BAYESIAN STATISTICS

ETHAN LEVIEN

Contents

1. Parameters as random variables

Let's revisit the very basic statistic model

$$Y \sim \text{Bernoulli}(q). \tag{1}$$

In our previous treatment of the statistical inference for this problem, we introduced the MLE which was obtained by maximizing the probability distribution $\mathbb{P}(Y|q)$. In classical statistics q is a fixed number, NOT a random variable. Therefore, although we may not know the value of q beforehand, it is assumed that there is some specific value of q which is correct. The standard errors represent how much q will vary between different replicates of the data (e.g. different subsamples of a population which are collected for a survey). There is a different way to think about uncertainty in parameters though: We can think of them as being random variables themselves. This has two advantages: First, it allows us to more easily quantify uncertainty in our parameters. Second, it allows to incorporate prior, sometimes subjective, knowledge about the parameters into our inference.

In some sense, treating parameters as random variables is a simple mater of exapanding our model, as the following example makes clear.

Example 1. *Consider the model*

$$Y \sim \text{Bernoulli}(q) \tag{2}$$

$$q \sim \text{Beta}(2,1) \tag{3}$$

If we flip the coin twice, what the chance we get two heads in a row?

When we write down models in this way, with the probability distribution for the parameters specified as well as the other variables, we call the distribution of the parameters (a Beta distribution in the case) our priors. Where do priors come from?

Example 2. *Suppose that you are* 95 *confident that a coin is fair, what are our priors?*

Exercise 1. *Suppose we conduct a survey of Dartmouth students asking the question: where you alive in 1995? We suspect that the answer will be no*

1.1. Priors for a regression model.

Example 3. *Consider the model, with priors,*

$$Y \sim \text{Normal}(\alpha + \beta x, 0.4) \tag{4}$$

$$\alpha \sim \text{Normal}(0,2) \tag{5}$$

$$\beta \sim \text{Normal}(1,2) \tag{6}$$

What is the chance that $Y > 0.3$.

In the above examples, we fixed σ, but a it is reasonable to place priors on this as well. The question is: How do we do this in a way that ensures $\sigma > 0$?

Exercise 2. *Consider the election example, suppose we know (e.g. from polling data that). What does this tell use about* α*?*

Date: April 2022.

2. Bayesian inference

The question we know address is: How do we incorporate priors into our statistical inference? Bayes' theorem tell us exactly how to do this (in theory). In general, if D is a our data and θ is our parameter(s), we can think of there being a joint distribution $\mathbb{P}(D,\theta)$, and hence

$$P(D|\theta)P(\theta) = P(\theta|D)\mathbb{P}(D) \tag{7}$$

Rearraning this yields

$$P(\theta|D) = \frac{P(D|\theta)P(\theta)}{P(D)} \tag{8}$$

Exercise 3. *Can you explain why Bayes theorem holds for both probability distributions and probability densities?*

The distributions appearing in Equation (8) have the following names and interpretations:

- $P(\theta|D)$ is the posterior.
- $P(D|\theta)$ is the likelihood
- $P(\theta)$ is the prior distribution
- $P(D)$ is the evidence. It represents the chance we observe the data *unconditional* on the parameters. This can be obtained by marginalizing over the priors.

While in classical statistics, the objective is to determine (estimate) a parameter θ, in Bayesian statistics our objective is to compute the posteior distribution. Once we have this, we can obtain so-called **point estimates**, for example, by taking the average of θ, most of the computational work that needs to be done surrounds the computation of the posterior thogh.

2.1. MCMC.

In some cases, we can obtain analytical formula for the posterior distribution, but this is extremely rare. In reality, the best we can do is draw samples from the posterior distribution. The algorithms which achieve this are called Markov Chain Monte Carlo algorithms. The challange in Monte Carlo simulations is deal with the fact that $p(D)$ is unknown in Equation (8). If $p(D)$ was known, we would have a formula for $p(\theta|D)$, since the likelihood and the prior are both detemined by our model. The trick is that $p(D)$ doesn't depend on the parameters, and therefore, and therefore all distributions for which

$$p(\theta|D) \propto U(\theta) = p(D|\theta)p(\theta) \tag{9}$$

We want to generate samples in way that ensures the ratio between the frequancies of θ_1 and θ_2 are $U(\theta_1)/U(\theta_2)$. Let's say we have a sample θ_1. Then we take a step $\theta_2 = \theta + \Delta$. In order decide if we accept this new sample, we look at the ratio $U(\theta_1)/U(\theta_2)$. After many steps of this algorithm, we should have a set of samples of θ which is close to the actual.

Example 4. *Write a function which carries out Markov Chain Monte Carlo simulations*

2.2. Probabilistic programming in pymc3.

It turns out the way of doing MCMC simulations described above is not very practical for realy scienfitifc problems and more advanced methods require much more mathematics and are challanign to implement. Fortunately there are many esablished packages for performing these simulations. The one we will work with is called Pymc3. In Pymc3 we can define a model in a way similar to how we usually write them down.

```
> # now we build a model for the coin
> # we start by simply defining a model using the Model() constructor
> coin_flipping = Model()
>
> # right now coin_flipping is just an "empty" model
> # it has no variables or paramaters.
> # model specifications in pymc3 are wrapped in a with-statement
> # don't overthink this statement, basically within with is where we tell pymc what to do
> with coin_flipping: #
>     # first we define out priors
>     # the model has one parameter p (the probability to get heads)
>     q = Uniform("q", lower=0, upper=1)
>
>     # now we tell pymc3 what the probability distribution of our data is
>     # this is generally what we think of as a our "model" or likelihood
>     y = Bernoulli("y", p=q, observed=flips)
```

Now that we've given pymc3 everything it needs to know about our statistical model.

NOTE: hen we specify distributions in pymc we do not use the "np.random" functions. Those generate samples from a random distribution, while here we are just telling pymc3 what the distributions of parameters and data are.

The next step is to sample from the Posterior distribution using Markov chain Monte Carlo simulations

```
>   # anything that we do with the model happens inside the with statement
>   with coin_flipping:
>     # this will generate samples from the posterior distribution
>     # it may take a minute or so
>     trace = sample(4000,cores =3)
```

The structure of the pymc inference code is:

(1) Tell pymc you are making a model by writing
    name$_of_model = Model()$
(2) In the indented code below a with statement tell pymc what you the distributions of your parameters and variables are. These can either be parameters or variables in our model we have data for.
    This looks like:
    ```
    > with name_of_model:
    >   # this defines out prior
    >   name_of_parameter = NameOfDistribution("name of variable",parameters)
    >   .
    >   .
    >   .
    >   # this defines our likelihood (our model)
    >   name_of_variable = NameOfDistribution("name of variable",parameters,observed = our dat
    ```
    Generally the distribution functions that specify what distributions for pymc3 to use can take arrays for the parameters and observed data. For example, we can make the mean of the normal distribution a vector and the ith entry will be used as the mean for the ith entry of y. In particuar, for a regression model we can write:
    ```
    > y = Normal("y",mu = a*x_data + b,sigma = sigma,observed = y_data)
    ```
(3) In the indented code below a with statement tell pymc to generate samples from the posterior distribution. The pymc function for doing this is called sample.
    ```
    > with name_of_model:
    >   trace = sample(number of samples)
    ```

Some things to remember:

- In general, don't do anything inside the with statement besides these things!
- do things in this order
- if you change something in your model or are making a new model do the first step agian

Trace is a python object which saves all the information about the posterior distribution. perhaps most importantly, trace stores a list of values of p. each number in this list is a random sample from the posterior distribution

```
> q_samples = trace["q"]
```

3. Bayesian p-values