Section 03 - Comparing hypotheses

MCB 111

September 23, 2022

A recap

For the same data D, how to decide between two competing hypotheses H_1 and H_2?

Naturally, one wants to know which hypothesis is more probable, relatively speaking. So,

$$\text{Ratio of posteriors} = \frac{\mathbb{P}(H_1|D)}{\mathbb{P}(H_2|D)} \tag{1}$$

We can re-write it using the Bayes theorem:

$$\text{Ratio of posteriors} = \frac{\mathbb{P}(D|H_1)\mathbb{P}(H_1)}{\mathbb{P}(D|H_2)\mathbb{P}(H_2)} = \text{Bayes factor} \times \text{Ratio of priors} \tag{2}$$

We can re-write it using the Bayes theorem:

$$\text{Ratio of posteriors} = \frac{\mathbb{P}(D|H_1)\mathbb{P}(H_1)}{\mathbb{P}(D|H_2)\mathbb{P}(H_2)} = \text{Bayes factor} \times \text{Ratio of priors} \tag{2}$$

If no prior information is incorporated, Bayes factor is what we need

How to compute $\mathbb{P}(D|H)$ given a hypothesis that contains a range of possible parameter values?

How to compute $\mathbb{P}(D|H)$ given a hypothesis that contains a range of possible parameter values?

Use **Marginalization**:

$$\mathbb{P}(D|H) = \int_{\lambda_1} \mathbb{P}(D|\lambda_1, H) p(\lambda_1|H) \, \mathrm{d}\lambda_1 \tag{3}$$

How to compute $\mathbb{P}(D|H)$ given a hypothesis that contains a range of possible parameter values?

Use **Marginalization**:

$$\mathbb{P}(D|H) = \int_{\lambda_1} \mathbb{P}(D|\lambda_1, H) p(\lambda_1|H) \, \mathrm{d}\lambda_1 \tag{3}$$

For more than one parameters:

$$\mathbb{P}(D|H) = \int_{\lambda_1} \cdots \int_{\lambda_n} \mathbb{P}(D|\lambda_1, \cdots, \lambda_n, H) p(\lambda_1, \cdots, \lambda_n|H) \, \mathrm{d}\lambda_1 \cdots \mathrm{d}\lambda_n \tag{4}$$

How to compute $\mathbb{P}(D|H)$ given a hypothesis that contains a range of possible parameter values?

Use **Marginalization**:

$$\mathbb{P}(D|H) = \int_{\lambda_1} \mathbb{P}(D|\lambda_1, H) p(\lambda_1|H) \, \mathrm{d}\lambda_1 \tag{3}$$

For more than one parameters:

$$\mathbb{P}(D|H) = \int_{\lambda_1} \cdots \int_{\lambda_n} \mathbb{P}(D|\lambda_1, \cdots, \lambda_n, H) p(\lambda_1, \cdots, \lambda_n|H) \, \mathrm{d}\lambda_1 \cdots \mathrm{d}\lambda_n \tag{4}$$

$p(\lambda_1|H)$,$p(\lambda_1, \cdots, \lambda_n|H)$ contain what the hypothesis says about the parameters. Make sure they are normalized:

$$\int_{\lambda_1} \cdots \int_{\lambda_n} p(\lambda_1, \cdots, \lambda_n|H) \, \mathrm{d}\lambda_1 \cdots \mathrm{d}\lambda_n = 1 \tag{5}$$

(Some blackboard examples..)

Concrete example

The bacteria mutation time problem:

Given a set of mutation times $D = \{t_1, t_2, \cdots, t_N\}$, what's the probability of D given the following hypothesis?

$$H_1 : \text{ They follow an exponential distribution with parameter } \lambda \tag{6}$$

Concrete example

The bacteria mutation time problem:

Given a set of mutation times $D = \{t_1, t_2, \cdots, t_N\}$, what's the probability of D given the following hypothesis?

$$H_1 : \text{ They follow an exponential distribution with parameter } \lambda \tag{6}$$

What does the hypothesis say about λ? (aka, how to choose $p(\lambda|H_1)$?)

