STATISTICAL MODELS AND SIMULATION

ETHAN LEVIEN

Contents

1. Statistical models

A central concept to this course is that of a **model**. Before doing any mathematics, let's reflect on the notion of a model in the abstract.

Exercise 1. *What does it mean to model something?*

Broadly speaking, models are simplified representations of the world[1] There are many ways to represent models, but in science (and life), we often use **mathematical models**. For example, you might be familiar with Newton's equation:

$$F = ma \tag{1}$$

which related the force (F) acceleration (a) and mass (m) of a particle. This a mathematical model of the motion of a (non-relativistic) particle in a force field. While it isn't always true, it holds for such a wide range of applications that we call it a *fundamental* law of nature, and it turns out to be extremely powerful. We can combine Newton's equation with other fundamental lows build models of more complicated systems involving many particles. For example, to build a model of planetary motion, we can combine Newton's equation with his other law of gravity, which states that the gravitation force between two objects of masses m_1 and m_2 a distance r from each other is

$$F = G\frac{m_1 m_2}{r^2} \tag{2}$$

where G is a constant. We refer to these types of models – that is, those which are built upon fundamental laws of nature – as **mechanistic** models.

In statistics, we are often interested in problems where nothing remotely close to a fundamental laws exist. Instead, they are based directly on observations (data) or our intuitions. In economics, for example, models are built in numerous assumptions about the human nature, many of which may seem obvious or intuitive. However, our intuition about economics is itself shaped by the particular economic system we experience, and in reality these assumptions regularly break down as economic conditions change. We will call these **phenomenological models**. Such models still be quite useful provided we are aware of their limitations. Of course, there is not sharp distinction between mechanistic and phenomenological models, but the distinction is helpful nonetheless.

Often our models cannot make exact predictions about the value of a variable (this is true in both mechanistic and phenomenological models, but especially the latter). Instead, they only tell us the probability that a variable has a certain value, or falls within a range of values. For example, if we were to construct a model of

Date: April 2022.

[1]Model is sometimes used interchangeably with **theory**, although I usually think of models as being smaller in scope.

the (y) of all the pine trees in New Hampshire as a function of their age (x), we might begin by searching for a function f such that

$$y = f(x) \tag{3}$$

However, if we take a **sample** of the population, meaning we go out and measure the height of some trees for which we know the age, we will quickly find that trees of the same age can have different heights. This variation will be as result of many variables which are not in our model, e.g. the surrounding, the specific subspecies of pine, genetic variation within a subspecies to name just a few. We could construct a more sophisticated model which includes, say, the DNA sequence of the tree (g):

$$y = f(x, g). \tag{4}$$

This model would presumably have less variation. That is, if we collect a sample of trees for which we know the height and DNA sequence, we would find that the variation between trees with the same DNA and height is less than the variation between trees with only the same height. Yet variation will remain unless we include all the variables which effect tree height. This is of course impossible, as there are million, perhaps billions of such variables and their effects are far beyond our understanding of biology. Moreover, it is not so useful to include these variables, since we can actually measure most of them and many of them will have small effects. One approach to dealing with these "hidden" sources of variation is to define a **statistical model**, where y is not the same for each tree of the same height, but is instead a **random variable**. On type of a statistical model is a regression model

$$y = f(x) + \epsilon \tag{5}$$

where ϵ is a random variable (usually one that is zero, on average).

2. Random variables and distributions

The rigorous mathematical theory for random variables is very useful, but requires certain machinery which is beyond the scope of these notes. Fortunately, we go a long way without such formalism. For our purposes, a random variable can be understood as a variable which we cannot predict prior to an observation, regardless of how much information we have. We can define the space of **outcomes** as all the possible values that a random variable may take on. The outcomes for the roll of a dice are $1, 2, \ldots, 6$ for the dice, or positive numbers for the height of a tree. Usually the outcomes are numbers, even if we use a number to represent a non-numerical quantity (e.g. someone's gender). In probability theory, one distinguishes between outcomes and **events** – the latter are subsets of outcomes. For example, we might refer to the event that the roll of a die is grater than 2. It's good to be aware of these definitions, but you don't need to memorize them.

