How to Generate Random Numbers in Python

The use of randomness is an important part of the configuration and evaluation of machine learning algorithms.

From the random initialization of weights in an artificial neural network, to the splitting of data into random train and test sets, to the random shuffling of a training dataset in stochastic gradient descent, generating random numbers and harnessing randomness is a required skill.

In this tutorial, you will discover how to generate and work with random numbers in Python.

After completing this tutorial, you will know:

- That randomness can be applied in programs via the use of pseudorandom number generators.
- How to generate random numbers and use randomness via the Python standard library.
- How to generate arrays of random numbers via the NumPy library.

Kick-start your project with my new book Statistics for Machine Learning, including *step-by-step tutorials* and the *Python source code* files for all examples.

Let’s get started.

Tutorial Overview

This tutorial is divided into 3 parts; they are:

1. Pseudorandom Number Generators

The source of randomness that we inject into our programs and algorithms is a mathematical trick called a pseudorandom number generator.

A random number generator is a system that generates random numbers from a true source of randomness. Often something physical, such as a Geiger counter, where the results are turned into random numbers. We do not need true randomness in machine learning. Instead we can use pseudorandomness. Pseudorandomness is a sample of numbers that look close to random, but were generated using a deterministic process.

Shuffling data and initializing coefficients with random values use pseudorandom number generators. These little programs are often a function that you can call that will return a random number. Called again, they will return a new random number. Wrapper functions are often also available and allow you to get your randomness as an integer, floating point, within a specific distribution, within a specific range, and so on.

The numbers are generated in a sequence. The sequence is deterministic and is seeded with an initial number. If you do not explicitly seed the pseudorandom number generator, then it may use the current system time in seconds or milliseconds as the seed.

The value of the seed does not matter. Choose anything you wish. What does matter is that the same seeding of the process will result in the same sequence of random numbers.

Let’s make this concrete with some examples.

2. Random Numbers with the Python Standard Library

The Python standard library provides a module called random that offers a suite of functions for generating random numbers.

Python uses a popular and robust pseudorandom number generator called the Mersenne Twister.

In this section, we will look at a number of use cases for generating and using random numbers and randomness with the standard Python API.

Seed The Random Number Generator

The pseudorandom number generator is a mathematical function that generates a sequence of nearly random numbers.

It takes a parameter to start off the sequence, called the seed. The function is deterministic, meaning given the same seed, it will produce the same sequence of numbers every time. The choice of seed does not matter.

The seed() function will seed the pseudorandom number generator, taking an integer value as an argument, such as 1 or 7. If the seed() function is not called prior to using randomness, the default is to use the current system time in milliseconds from epoch (1970).

The example below demonstrates seeding the pseudorandom number generator, generates some random numbers, and shows that reseeding the generator will result in the same sequence of numbers being generated.

```
# seed the pseudorandom number generator
from random import seed
from random import random
# seed random number generator
seed(1)
# generate some random numbers
print(random(), random(), random())
# reset the seed
seed(1)
# generate some random numbers
print(random(), random(), random())
```

Running the example seeds the pseudorandom number generator with the value 1, generates 3 random numbers, reseeds the generator, and shows that the same three random numbers are generated.

```
0.13436424411240122 0.8474337369372327 0.763774618976614
0.13436424411240122 0.8474337369372327 0.763774618976614
```

It can be useful to control the randomness by setting the seed to ensure that your code produces the same result each time, such as in a production model.

For running experiments where randomization is used to control for confounding variables, a different seed may be used for each experimental run.

Random Floating Point Values

Random floating point values can be generated using the random()function. Values will be generated in the range between 0 and 1, specifically in the interval [0,1).

Values are drawn from a uniform distribution, meaning each value has an equal chance of being drawn.

The example below generates 10 random floating point values.

```
# generate random floating point values
from random import seed
from random import random
# seed random number generator
seed(1)
# generate random numbers between 0-1
for _ in range(10):
    value = random()
    print(value)
```

Running the example generates and prints each random floating point value.

```
0.13436424411240122
0.8474337369372327
0.763774618976614
0.2550690257394217
0.49543508709194095
0.4494910647887381
0.651592972722763
```

```
0.7887233511355132
0.0938595867742349
0.02834747652200631
```

The floating point values could be rescaled to a desired range by multiplying them by the size of the new range and adding the min value, as follows:

```
scaled value = min + (value * (max - min))
```

Where *min* and *max* are the minimum and maximum values of the desired range respectively, and *value* is the randomly generated floating point value in the range between 0 and 1.

Random Integer Values

Random integer values can be generated with the randint() function.