Concrete example

The bacteria mutation time problem:

Given a set of mutation times $D = \{t_1, t_2, \cdots, t_N\}$, what's the probability of D given the following hypothesis?

$$H_1 : \text{ They follow an exponential distribution with parameter } \lambda \tag{6}$$

What does the hypothesis say about λ? (aka, how to choose $p(\lambda|H_1)$?)

- "$\lambda \leq 0$" is impossible
- "$\lambda >$ The age of the Earth" is unlikely

Concrete example

The bacteria mutation time problem:

Given a set of mutation times $D = \{t_1, t_2, \cdots, t_N\}$, what's the probability of D given the following hypothesis?

$$H_1 : \text{ They follow an exponential distribution with parameter } \lambda \tag{6}$$

What does the hypothesis say about λ? (aka, how to choose $p(\lambda|H_1)$?)

- "$\lambda \leq 0$" is impossible
- "$\lambda >$ The age of the Earth" is unlikely

Perhaps a uniform distribution over a range: $p(\lambda|H_1) = \dfrac{1}{\lambda^+ - \lambda^-} = \dfrac{1}{\sigma}$

(Some additional caveats: two hypotheses may differ only at $p(\lambda|H_i)$!)

Concrete example

The probability of data given H_1 becomes:

$$\mathbb{P}(D|H_1) = \frac{1}{\sigma}\int_{\lambda^-}^{\lambda^+} \mathbb{P}(D|\lambda, H_1)\,\mathrm{d}\lambda \tag{7}$$

Since each data point is independently produced, the full likelihood can be factorized by the likelihood of each data point:

$$\mathbb{P}(D|H_1) = \frac{1}{\sigma}\int_{\lambda^-}^{\lambda^+} \prod_{i=1}^{N} \mathbb{P}(t_i|\lambda, H_1)\,\mathrm{d}\lambda \tag{8}$$

What does $\mathbb{P}(t_i|\lambda, H_1)$ look like?

Concrete example

$\mathbb{P}(t|\lambda, H_1)$ should be the density of an exponential distribution, but truncated at the starting time (t_-) and the stopping time (t_+) of the experiment:

$$\mathbb{P}(t|\lambda, H_1) = \frac{\exp(-t/\lambda)}{Z(\lambda)} \tag{9}$$

$Z(\lambda)$ is a normalization factor. You can get it by solving:

$$\int_{t_-}^{t_+} \frac{\exp(-t/\lambda)}{Z(\lambda)} \, \mathrm{d}t = 1 \tag{10}$$

Concrete example

We already have all the ingredients for this likelihood

$$\begin{aligned}\mathbb{P}(D|H_1) &= \frac{1}{\sigma}\int_{\lambda^-}^{\lambda^+}\prod_{i=1}^{N}\left[\frac{\exp(-t_i/\lambda)}{Z(\lambda)}\right]\mathrm{d}\lambda \\ &= \frac{1}{\sigma}\int_{\lambda^-}^{\lambda^+}\left[\frac{\exp(-\sum_{i=1}^{N} t_i/\lambda)}{Z^N(\lambda)}\right]\mathrm{d}\lambda \end{aligned} \tag{11}$$

`Coding tips: numerical integration, the log-sum-exp trick`

The other hypothesis

Consider a more complex hypothesis:

$$H_2: \text{ the bacteria population is a mixture with two waiting times } \lambda_1 \text{ and } \lambda_2 \tag{12}$$

Suppose the mixing ratio $\eta : 1 - \eta$ is known in H_2.