We can describe a characterize a random variable using a **probability distribution**, which maps a set of possible events to real numbers between 0 and 1. For example, suppose we ask a random student in the college whether they were born in the US. A probability distribution $P(Y)$ which *models* their answer is the **Bernoulli distribution**,

$$P(Y) = \begin{cases} q & Y = \text{YES} \\ 1 - q & Y = \text{NO} \end{cases} \tag{6}$$

where q is the fraction of students in the college who were born in the US. We say that q is a **parameter** in our model because regardless of it's value our model is still a Bernoulli distribution. It is very important that the sum of $P(Y)$ over all possible outcomes is 1 – this is simply saying that we are certain one of the outcomes will happen.

The Bernoulli distribution is our default model for any variable that can take two possible outcomes, usually abstracted as 0 or 1. In order to state that a Bernoulli distribution is a model for some random variable Y, we write

$$Y \sim \text{Bernoulli}(q). \tag{7}$$

We might also say "Y *follows* as Bernoulli distribution" or "Y is a Bernoulli random variable". More generally, we say that a variable in a model follows a given distribution by writing

$$\text{Variable} \sim \text{Distribution}(\text{parameters}). \tag{8}$$

We will sometimes use θ to denote the parameters.

Turning back to the example of our survey, let's suppose we don't have information about every students in the college. Rather, a survey of five students from this class is conducted, finding 4 yeses and 1 no. It seems like our best estimate of

q should be the fraction of $4/5$. This is an example of **statistical inference**. More generally, we use statistical inference to make predictions about things we don't observe based one what we do observe (data).

3. Python as a tool for statistical modeling

When we generate samples using a computer we call them **simulations**. We will use python to perform simulations, and it is therefore important to have a basic understanding of the python language. It is assumed that you will go through the separate python tutorial notebook. For convenience, here is a very brief summary of some things you will need to do in Python. First, we will assign variables and perform arithmetic. These operations are pretty intuitive:

```
> x = 1
> y = 2
> z = x + y
> print(z)
> z = x*y
> print(z)
```

We will work heavily with vectors, or lists of numbers in this course.

```
> a = [1,2,3] # make empty list
> a.append(5) # add 5 to the end of the list
> print(a) # print the list
> print(a[2])        # index the list
> print(a[0:2])
> print(a[a>2]) # obtain a subset of the list
> # easy way to make an array
> a = range(5)
> print(a[0])
```

We will work with a package called numpy, which provides a wrapper for the python list affording us some additional functionality.

```
> import numpy as np # import the numpy package
> a = np.zeros(5)
> a = np.array([0,0,0,0,0])
> # advantage of numpy arrays:
> a = np.array([1,2,3])
> b = np.array([5,3,2])
> a + b
> # get the length of an array
> len(a)
> # generate array
> a = np.linspace(0,1,100)
> print(a)
```

Loops allow us to repeat an operation, or carry out an operation over some range of values

```
> for k in range(5):
>         print(k)
```

The same thing can be achieved with a while loop

```
> k = 0
> while k<5:
>         print(k)
>         k = k+1
```

In many instances, it is helpful to put code we will reuse in a function.

```
> def test_func(z)
>         k = 0
>         while k<z:
>                 print(k)
>                 k = k+1
>          return k
> a = test_func(k)
```

Exercise 2. *Consider the following code:*

```
> for i in range(5):
>   for j in range(i)
>     print(i)
```

```
>     print(",")
>   println()
```

Based on the output of this code, what do you think `println()` *does? Check the documentation to confirm you answer. Check the documentation to confirm your answer. Then, modify this code and put it in a function* `print_triangle(n)` *which prints*

```
> 1,1
> 1,2,2
> 1,2,3,3
> .
> .
> .
> 1,2,3,...,n,n
```