This function takes two arguments: the start and the end of the range for the generated integer values. Random integers are generated within and including the start and end of range values, specifically in the interval [start, end]. Random values are drawn from a uniform distribution.

The example below generates 10 random integer values between 0 and 10.

```
# generate random integer values
from random import seed
from random import randint
# seed random number generator
seed(1)
# generate some integers
for _ in range(10):
    value = randint(0, 10)
    print(value)
```

Running the example generates and prints 10 random integer values.

```
2
9
1
4
1
7
7
7
10
6
```

Random Gaussian Values

Random floating point values can be drawn from a Gaussian distribution using the gauss() function.

This function takes two arguments that correspond to the parameters that control the size of the distribution, specifically the mean and the standard deviation.

The example below generates 10 random values drawn from a Gaussian distribution with a mean of 0.0 and a standard deviation of 1.0.

Note that these parameters are not the bounds on the values and that the spread of the values will be controlled by the bell shape of the distribution, in this case proportionately likely above and below 0.0.

```
# generate random Gaussian values
from random import seed
from random import gauss
# seed random number generator
seed(1)
# generate some Gaussian values
for _ in range(10):
    value = gauss(0, 1)
    print(value)
```

Running the example generates and prints 10 Gaussian random values.

```
1.2881847531554629
1.449445608699771
0.06633580893826191
-0.7645436509716318
-1.0921732151041414
0.03133451683171687
-1.022103170010873
-1.4368294451025299
0.19931197648375384
0.13337460465860485
```

Randomly Choosing From a List

Random numbers can be used to randomly choose an item from a list.

For example, if a list had 10 items with indexes between 0 and 9, then you could generate a random integer between 0 and 9 and use it to randomly select an item from the list. The choice() function implements this behavior for you. Selections are made with a uniform likelihood.

The example below generates a list of 20 integers and gives five examples of choosing one random item from the list.

```
# choose a random element from a list
from random import seed
from random import choice
# seed random number generator
seed(1)
# prepare a sequence
sequence = [i for i in range(20)]
print(sequence)
# make choices from the sequence
for _ in range(5):
    selection = choice(sequence)
    print(selection)
```

Running the example first prints the list of integer values, followed by five examples of choosing and printing a random value from the list.

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 1
```

```
4
18
2
8
3
```

Random Subsample From a List

We may be interested in repeating the random selection of items from a list to create a randomly chosen subset.

Importantly, once an item is selected from the list and added to the subset, it should not be added again. This is called selection without replacement because once an item from the list is selected for the subset, it is not added back to the original list (i.e. is not made available for re-selection).

This behavior is provided in the sample*()* function that selects a random sample from a list without replacement. The function takes both the list and the size of the subset to select as arguments. Note that items are not actually removed from the original list, only selected into a copy of the list.

The example below demonstrates selecting a subset of five items from a list of 20 integers.

```
# select a random sample without replacement
from random import seed
from random import sample
# seed random number generator
seed(1)
# prepare a sequence
sequence = [i for i in range(20)]
print(sequence)
# select a subset without replacement
subset = sample(sequence, 5)
print(subset)
```

Running the example first prints the list of integer values, then the random sample is chosen and printed for comparison.

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 1
[4, 18, 2, 8, 3]
```

Randomly Shuffle a List

Randomness can be used to shuffle a list of items, like shuffling a deck of cards.

The *shuffle()* function can be used to shuffle a list. The shuffle is performed in place, meaning that the list provided as an argument to the shuffle() function is shuffled rather than a shuffled copy of the list being made and returned.

The example below demonstrates randomly shuffling a list of integer values.

```
# randomly shuffle a sequence
from random import seed
from random import shuffle
# seed random number generator
seed(1)
# prepare a sequence
sequence = [i for i in range(20)]
print(sequence)
# randomly shuffle the sequence
shuffle(sequence)
print(sequence)
```

Running the example first prints the list of integers, then the same list after it has been randomly shuffled.

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 1
[11, 5, 17, 19, 9, 0, 16, 1, 15, 6, 10, 13, 14, 12, 7, 3, 8,
```

3. Random Numbers with NumPy

In machine learning, you are likely using libraries such as scikit-learn and Keras.

These libraries make use of NumPy under the covers, a library that makes working with vectors and matrices of numbers very efficient.

NumPy also has its own implementation of a pseudorandom number generator and convenience wrapper functions.

NumPy also implements the Mersenne Twister pseudorandom number generator.

Let’s look at a few examples of generating random numbers and using randomness with NumPy arrays.

Seed The Random Number Generator

The NumPy pseudorandom number generator is different from the Python standard library pseudorandom number generator.

Importantly, seeding the Python pseudorandom number generator does not impact the NumPy pseudorandom number generator. It must be seeded and used separately.

The seed() function can be used to seed the NumPy pseudorandom number generator, taking an integer as the seed value.

The example below demonstrates how to seed the generator and how reseeding the generator will result in the same sequence of random numbers being generated.