The other hypothesis

For each data point, its likelihood is simply the mixture between the two truncated exponential distributions:

$$\mathbb{P}(t|\lambda_1, \lambda_2, H_2) = \eta \frac{\exp(-t/\lambda_1)}{Z(\lambda_1)} + (1-\eta)\frac{\exp(-t/\lambda_2)}{Z(\lambda_2)} \tag{13}$$

The other hypothesis

Then the full likelihood is

$$\mathbb{P}(D|\lambda_1, \lambda_2, H_2) = \prod_{i=1}^{N} \left[\eta \frac{\exp(-t_i/\lambda_1)}{Z(\lambda_1)} + (1-\eta) \frac{\exp(-t_i/\lambda_2)}{Z(\lambda_2)} \right] \tag{13}$$

The other hypothesis

Then the full likelihood is

$$\mathbb{P}(D|\lambda_1, \lambda_2, H_2) = \prod_{i=1}^{N} \left[\eta \frac{\exp(-t_i/\lambda_1)}{Z(\lambda_1)} + (1-\eta) \frac{\exp(-t_i/\lambda_2)}{Z(\lambda_2)} \right] \tag{13}$$

So the probability of data given H_2 is

$$\mathbb{P}(D|H_2) = \frac{1}{\sigma_1 \sigma_2} \int_{\lambda_1^-}^{\lambda_1^+} \int_{\lambda_2^-}^{\lambda_2^+} \prod_{i=1}^{N} \left[\eta \frac{\exp(-t_i/\lambda_1)}{Z(\lambda_1)} + (1-\eta) \frac{\exp(-t_i/\lambda_2)}{Z(\lambda_2)} \right] \mathrm{d}\lambda_1 \, \mathrm{d}\lambda_2 \tag{14}$$

Bayes factor:

$$\frac{\mathbb{P}(D|H_1)}{\mathbb{P}(D|H_2)} = \frac{\sigma_1\sigma_2}{\sigma} \frac{\displaystyle\int_{\lambda^-}^{\lambda^+} \prod_{i=1}^{N} \left[\frac{\exp(-t_i/\lambda)}{Z(\lambda)}\right] \mathrm{d}\lambda}{\displaystyle\int_{\lambda_1^-}^{\lambda_1^+} \int_{\lambda_2^-}^{\lambda_2^+} \prod_{i=1}^{N} \left[\eta \frac{\exp(-t_i/\lambda_1)}{Z(\lambda_1)} + (1-\eta) \frac{\exp(-t_i/\lambda_2)}{Z(\lambda_2)}\right] \mathrm{d}\lambda_1 \, \mathrm{d}\lambda_2} \tag{15}$$

```
import numpy as np
import matplotlib.pyplot as plt
%pylab inline
```

```
Populating the interactive namespace from numpy and matplotlib
```

Probabilities can be very small numbers

Calculating the probability of some data given a model, usually requires multiplying many different probably terms

$$P(D \mid M) = \prod_{i=1}^{N} P(d_i \mid M).$$

This quantity easily underflows, which means that you are trying to calculate something smaller than the precision of your computer.

If you had a data set D, in which each individual datum had probability of the order of 0.001 (which is not such small quantity), then the probability of all the data is

$$P(D) = \left(10^{-3}\right)^N = 10^{-3\times N}$$

```
## try it yourself!

10**(-323), 10**(-324)
```

```
(1e-323, 0.0)
```

The smallest double-precision floting point number,

$$k_{min} \approx 10^{-323}$$

which means that at $N = 100$ data points,

$$P(D; N = 100) = 10^{-300}$$

Has become so small that you would not be able to distinguish having a data set with $N = 100$ measurements from having another with $N = 200$ points. In other words when we go and try to calculate said probabiliy we are going to end up with 0 instead of what we know it to actually be: a really, really small number.

How do we get around this?

Working in log space

Working with the log of the probability can be very useful to avoid underflow problems. Because it expands the dynamic range of the probability effectively giving you access to those small numbers that you couldn't calculate before. Also, because the logarithm of the probability is monotonically increasing, maximizing the probability and the logarithm of the probability is the same task.