Solution:
Making only a few changes to the existing code and putting it in a function

```
> def print_triangle(n)
>     for i in range(n):
>       for j in range(i)
>         print(i)
>         print(",")
>       print(i)
>       println()
```

3.1. **Simulations.** Here, we will focus on tools relevant for statistics. In Python, we can simulate random variables using the `numpy` library:

```
> import numpy as np
> q = 0.5
> y = np.random.choice(range(2),p=[q,1-q])
```

We can generate multiple samples using a for loop

```
> n_samples = 100
> y = np.zeros(n_samples) # makes an empty list (i.e. array) of n_samples zeros.
> for k in np.range(n_samples):
>   y[k] = np.random.choice(range(2),p=[q,1-q])
```

A simpler way of doing this is

```
> y = np.random.choice(range(2),n_samples,p=[q,1-q])
```

The more general form of this command is

```
> y = np.random.choice(range(k),n_samples,p=[q_1,q_2,...,q_k])
```

where $q_1 + \cdots q_k = 1$. This will generate a sample from

Exercise 3. *Write down the probability distribution for a the value rolled by a 6 sided die and use the random choice function in python generate simulated rolls of your dice. Then, write down a probability distribution for a* 6 *sided die with the one face replaced with a two.*

Solution:
For the fair die we have

```
> y = np.random.choice(range(6),n_samples,p=[1/6,1/6,1/6,1/6,1/6,1/6])
```

For the other one we have

```
> y = np.random.choice(range(6),n_samples,p=[0,2/6,1/6,1/6,1/6,1/6])
```

We can also generate simulations of more complex random variables using simple ones. For, example:

Example 1. *Write a function which simulates flipping a fair coin until we get* 2 *heads in a row.*

Solution:

```
> def flip_until_two():
>   num_heads = 0
>   total_flips = 0
>   while num_heads <2:
>     y = np.random.choice([0,1])
>     if y == 0:
```

```
>       num_heads = 0
>     else:
>       num_heads = num_heads + 1
>     total_flips = total_flips + 1
>   return total_flips
```

Exercise 4. *By changing the code above, write a function that rolls a dice until we get two ones in a row.*

Solution:
We need to change (1) the distribution we sample y from and (2) the condition to stop.

```
> def dice_until_two():
>   num_ones = 0
>   total_rolls = 0
>   while num_ones <2:
>     y = np.random.choice([1,2,3,4,5,6])
>     if y == 0:
>       num_ones = 0
>     else:
>       num_ones = num_ones + 1
>     total_rolls = total_rolls + 1
>   return total_rolls
```

3.2. **Visualization.** An important tool for visualizing samples is a histogram

```
> plt.hist(samples,100,density=true)
```

The histogram shows us the frequency of different outcomes.

3.3. **Monte Carlo.** Often, we run many simulations of a model in order to say something about the distribution without performing any analytical calculations. We call these **Monte Carlo** simulations.

Example 2. *Suppose we flip a coin until we get 2 heads in a row, as we implemented above, and let J denote the number of flips. Use Monte Carlo simulations the chance that $J > 5$. Also plot a histogram.*

Solution:
We will make a loop which performs the coin flipping many times and save the number of flips.

```
> n_monte = 1000
> Js = np.zeros(n_monte)
> for k in range(n_monte):
>   Js[k] = flip_until_two()
> len(Js[Js>5])/len(Js)
> plt.hist(Js)
```

3.4. **Means and variances.** There are ways in which we summarize attributes of random variables. If we have many samples $Y_1, Y_2, \ldots, Y_n$ of a random variable (e.g. answers to a survey question), the **sample mean** is defined as

$$\bar{y} = \frac{1}{n}\sum_i y_i \tag{9}$$

Often it is useful to quantify the deviations from the mean. Suppose y_i can take on outcomes $y_1, \ldots, y_m$. If n is large, then the fraction of samples for which $Y_1 = y_1$ will be $P(Y_1 = y_1)$.

$$\bar{y} \approx \frac{1}{n}\sum_{\text{outcomes}} y_i n_i = \sum y_i P(Y_i = y_i) \tag{10}$$

The expression on the right is the definition of expected value, often denoted $\mathbb{E}[Y]$.

