Background Document for Random Numbers

Relevant modules in Python: time, random

Thoughts to start off with: Does it make sense to talk about one single random number? Is the number 43 random?

Definitions and Acronyms

RNG: Random Number Generator

Period of a RNG: the number of pseudo random numbers generated before the sequence starts repeating itself

Pseudo Random Numbers

Random numbers generated by a computer are not really random numbers in the strictest sense, which would require them to be independent from each other. Rather, a computer ALWAYS generates a pseudo random number sequence, which generates numbers that appear to be at random. In essence, a computer uses an algorithm to generate these numbers, with a starting point that is called the seed.

Starting points (Seeding the random number generator)

Pseudo random number generators used in computers will always generate the same sequence of pseudorandom numbers when starting at the same starting point. This is both good and bad. It is good for testing, because then we usually fix the seeds to the same value when initially testing so we can reproduce what happens and to trace errors. It is bad for production code to always generate the same sequence, so in that case we fix the seed to different starting points when running the programs, typically using the time in ms.

How to get the time in ms in Python and set the seed

The starting point of the pseudo random number sequence is determined by the seed value, which has to be an integer. For testing, it really does not matter which value you use. The random module includes the method seed() to set that starting point.

```
>>> import random
>>> random.seed(43)
>>>
```

To get the current time in Python, you need to use the time module. The method time.time() will show seconds since a predefined starting point (you cannot rely on that starting point to be the same across all computers and Python implementations you might use). You will notice that the

number of seconds returned is not an integer value, so you need to convert it to an integer before using it as a seed value. In principle, the time will always increase and therefore, in production code, you will never use the same seed if you use the current time in seconds.

Example use:

```
>>> import time
>>> myTime=time.time()
>>> print(myTime)
1589802413.933423
```

To then set the seed, you take the time in seconds (including the fractional part because that repeats in contrast to the integer part, round it and convert it to an integer.This is an example of a nested function call (more on functions in a few weeks).

```
>>> random.seed(int(round(time.time()*1000)))
>>>
```

Techniques used to find (Pseudo) Random Numbers

- Hardware generator using natural phenomena (impractical to use with user software)
- Published tables (also impractical to use with software)
- Pseudo random number generator using software (easy to integrate) o Middle square method
    - o Linear congruential number generator o Inverse congruential Generator o Compound inversive generators o Many others

    All of the software generators share the characteristics that the numbers they are generating are dependent on the starting point, and that the sequences will repeat after some time. As long as the starting points are chosen carefully (which typically means that they need to differ for different executions of the code) and the sequence of the RNG is long enough (out of your control unless you write your own), they are suitable for most data science applications

Pseudo Random Number generators used in Python

The relevant random number generator used in the random module is the Mersenne Twister. It is not sufficient for security purposes, but it is fine for most data science applications because of its

long period (the number of pseudo random numbers it generates before the sequence repeats). Given the same starting point, it also will generate the same sequence. Not all implementations of all programming languages support the Mersenne Twister.

Some Uses of Pseudo Random Numbers in Data Science

- Monte Carlo approaches o Monte Carlo methods are methods that rely on the use of pseudo random numbers in some way. We will use them to estimate Pi and an integral in our lab. In principle, they work by selecting a random number, evaluating a result, and repeating this process a large number of times to obtain an estimate of a value. o For our lab, we will estimate the value of Pi by selecting random locations from a square area into which we prescribe a circle that is touching the sides of the square. For each of the two dimensional randomly chosen locations (it's important to choose using uniformly distributed random numbers), the location is either inside the circle or not. By relating the numbers of locations chosen inside and outside the circle after selecting a large number of locations), we can get an estimate for Pi
    - o We can get an estimate of the area under a curve by counting random locations that fall above or under the curve. We will use this method to estimate an integral of a function in the lab
- Sampling o When working with large datasets, or datasets with imbalanced class distributions, one often samples to reduce the runtime or to avoid skewing the model towards the majority class (the class to which most of the samples belong). It is important to remember that datasets stored in files are often sorted in some way (by time of collection, some id that might not necessarily be stored in the file as well, by location, etc). It is therefore a good idea to shuffle the data before sampling.

Historical viewpoint: Randu

In the 60's, a random number generator often used was Randu. To quote Donald Knuth from The Art of Computer Programming, Vol 2, "it's very name is enough to bring dismay into the eyes and stomachs of many computer scientists". It was used for many, if not most, scientific papers of that time, with results that were often invalid because of the planes along which the random numbers fall if one is looking at three dimensions or more. It is occasionally still supported and/or implemented and found on IBM mainframes from that time.