But working in log-space can be helpful in many ways, let's go back to our example:

$$\log P(D \mid M) = \log \prod_{i=1}^{N} P(d_i \mid M) = \sum_{i=1}^{N} \log P(d_i \mid M).$$

If you are having trouble seeing why this is the case remember that it is a property of logarithms that $\log(a_1 a_2) = \log(a_1) + \log(a_2)$. Convince yourself that this extends to N elements, $\log(a_1 a_2 \ldots a_N) = \log(a_1) + \log(a_2) + \ldots + \log(a_N)$ when you are sold, it is trivial to see that $\log(a_1 a_2 \ldots a_N) = \sum_{i=1}^{N} \log(a_i)$.

If we go back to the previous example:

$$\log P(D; N = 100) = -300 * \log(10) = -690.776$$

but

$$\log P(D; N = 200) = -600 * \log(10) = -1382.551$$

thus, the two cases are easily distinguishable in log space.

$$\log P(D; N = 200) - \log P(D; N = 100) = -691.775$$

or

$$P(D; N = 200) = e^{-691.775} P(D; N = 100)$$

```
np.log(np.exp(1000)+np.exp(999))
```

```
/var/folders/y7/ndhfk28s5_74nw_nw4pt66_w0000gq/T/ipykernel_59072/913420253.py:1: Runti
  np.log(np.exp(1000)+np.exp(999))
```

```
inf
```

Sometimes you describe your system with a mixture of probability distributions. For instance like mixture of Gaussian distributions, or like in the case of our homework this week a mixture of exponential distributions.

Now we work in log space so you have, so you end up having to do the following calculation $\log(e^a + e^b)$

If a and b are large and negative (as it is the case when they represent log probabilities), then that calculation cannot be directly. For instance, a naive calculation of $log(e^{-1000} + e^{-999})$ will give you -infinity. And if we tried with positive number, for instance $\log(e^{1000} + e^{999})$ will give you +infinity.

The way to do this calculation robustly is as follows, assume $a > b$, then

$$\begin{aligned}
\log(e^a + e^b) &= \log(\frac{e^a}{e^a}(e^a + e^b)) \\
\log(e^a + e^b) &= \log\big(e^a(1 + e^{b-a})\big) \\
\log(e^a + e^b) &= \log(e^a) + \log\big(1 + e^{b-a}\big) \\
\log(e^a + e^b) &= a + \log\big(1 + e^{b-a}\big)
\end{aligned}$$

since $b - a < 0$, the exponential $e^{b-a} \leq 1$ never becomes a large number, and the calculation is robust.

Then you will calculate

$$\log(e^{-1000} + e^{-999}) = -999 + \log(1 + e^{-1}) = -999 + 0.31 = -998.69,$$

and

$$\log(e^{1000} + e^{999}) = 1000 + \log(1 + e^{-1}) = 1000 + 0.31 = 1000.31,$$

The log-sum-exp trick can be generalized for an arbitrary number of terms

$$n$$

$$\log(e^{a_1} + \ldots + e^{a_n}) = a_{max} + \log\left(1 + \sum_{i=1}^{n} e^{(a_i - a_{max})}\right),$$