For this, we have the **sample standard deviation**

$$\sigma = \sqrt{\frac{1}{n-1}\sum_i (y_i - \bar{y})^2}. \tag{11}$$

We will see why this makes sense as a measure of how spread our a distribution is later on when we talk about inference. For large n, this converges to the square root of the variance

$$\mathrm{Var}(Y) = \sum (y_i - \mathbb{E}[Y])^2 P(Y_i = y_i). \tag{12}$$

Importantly, we can always calculate the mean and standard deviation of a sample, regardless of the distribution it has been drawn from. However, we need to be careful, as the results may not be so meaningful.

In python, functions for implementing the mean and standard deviation are follows:

```
> np.mean(y)
> np.std(y)
```

Example 3. *Verify that the formula for the mean and standard deviations with Monte Carlo simulations*

$$\mathbb{E}[Y] = q \tag{13}$$

Solution:
The following code will estimate the mean and standard deviations for various values of q:

```
> def mean_bern(q):
>   sample = np.random.choice([0,1],500,p=[1-q,q])
>   return np.mean(sample)
>
> # generates one hundred evenly spaced numbers between 0 and 1
> q_range = np.linspace(0,1,100)
> means = np.zeros(len(q_range))
> for j in range(len(q_range)):
>   means[j]= mean_std_bern(q_range[j])
```

Here is a plot of the mean compared to the prediction

```
> fig,ax = plt.subplots(figsize=(5,2))
> ax.plot(q_range,means)
> ax.plot(q_range,q_range,"k--")
```

Exercise 5. *Modify the code above to verify that*

$$\sqrt{\mathrm{Var}(Y)} = \sqrt{q(1-q)} \tag{14}$$

Solution:
We only need to change the mean to the standard deviation

```
> def std_bern(q):
>   sample = np.random.choice([0,1],500,p=[1-q,q])
>   return np.std(sample)
>
> # generates one hundred evenly spaced numbers between 0 and 1
> q_range = np.linspace(0,1,100)
> means = np.zeros(len(q_range))
> for j in range(len(q_range)):
>   stds[j]= std_bern(q_range[j])
```

4. Joint probabilities, Independence

We introduce, very briefly, the concepts of independence and conditioning. Just as we have considered single random variables, we can consider multiple random variables within the same model. Suppose we have two Bernoulli random variables Y_1 and Y_2 which model whether a person has mutations at two different genomes. In this case, we need a model of both variables together:

$$\mathbb{P}(Y_1, Y_2) = \begin{cases} q_{00} & \text{if } Y_1 = 0 \text{ and } Y_2 = 0 \\ q_{01} & \text{if } Y_1 = 0 \text{ and } Y_2 = 1 \\ q_{10} & \text{if } Y_1 = 1 \text{ and } Y_2 = 0 \\ q_{11} & \text{if } Y_1 = 1 \text{ and } Y_2 = 1 \end{cases} \tag{15}$$

Example 4. *Write python code to generate a sample of Y_1 and Y_2 using*

$$q_{00} = 1/8, \quad q_{01} = 1/8, \quad q_{10} = 1/4, \quad q_{11} = 1/2 \tag{16}$$

Solution:
There are 4 outcomes, so we can identify each with a number as follows:

$$(0,0) \to 1, \quad (0,1) \to 2, \quad (1,0) \to 3, \quad (1,1) \to 4 \tag{17}$$

We then use the random choice function

```
> np.random.choice([1,2,3,4],p=[1/8,1/8,1/4,1/2])
```