```
# seed the pseudorandom number generator
from numpy.random import seed
from numpy.random import rand
# seed random number generator
seed(1)
```

```
# generate some random numbers
print(rand(3))
# reset the seed
seed(1)
# generate some random numbers
print(rand(3))
```

Running the example seeds the pseudorandom number generator, prints a sequence of random numbers, then reseeds the generator showing that the exact same sequence of random numbers is generated.

```
[4.17022005e-01 7.20324493e-01 1.14374817e-04]
[4.17022005e-01 7.20324493e-01 1.14374817e-04]
```

Array of Random Floating Point Values

An array of random floating point values can be generated with the rand() NumPy function.

If no argument is provided, then a single random value is created, otherwise the size of the array can be specified.

The example below creates an array of 10 random floating point values drawn from a uniform distribution.

```
# generate random floating point values
from numpy.random import seed
from numpy.random import rand
# seed random number generator
seed(1)
# generate random numbers between 0-1
values = rand(10)
print(values)
```

Running the example generates and prints the NumPy array of random floating point values.

```
[4.17022005e-01 7.20324493e-01 1.14374817e-04 3.02332573e-01
 1.46755891e-01 9.23385948e-02 1.86260211e-01 3.45560727e-01
 3.96767474e-01 5.38816734e-01]
```

Array of Random Integer Values

An array of random integers can be generated using the randint()NumPy function.

This function takes three arguments, the lower end of the range, the upper end of the range, and the number of integer values to generate or the size of the array. Random integers will be drawn from a uniform distribution including the lower value and excluding the upper value, e.g. in the interval [lower, upper).

The example below demonstrates generating an array of random integers.

```
# generate random integer values
from numpy.random import seed
from numpy.random import randint
# seed random number generator
seed(1)
# generate some integers
values = randint(0, 10, 20)
print(values)
```

Running the example generates and prints an array of 20 random integer values between 0 and 10.

```
[5 8 9 5 0 0 1 7 6 9 2 4 5 2 4 2 4 7 7 9]
```

Array of Random Gaussian Values

An array of random Gaussian values can be generated using the randn() NumPy function.

This function takes a single argument to specify the size of the resulting array. The Gaussian values are drawn from a standard

Gaussian distribution; this is a distribution that has a mean of 0.0 and a standard deviation of 1.0.

The example below shows how to generate an array of random Gaussian values.

```
# generate random Gaussian values
from numpy.random import seed
from numpy.random import randn
# seed random number generator
seed(1)
# generate some Gaussian values
values = randn(10)
print(values)
```

Running the example generates and prints an array of 10 random values from a standard Gaussian distribution.

```
[ 1.62434536 -0.61175641 -0.52817175 -1.07296862  0.86540763
  1.74481176 -0.7612069   0.3190391  -0.24937038]
```

Values from a standard Gaussian distribution can be scaled by multiplying the value by the standard deviation and adding the mean from the desired scaled distribution. For example:

```
scaled value = mean + value * stdev
```

Where *mean* and *stdev* are the mean and standard deviation for the desired scaled Gaussian distribution and *value* is the randomly generated value from a standard Gaussian distribution.

Shuffle NumPy Array

A NumPy array can be randomly shuffled in-place using the shuffle()NumPy function.

The example below demonstrates how to shuffle a NumPy array.

```
# randomly shuffle a sequence
from numpy.random import seed
from numpy.random import shuffle
# seed random number generator
seed(1)
# prepare a sequence
sequence = [i for i in range(20)]
print(sequence)
# randomly shuffle the sequence
shuffle(sequence)
print(sequence)
```

Running the example first generates a list of 20 integer values, then shuffles and prints the shuffled array.

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 1
[3, 16, 6, 10, 2, 14, 4, 17, 7, 1, 13, 0, 19, 18, 9, 15, 8,
```

Further Reading

This section provides more resources on the topic if you are looking to go deeper.

- Embrace Randomness in Machine Learning
- random – Generate pseudo-random numbers
- Random sampling in NumPy
- Pseudorandom number generator on Wikipedia

Summary

In this tutorial, you discovered how to generate and work with random numbers in Python.

Specifically, you learned:

- That randomness can be applied in programs via the use of pseudorandom number generators.

- How to generate random numbers and use randomness via the Python standard library.
- How to generate arrays of random numbers via the NumPy library.