PyQB

Monga

Programming in Python[1]

Mattia Monga

Dip. di Informatica
Università degli Studi di Milano, Italia
mattia.monga@unimi.it

Academic year 2020/21, II semester

PyQB

Monga

Lecture XVIII: Probabilistic programming

How science works

PyQB

Monga

Describing one single "scientific method" is problematic, but a schema many will accept is:

1. Imagine a hypothesis
2. Design (mathematical/convenient) **models** consistent with the hypothesis
3. Collect experimental data
4. Discuss the fitness of data given the models

It is worth noting that the falsification of models is not *automatically* a rejection of hypotheses (and, more obviously, neither a validation).

The role of Bayes Theorem

PyQB

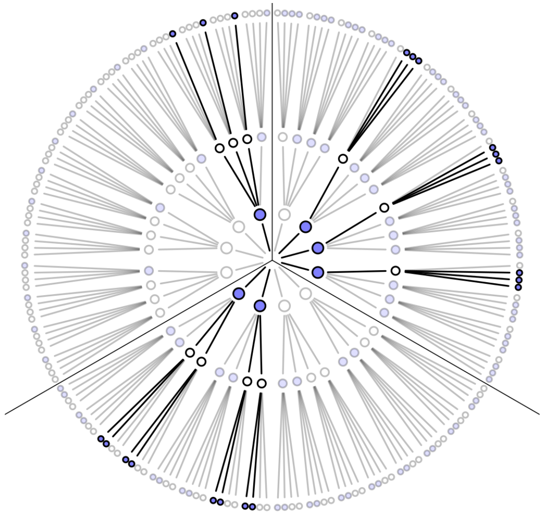
Monga

In this discussion, a useful relationship between data and models is Bayes Theorem.
$P(M, D) = P(M|D) \cdot P(D) = P(D|M) \cdot P(M)$
Therefore:
$P(M|D) = \frac{P(D|M) \cdot P(M)}{P(D)}$
The plausibility of the model given some observed data, is proportional to the number of ways data can be *produced* by the model and the prior plausibility of the model itself.

Simple example

- Model: a bag with 4 balls in 2 colors
- Observed: BWB
- Which is the plausibility of BBBB, BBBW, BBWW, BWWW, WWWW?

Bayes Theorem is the formalization of this **counting**



Picture from: R. McElreath, Statistical Rethinking

A computational approach

This Bayesian strategy is (conceptually) easy to transform in a computational process.

1. Code the models
2. Run the models
3. Compute the plausibility of the models based on observed data

Classical binomial example

- Which is the proportion p of water covering Earth? The models are indexed by the float $0 < p < 1$
- Given p, the probability of observing some W,L in a series of **independent random observations** is: $P(W, L|p) = \frac{(W+L)!}{W! \cdot L!} p^W \cdot (1-p)^L$ (binomial distribution).
- Do we have an initial (prior) idea?
- Make observations, apply Bayes, update prior!

A conventional way of expressing the model

$$
\begin{aligned}
W &\sim \mathit{Binomial}(W+L, p) \\
p &\sim \mathit{Uniform}(0, 1)
\end{aligned}
$$

Probabilistic programming is systematic way of coding this kind of models, combining predefined statistical distributions and Monte Carlo methods for computing the posterior plausibility of parameters.

In principle you can do it by hand


```python
def dbinom(success: int, size: int, prob: float) -> float:

    fail = size - success
    return np.math.factorial(size)/(np.math.factorial(success)*np.math.factorial(fail))*p
    ↪ rob**success*(1-prob)**(fail)
```

```python
W, L = 7, 3
p_grid = np.linspace(start=0, stop=1, num=20)
prior = np.array([1] * 20)

likelihood = dbinom(W, n=W+L, p=p_grid)

unstd_posterior = likelihood * prior

posterior = unstd_posterior / unstd_posterior.sum()
```

Unfeasible with many variables!

PyMC


```python
import pymc as pm

W, L = 7, 3
earth = pm.Model()
with earth:
    p = pm.Uniform("p", 0, 1)  # uniform prior
    w = pm.Binomial("w", n=W+L, p=p, observed=W)
    posterior =  pm.sample(2000)

posterior['p']
```