The probability distribution $P(Y_1, Y_2)$ tells us the probabilities for observing *both* variables together, e.g. observing a person with both mutations. It does not directly tell us the probabilities of observing e.g. someone with only one mutation. This can be obtained via marginalization; that is, summing over the other variable:

$$P(Y_1) = \sum_y P(Y_1, y) = P(Y_1, 0) + P(Y_1, 1) \tag{18}$$

where in the general the sum is taken over all possible outcomes for the second variable. $\mathbb{P}(Y_1)$ is defined similarly. Sometimes, we are interested in the value of a random variable given, or **conditioned on** another random variable. The probability of that $Y_1 = 1$ if we know $Y_2 = 0$, denoted $P(Y_1 = 1|Y_2 = 0)$ would be the chance that gene 1 has a mutation in a person if we know there is no mutation at gene 2. Said another way, we are looking at the chance of someone having a mutation at gene 1 among only those people that do not have a mutation at a gene 2.

How do we calculate this? Bayes theorem tell us that for two random variables X and Y

$$\mathbb{P}(Y|X) = \frac{P(Y, X)}{P(X)} \tag{19}$$

Two variables are said to be **independent** if $\mathbb{P}(Y|X) = P(X)$. This is also true for events. For our purposes, it is more important to understand the process of conditioning with data.

Example 5. *Consider the data $\{1, 2, 2, 1, 3, 4\}$ where each number corresponds to a different configuration of the two genes. Estimate $P(Y_1 = 0|Y_2 = 0)$ from Monte Carlo simulations*

Solution:
First we generate some data

```
> y = np.random.choice([1,2,3,4],p=[1/8,1/8,1/4,1/2])
```

We can then obtain the sample conditioned on Y_2 having the mutation:

```
> y[y ==1 or y==3]
```

The conditional probability would be

```
> len(y_sub[y==1])/len(y_sub)
```

Exercise 6. *Let Y be the value of a 6 sided die, but the one face has been replaced with a two (so there are two, two faces). What is $\mathbb{P}(Y = 3|Y > 2)$? Confirm you answer with Monte Carlo simulations.*

Solution:
This is achieved by the following code:

```
> y = np.random.choice([1,2,3,4,5,6],1000,p=[1/6,1/6,1/6,1/6,1/6,1/6])
> y_sub = y[y >2]
> len(y_sub[y==3])/len(y_sub)
```

5. More random variables

5.1. Binomial: Distribution.

A situation that often arrises is that we take many, say n, independent samples from a Bernoulli distribution. Now let Y be the number of 1s. Then Y follows **binomial distribution**:

$$Y \sim \text{Binomial}(n, q) \tag{20}$$

The binomial distribution has two parameters, k and p, and the probability distribution is

$$\mathbb{P}(Y) = \binom{n}{Y} q^Y (1 - q)^{n-Y} \tag{21}$$

The binomial distribution has a mean qn and variance $nq(1 - q)$. We can draw samples from a binomial distribution

```
> y = np.random.binomial(n,p,n_samples)
```

Exercise 7. *Run simulations of the Binomial distribution and plot the average and standard deviation of the average as a function of n. Also plot a few histogram for different n.*

Solution:
The following code will estimate the mean and standard deviations for various values of q:

```
> def mean_std_bern(n,q):
>   sample = np.random.binomial(n,q,100)
>   return np.mean(sample),np.std(sample)
>
> # generates one hundred evenly spaced numbers between 0 and 1
> q_range = np.linspace(0,1,100)
> means = np.zeros(len(q_range))
> stds = np.zeros(len(q_range))
> for j in range(len(q_range)):
>   means[j],stds[j] = mean_std_bern(q_range[j])
```

Here is a plot of the mean compared to the prediction

```
> fig,ax = plt.subplots(figsize=(5,2))
> ax.plot(q_range,means)
> ax.plot(q_range,q_range,"k--")
```

and the standard deviations

```
> fig,ax = plt.subplots(figsize=(5,2))
> ax.plot(q_range,stds)
> ax.plot(q_range,np.sqrt((1-q_range)*q_range),"k--")
```

5.2. **Normal distribution and the central limit theorem.** In the previous example, we say that if we take the average of many Bernoulli random variables, we get a histogram that looks a lot like a "bell curve" with a standard deviation that scales as $1/\sqrt{n}$.

