiy for p is a constant normalized so it reads as probability:

$$\Pr(p)_0 = \frac{u}{Z_0},$$

where Z_0 is basically the `np.sum(ways)` term in our previous code snippets.

The first observation, say, water, allows us to calculate the posterior:

$$\Pr(p)_1 = \frac{p^1 (1-p^0) \Pr(p)_0}{Z_1}$$

and so the second observation gives $\Pr(p)_2$ in the same manner, with Z_2 being whatever constant needed to normalize the probability to 1. For the beta distribution is easy to find analytical expressions for the posteriors as we iterate. Instead, let us generalize our function to update posteriors numerically. First we import `beta` from `scipy`

```
from scipy.stats import beta
```

And now we write our function

```
def compute_posterior(proportions, sample, prior):
    W, L = np.sum(sample=='W'), np.sum(sample=='L')
    Wprior = np.sum(prior=='W')
```

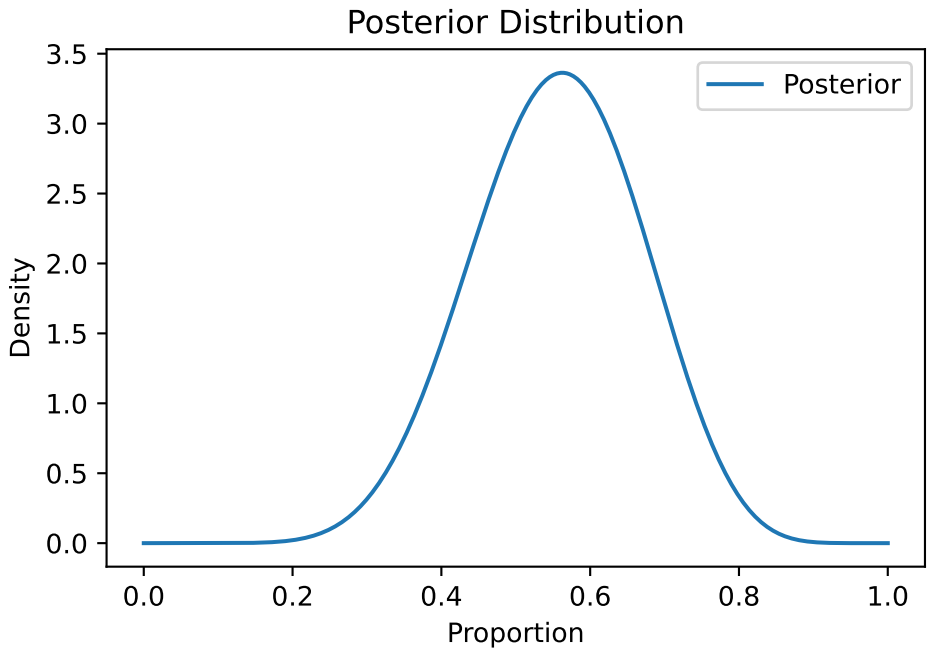
```
    Lprior = np.sum(prior=='L')
    return beta.pdf(proportions, W + Wprior, L + Lprior)
```

```
x_values = np.linspace(0, 1, 1000)

the_sample = np.array(['W', 'W', 'L', 'W', 'L', 'W', 'L', 'L', 'W'])
prior = np.array(['W', 'W', 'L', 'W', 'L', 'W', 'L', 'L', 'W'])

# Compute the posterior values for the given x values
posterior_values = compute_posterior(x_values, the_sample, prior)

# Plot the curve
mplt.figure()
mplt.plot(x_values, posterior_values, label='Posterior')
mplt.title('Posterior Distribution')
mplt.xlabel('Proportion')
mplt.ylabel('Density')
mplt.legend()
mplt.show()
```



Basically we count the ocurrences of water and land in the sample and in the prior, and use

them as parameters to sample from the beta distribution. for a continuous .