where a_{max} is the maximum value of the set $\{a_1, \ldots, a_n\}$.

```
from calendar import c
from cmath import log
from scipy.special import logsumexp

#a, b, c

#e^a, e^b, e^c

#e^a + e^b + e^c

#log(e^a + e^b + e^c)

?logsumexp
```

```
Signature: logsumexp(a, axis=None, b=None, keepdims=False, return_sign=False)
Docstring:
Compute the log of the sum of exponentials of input elements.

Parameters
----------
a : array_like
    Input array.
axis : None or int or tuple of ints, optional
    Axis or axes over which the sum is taken. By default `axis` is None,
    and all elements are summed.

    .. versionadded:: 0.11.0
keepdims : bool, optional
    If this is set to True, the axes which are reduced are left in the
    result as dimensions with size one. With this option, the result
    will broadcast correctly against the original array.

    .. versionadded:: 0.15.0
b : array-like, optional
    Scaling factor for exp(`a`) must be of the same shape as `a` or
    broadcastable to `a`. These values may be negative in order to
    implement subtraction.

    .. versionadded:: 0.12.0
return_sign : bool, optional
    If this is set to True, the result will be a pair containing sign
    information; if False, results that are negative will be returned
    as NaN. Default is False (no sign information).

    .. versionadded:: 0.16.0

Returns
-------
res : ndarray
    The result, ``np.log(np.sum(np.exp(a)))`` calculated in a numerically
    more stable way. If `b` is given then ``np.log(np.sum(b*np.exp(a)))``
    is returned.
sgn : ndarray
    If return_sign is True, this will be an array of floating-point
    numbers matching res and +1, 0, or -1 depending on the sign
    of the result. If False, only one result is returned.

See Also
--------
numpy.logaddexp, numpy.logaddexp2

Notes
-----
NumPy has a logaddexp function which is very similar to `logsumexp`, but
only handles two arguments. `logaddexp.reduce` is similar to this
function, but may be less stable.

Examples
--------
>>> from scipy.special import logsumexp
>>> a = np.arange(10)
>>> np.log(np.sum(np.exp(a)))
9.4586297444267107
>>> logsumexp(a)
9.4586297444267107

With weights

>>> a = np.arange(10)
>>> b = np.arange(10, 0, -1)
>>> logsumexp(a, b=b)
9.9170178533034665
>>> np.log(np.sum(b*np.exp(a)))
```

```
9.9170178533034647

Returning a sign flag

>>> logsumexp([1,2],b=[1,-1],return_sign=True)
(1.5413248546129181, -1.0)

Notice that `logsumexp` does not directly support masked arrays. To use it
on a masked array, convert the mask into zero weights:

>>> a = np.ma.array([np.log(2), 2, np.log(3)],
...                  mask=[False, True, False])
>>> b = (~a.mask).astype(int)
>>> logsumexp(a.data, b=b), np.log(5)
1.6094379124341005, 1.6094379124341005
File:      ~/opt/anaconda3/lib/python3.9/site-packages/scipy/special/_logsumexp.py
Type:      function
```

```
import numpy as np
np.logaddexp?
```

How to integrate?

We want to find the Area under the curves!

```
import numpy as np
import matplotlib.pyplot as plt
```

```
xs = np.linspace(0,1,int(1e7)) # define bins
dx = xs[1]-xs[0] # width of a bin
```

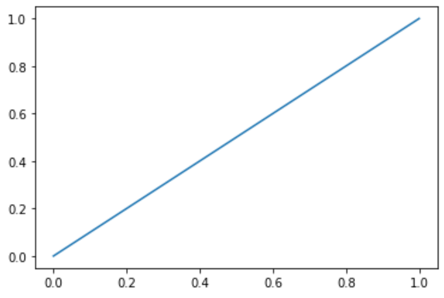
```
xs #see what our bin list looks like
```

```
array([0.0000000e+00, 1.0000001e-07, 2.0000002e-07, ..., 9.9999980e-01,
       9.9999990e-01, 1.0000000e+00])
```

```
def g(x): # a function to integrate
    return x
```

```
plt.step(xs,g(xs))
```

```
[<matplotlib.lines.Line2D at 0x7f8dd928d5d0>]
```



```
%%time
bin_vals = []
for x in xs: # For every value in our list of bins get the area of the particular bin
    bin_vals.append(x*dx)
```

```
print(np.sum(bin_vals)) # sum up all the areas.
```

```
0.5000000500000026
CPU times: user 1.87 s, sys: 162 ms, total: 2.03 s
Wall time: 2.04 s
```

```
%%time
np.sum(g(xs)*dx) # numpy broadcasting Vector, we do the operation to ever entry on the
```