It turns out this is true when we add up *any* random variables which are sufficiently independent (we will make this precise soon). Since the "bell curve" arrises in the limit where we sum or average many random variables. This motivates us to define the Normal distribution. Unlike previous random variables we've seen, a normal random variable

$$Y \sim \text{Normal}(\mu, \sigma) \tag{22}$$

can take on any number, positive or negative, decimal or integer. We can generate Normal random variables in python with

```
> np.random.normal(0,1)
```

We want to describe this random variable in terms of a probability distribution, but the probability for Y to equal any given value of Y is zero. To see this, note that the probabilities must sum to one In this case, we can describe the probability distribution

$$\mathbb{P}(y_1 < Y < y_2) = \int_{y_1}^{y_2} f(y)dy \tag{23}$$

for a function f called the probability density. It is given by

$$f(y) = \frac{1}{\sqrt{2\pi\sigma}} \, ^{-\frac{-(y-\mu)^2}{2\sigma}}. \tag{24}$$

Example 6. *Suppose that*

$$Y \sim \text{Normal}(5, 2) \tag{25}$$

what is $P(Y > 7)$? Confirm the answer with simulations.

Exercise 8. *Suppose that*

$$Z \sim \text{Normal}(0, 1). \tag{26}$$

In this case we say that Z is a **standard normal** *random variable. What is the chance that*

$$P(5Y - 2 > 3) \tag{27}$$

and confirm you answer with simulations

Going forward, you will need to know the following properties of Normally distributed random variables. Let

$$Y_1 \sim \text{Normal}(0, 1), \quad Y_2 \sim \text{Normal}(0, 1) \tag{28}$$

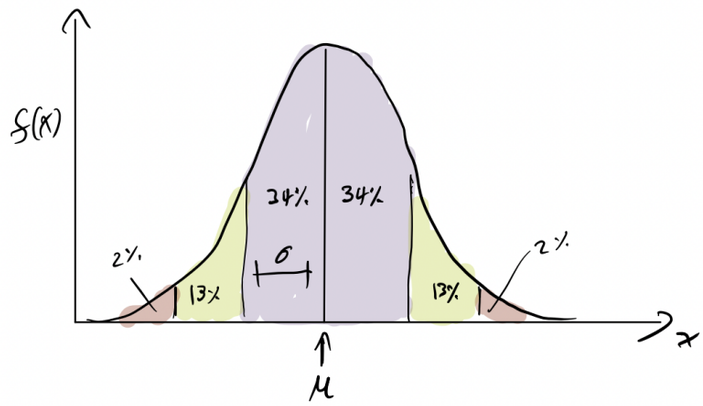

FIGURE 1. Probabilities in the Normal distribution

Then

$$aY_1 + b \sim \text{Normal}(b, a) \tag{29}$$

$$aY_1 + bY_2 \sim \text{Normal}(0, \sqrt{a^2 + b^2}) \tag{30}$$

We now return to the connection to sums of random variables and the Binomial distribution. The **central limit theorem** tells us that for a set of random variables $x_1, \ldots, x_n$ with each have $\mathbb{E}[x_i] = \mu_x$ and $\text{var}(x_i) = \sigma_x^2$,

$$Y = \sum x_i \tag{31}$$

is approximately Normal with mean μ and variance $\sigma/\sqrt{n}$. In other words Normal random variables emerge when we add up many small sources of randomness.

Exercise 9. *Another important distribution is the Beta distribution. A random variable*

$$Y \sim \text{Beta}(\alpha, \beta) \tag{32}$$

is characterized by two parameters, α and β.

```
> np.random.beta(1,2,100)
```

Experiment with the beta distribution and determine how the two arguments influence the shape of the distribution by plotting histograms. To help guide you, consider exploring the questions

1. *What happens if $\alpha = \beta$? How does α change the shape?*
2. *What happens if $\alpha + \beta = c$ is a constant.*