Working with the posterior

Sampling the posterior

Suppose we constructed the posterior for the observation of 6 water and 3 land. We can then sample the posterior distribution Beta$(p, 6+1, 3+1)$. A thousand samples read

```
post_samples = np.random.beta(6+1, 3+1, 1000)
```

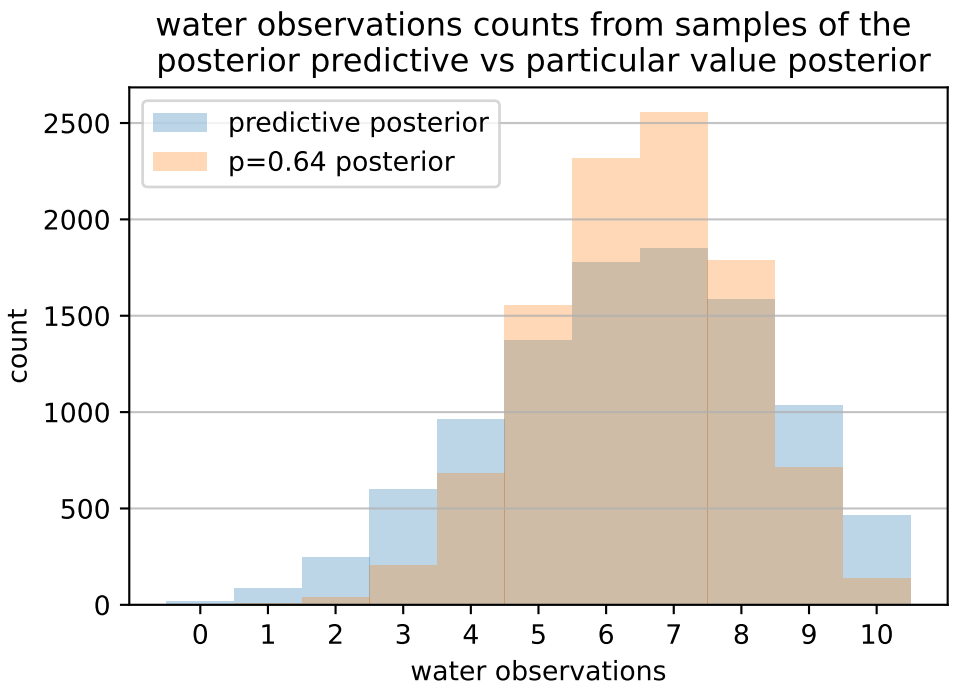
Posterior Predictive Distribution

Given the posterior distribution, how can we make predictions? Well, we could choose a particular value of p, maybe the mode, the peak of the distribution, the most probable value, and use it to make predictions about water and land observations using the likelihood function or the generative model. But how do we incorporate the uncertainty in p in our estimates? How do we propagate the associated error of choosing a particular value and then doing a prediction? The solution is to sample values for p from the posterior, then calculate the likelihood for a given p or sample the generative model and construct a posterior distribution from the sampled values.

Below, we sample the water proportions from the posterior distribution and then sample the generative model 10-times for each value of proportion to obtain the posterior predictive distribution. We also sample the posterior at a particular value of $p = 0.64$ and compare the results.

```
post_samples = np.random.beta(6+1, 3+1, 10_000)
pred_post = [np.sum(toss_globe(p, 10)=='W') for p in post_samples]

pred64 = [np.sum(toss_globe(0.65, 10)=='W') for _ in range(10_000)]
mplt.hist(pred_post, bins=np.arange(12)-0.5, alpha=0.3, label='predictive posterior')
mplt.hist(pred64, bins=np.arange(12)-0.5, alpha=0.3, label='p=0.64 posterior')
mplt.legend()
mplt.xlabel('water observations')
mplt.ylabel('count')
mplt.xticks(range(11))
mplt.grid(axis='y', alpha=0.75)
mplt.title('water observations counts from samples of the \n posterior predictive vs parti
mplt.show()
```



Notice the Predictive posterior is much more spread out than the posterior for $p = 0.64$. This is because it incorporates the uncertainty in the estimate for p.