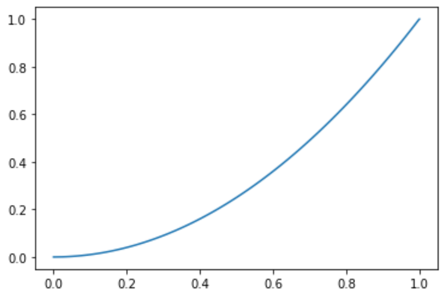
```
CPU times: user 6.72 ms, sys: 2.22 ms, total: 8.94 ms
Wall time: 6.97 ms
```

```
0.5000000500000026
```

```
def g(x): #Another function to integrate.
    return x**2
```

```
plt.step(xs,g(xs))
```

```
[<matplotlib.lines.Line2D at 0x7fd788cb1310>]
```



```
%%time
bin_vals = []
for x in xs:  # For every value in our list of bins get the area of the particular bin
    bin_vals.append(x**2 * dx)

np.sum(bin_vals) # sum up all the areas.
```

```
CPU times: user 3.92 s, sys: 115 ms, total: 4.03 s
Wall time: 4.04 s
```

```
0.3333333833333404
```

```
%%time
np.sum(xs**2*dx) # numpy broadcasting Vector, we do the operation to ever entry on the
```

```
CPU times: user 10.6 ms, sys: 5.4 ms, total: 16 ms
Wall time: 15.3 ms
```

```
0.3333333833333404
```

But what about higher dimensional integrals?

$$\int\int(x^2 - y^2)dxdy$$

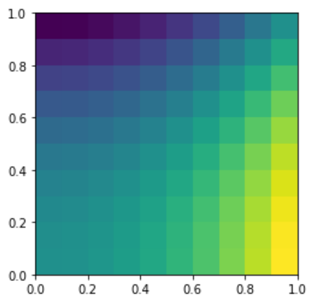
We don't want the area under the curve, we want the volume under the surface!

```
xs = np.linspace(0,1,10) # define bins
ys = np.linspace(0,1,10) # define bins
dx = xs[1]- xs[0] # width of a bin
dy = ys[1]- ys[0] # width of a bin
```

```
%%time
int_value = 0
bin_vals_x = []
for x in xs: # For every value in our list of bins in one axis
    bin_vals_y = []
    for y in ys: # For every value in our list of bins in  the other axis
        bin_vals_y.append((x**2-y**2)*dx*dy) #updating lists
        int_value += (x**2-y**2)*dx*dy # Summing the entry at every step
    bin_vals_x.append(bin_vals_y)
bin_vals_x = np.array(bin_vals_x) #create surface array
print(int_value) #value of the integral
```

```
-1.5178830414797062e-17
CPU times: user 244 µs, sys: 76 µs, total: 320 µs
Wall time: 315 µs
```

```
plt.imshow(bin_vals_x.T, origin='lower', extent=[0,1,0,1])
plt.show()
```



```
xs[:,np.newaxis].shape
```

```
(10, 1)
```

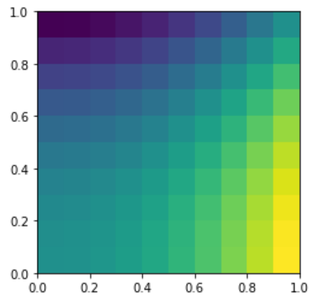
```
%%time
surf = xs[:,np.newaxis]**2 - ys**2 # Numpy broadcasting method of integrating with two
```

```
CPU times: user 43 µs, sys: 6 µs, total: 49 µs
Wall time: 50.1 µs
```

Try is yourself:

- What does the silcing `[:, np.newaxis]` do to the numpy array being sliced?
- How does the operation propagate?

```
plt.imshow(surf.T, origin='lower', extent=[0,1,0,1])
plt.show()
```



Don't know where to start with the homework! Or getting confused with the `log` computations?

Try starting here

```
def Z(lam, tmin, tmax):
  return lam * (np.exp(-tmin/lam) - np.exp(-tmax/lam))

## TODO FOR YOU, FOR HOMEWORK
## YOU'LL REALLY WANT TO USE LOGS TO COMPUTE THINGS
def logZ(lam, tmin, tmax):
    return -1

# check if similar:
np.isclose(Z(3, 0.05, 80), np.exp(logZ(3, 0.05, 80)))
```